Recent Directions in Netlist Partitioning: A Survey*

Charles J. Alpert and Andrew B. Kahng

UCLA Computer Science Department, Los Angeles, CA 90024-1596

Abstract

This survey describes research directions in netlist partitioning during the past two decades, in terms of both problem formulations and solution approaches. We discuss the traditional min-cut and ratio cut bipartitioning formulations along with multi-way extensions and newer problem formulations, e.g., constraint-driven partitioning (for FPGAs) and partitioning with module replication. Our discussion of solution approaches is divided into four major categories: **move-based** approaches, **geometric representations**, **combinatorial** formulations, and **clustering** approaches. Move-based algorithms iteratively explore the space of feasible solutions according to a *neighborhood operator*; such methods include greed, iterative exchange, simulated annealing, and evolutionary algorithms. Algorithms based on geometric representations embed the circuit netlist in some type of "geometry", e.g, a 1-dimensional linear ordering or a multi-dimensional vector space; the embeddings are commonly constructed using *spectral* methods. Combinatorial methods transform the partitioning problem into another type of optimization, e.g., based on network flows or mathematical programming. Finally, clustering algorithms merge the netlist modules into many small clusters; we discuss methods which combine clustering with existing algorithms (e.g., two-phase partitioning). The paper concludes with a discussion of benchmarking in the VLSI CAD partitioning literature and some perspectives on more promising directions for future work.

1 Introduction

The essence of netlist partitioning is to divide a system specification into clusters such that the number of intercluster connections is minimized. The partitioning task is ubiquitous to many subfields of VLSI CAD: any top-down hierarchical (i.e., "divide and conquer") approach to system design must rely on some underlying partitioning technique. There are several reasons why partitioning has recently emerged as a critical optimization in many phases of VLSI system synthesis, and why the past several years have seen so much research activity on the subject.

Above all, partitioning heuristics are used to address the increasing complexity of VLSI design. Systems with several million transistors are now common, presenting instance complexities that are unmanageable for existing logic-level and physical-level design tools. Partitioning divides a system

*Partial support for this work was provided by NSF Young Investigator Award MIP-9257982.

into smaller, more manageable components; the number of signals which pass between the components corresponds to the interactions between the design subproblems. In a top-down hierarchical design methodology, decisions made early in the system synthesis process (e.g., at the system and chip levels) will constrain succeeding decisions. Thus, the feasibility – not to mention the quality – of automatic placement, global routing and detailed routing depends on the quality of the partitioning solution. A bottom-up clustering may also be applied to decrease the size of the design, typically in cell- or gate-level layout. The current emphasis on a quick-turnaround ASIC design cycle reinforces the need for reliable and effective algorithms.

Partitioning heuristics also have a greater impact on system performance as designs become interconnect-dominated. In current submicron designs, wire delays tend to dominate gate delays [15]; the differences between on-chip and off-chip signal delays and the increasingly pin-limited nature of large chips make it desirable to minimize the number of signals traveling off a given chip. Larger die sizes imply that long on-chip global routes between function blocks will more noticeably affect system performance. Other considerations (e.g., design for testability, low-power design, etc.) also require partitioning algorithms to identify interconnect structure, albeit at more of a functional or communication-based level.

Finally, partitioning heuristics affect the layout area: wires between clusters at high levels of the hierarchy will tend to be longer than wires between clusters at lower levels, and total wirelength is directly proportional to layout area due to minimum wire spacing design rules. The traditional minimum-cut objective is natural for this application: if the layout area is divided into a dense uniform grid, total wirelength can be expressed in "grid" units or equivalently as the sum over all gridlines of the number of wires crossing each gridline. This view can also improve auto-routability since it suggests reducing the wire congestion in any given layout region.

All of these considerations motivate the development of netlist partitioning tools that identify interconnection and communication structure in a given system design. Indeed, one of the five-year predictions issued by participants at the 1991 CANDE workshop [142] was that stand-alone partitioning for every phase of system synthesis would comprise the next major class of CAD tools to emerge in the marketplace. That this prediction has come to pass is evident from the various new tools and startup companies which focus almost exclusively on partitioning.[1]

Today, leading applications of partitioning include:

- **General design packaging.** Logic must often be partitioned into clusters, subject to constraints

[1]Two notable examples are: High Level Design Systems, whose "advanced design planner" performs design planning at the floorplan level, and ACEO, whose "SoftWire" tool performs communication-based partitioning of multiple-FPGA systems. Partitioning is also at the heart of new tools for system-level design optimization and mapping of designs onto emulation or prototyping architectures composed of multiple field-programmable gate arrays (FPGAs).

on cluster area as well as possible I/O bounds. This problem is known as *design packaging* and is still the canonical partitioning application; it arises not only in chip floorplanning and placement, but also at all other levels of the system design.[2] The design packaging problem also arises whenever technology improves and existing designs must be repackaged onto higher-capacity modules ("technology migration"). Note that the problem is usually associated with large cluster sizes, with few constraints on the internal structure of the clusters. When a finite library of available module types is specified, the optimization is more along the lines of "covering" or "technology mapping".

- **Netlist-level partitioning in HDL-based synthesis.** Synthesis tools have emerged which reduce the design cycle by automatically mapping a high-level functional description to a gate- or cell-level netlist. However, even with increased maturity of such high-level synthesis tools, netlist partitioning remains central to the success of the design procedure. This is essentially because writing HDL code – as opposed to performing layout – can abstract away the physical layout effects of design choices (for example, a few lines of HDL code that specify a register file or crossbar connection can correspond to a large portion of the final layout area). As a result, the block decomposition of the functional (software) description does not necessarily map well into a decomposition of the physical layout. Hence, in contrast to previous building-block methodologies which yielded a small number of function blocks that could be optimally hand-partitioned, HDL-based synthesis virtually requires the physical design methodology to shift from working with a small number of building blocks to working with large, flattened design representations [53]. Partitioning of flattened inputs is also necessary for such applications as the design of "precursor systems" (i.e., finding the packaging tradeoffs that correspond to optimum cost-performance points at early stages in the product life cycle).

- **Estimation for design optimization.** Accurate estimation of layout area and wireability has always been a critical element of high-level synthesis and floorplanning. Now, such estimates are becoming critical to higher-level searches over the system design space. Predictive models often combine analysis of the netlist partitioning structure with analysis of the output characteristics of placement and routing algorithms, in order to yield estimates of wiring requirements and system performance. This use of system partitioning hierarchies is increasingly prominent as "design optimization" and "electronic *system* design automation" capture the attention of CAD users and vendors.

- **System emulation and rapid prototyping (FPGA partitioning).** Many logic emulation

[2]Despite the existence of partitioning applications throughout the system design cycle, calls for papers of major CAD conferences still implicitly classify partitioning within placement and/or floorplanning. Interestingly, research activity in "pure partitioning" arguably exceeds that in either of these mainstay categories.

systems and rapid system prototyping methodologies (e.g., those from Quickturn or Zycad/Inca) use partitioning tools to map complex circuit designs onto hundreds or even thousands of interconnected FPGAs. Typically, such partitioning instances are challenging because the timing, area, and I/O resource utilizations must satisfy hard device-specific constraints. Furthermore, the partitioning optimization is affected by the discrete nature of system resources – e.g., interconnect delay in routing segments, layout area in configurable logic blocks (CLBs), or individual FPGA chips in a multiple-FPGA system – all of which have large "quanta".

- **Hardware simulation and test.** A good partitioning will minimize the number of inter-block signals that must be multiplexed onto the bus architecture of a hardware simulator or mapped to the global interconnect architecture of a hardware emulator. Reducing the number of inputs to a block often reduces the number of test vectors needed to exercise the logic.

In this work, we survey the major research directions in netlist partitioning and establish a taxonomy of existing works based on the underlying solution methodology. While we have tried to make this survey both complete and self-contained, we emphasize more recent problem formulations and solution approaches, possibly at the expense of methods that have been treated in previous surveys.[3] Section 2 develops notation, discusses various graph and hypergraph representations of the circuit netlist, and formulates basic variants of the partitioning problem. These problem formulations include bipartitioning, multi-way partitioning, constraint-driven partitioning, and partitioning with replication. Sections 3 through 6 survey four major categories of partitioning approaches:

- Section 3 discusses **move-based** approaches: we classify an algorithm in this category if it explores the solution space by moving from one solution to another. Greedy and iterative exchange [117] approaches are most common – these always try to make the best move, but can easily can trapped in local minima. To avoid this behavior, many other strategies have been proposed including stochastic hill-climbing (simulated annealing), evolutionary algorithms, and the multistart strategy. We discuss these approaches along with many adaptations that have used these methods to address more complex formulations.

- Section 4 discusses methods that construct a **geometric representation** of the partitioning problem via such constructions as a 1-dimensional linear ordering or a multi-dimensional vector space. Such a representation offers possibilities for geometric approaches to solve problems that are intractable for general graphs. *Spectral* methods are commonly used to construct geometric representations, due to their ability to capture global netlist information.

[3]Limited surveys are given in the textbooks [149] and [171]; the former contains a more personal perspective on early works, by W. Donath. The book by Lengauer [133] is noteworthy, especially for its complete development of combinatorial algorithms (network flow, multicommodity flow, etc.).

- Section 5 discusses **combinatorial** approaches; we loosely classify an approach under this category if the partitioning problem can be transformed into some other "classic" type of optimization, e.g., maximum flow, mathematical programming, graph labeling, or set covering. These approaches are promising since complex formulations that include timing, module preassignment, replication, and other hard constraints can often readily be expressed in terms of a mathematical program or flow network. In addition, changing user requirements for solution quality and runtime, as well as improved computing platforms, have made such approaches more practical.

- Section 6 treats **clustering**-based approaches, which traditionally consist of bottom-up approaches that merge netlist modules into small clusters. We augment this class to include methods which use a clustering solution within another algorithm, such as two-phase partitioning or placement. Clustering-based approaches have received much recent attention since they are viewed as the most promising method for tackling the increasing problem sizes in VLSI CAD.

Section 7 concludes with a discussion of benchmarking practice, as well as a brief list of perspectives on future research in the field.

2 Partitioning Formulations

In this section, we describe the major variant formulations of the partitioning problem. We defer the description of several less well-studied variants – e.g., retiming formulations – to the discussions of their respective underlying solution strategies.

2.1 Preliminaries

Given a set of n netlist modules $V = \{v_1, v_2, \ldots, v_n\}$, the purpose of partitioning is to assign the modules to a specified number k of clusters satisfying prescribed properties.[4]

Definition: A *k-way partitioning* $P^k = \{C_1, C_2, \ldots, C_k\}$ consists of k *clusters* (subsets of V), $C_1, C_2, \ldots, C_k$, such that $C_1 \cup C_2 \cup \ldots C_k = V$. If $k = 2$, we refer to P^2 as a *bipartitioning*.

The objective to be optimized is denoted by $F(P^k)$, i.e., the objective is a function of the partitioning solution. We generally make the traditional assumption that the clusters are mutually disjoint; note however that *replication* formulations permit a module to be a member of more than one cluster.

[4]Many works distinguish between the partitioning problem where k is small (e.g., $k \leq 15$) and the clustering problem where k is large (e.g., $k = \Theta(n)$). Such works may refer to what we call as cluster as a "partition"; we choose not to make this distinction, since our multi-way partitioning formulations are independent of the relative size of k. For reasons of clarity, we will generally refer to P^k as a "partitioning" for small k and as a "clustering" for large k.

The most common method for representing the circuit netlist connections is as a hypergraph $H(V,E)$ with $E = \{e_1, e_2, \ldots, e_m\}$ being the set of signal nets (see, e.g., [23] for basic concepts of graphs and hypergraphs). Each net is a subset of V containing the modules that that net connects, and we assume that for each $e \in E$, $|e| \geq 2$. The equivalence between netlists and hypergraphs is exact if each net has at most one pin on any module. The modules in e may also be called the *pins* of e. We also assume a weighting function $w : V \rightarrow \Re$, generally used for the area of each module. The weighting function can be extended to clusters, i.e., $w(C) = \sum_{v \in C} w(v)$. Another weighting function $w' : E \rightarrow \Re$ can be defined for nets (e.g., to give higher weights for critical nets or input-output paths), but for ease of presentation we omit net-weighting from our discussion.[5]

Definition: For each module v, the set of nets *incident* to v is denoted by $N(v) = \{e \in E \mid v \in e\}$ and the set of modules that are *neighbors* of v is denoted by $M(v) = \{w \in V \mid \exists\ e\ v, w \in e, v \neq w\}$. We say that $deg(v) = |N(v)|$ is the *degree* of v, and $deg_{min} = \min_{v \in V} deg(v)$ and $deg_{max} = \max_{v \in V} deg(v)$ are respectively the minimum and maximum degrees of the hypergraph.

Definition: Each signal net e consists of a single *source* module $S(e)$ and a set of *destination* modules $D(e)$ (so $\{S(e)\} \cup D(e) = e$), which indicates the direction of signal flow.

Many algorithms either ignore source and destination information or assume it is not available.

Definition: The set of hyperedges *cut* by a cluster C is given by $E(C) = \{e \in E\ s.t.\ 0 < | e \cap C | < | e |\}$, i.e., $e \in E(C)$ if at least one, but not all, of the pins of e are in C. The set of nets cut by a partitioning solution P^k can be expressed as $E(P^k) = \cup_{h=1}^{k} E(C_i)$ or equivalently $E(P^k) = \{e \in E \mid \exists u, v \in e, h \neq l\ with\ u \in C_h\ and\ v \in C_l\}$. We say that $|E(P^k)|$ is the *cutsize* of P^k.

Sometimes, it may be easier to represent a partitioning solution in terms of vectors and matrices, hence our final definition:

Definition: Given P^k, the *indicator vector* for cluster C_h is the n-dimensional vector $\vec{X_h} = (x_{ih})$ with $x_{ih} = 1$ if $v_i \in C_h$ and $x_{ih} = 0$ if $v_i \notin C_h$. The $n \times k$ matrix X with column h equal to $\vec{X_h}$ is the *assignment matrix* for P^k.

When $k = 2$, we have $\vec{X_1} = \vec{1} - \vec{X_2}$, so only one indicator vector is needed. For this case, we let $\vec{x} = \vec{X_2}$ represent the bipartitioning solution.

[5]Useful information regarding the subcircuit function and the design hierarchy may be inferred from the module uniquenesses (module ID's), depending on how other development tools represent and output intermediate design representations. Current works in partitioning do not assume the availability of such information, but this may change in light of new applications such as functional clustering.

2.2 Circuit Representations

The choice of netlist representation is typically a consequence of the objective or the algorithmic approach, e.g., minimizing cut nets implicitly requires a hypergraph representation, and a maximum-flow solution for replication cut will require a directed network (graph) representation. Aside from the hypergraph model discussed above, standard netlist representations include:

- **Weighted Undirected Graph.** A graph $G = (V, E)$ is a special case of a hypergraph with all $|e_i| \equiv 2$. When certain standard matrix computations or algorithm implementations are used (e.g., spectral computations or mathematical programming), an undirected graph representation is often convenient. The netlist is represented by a symmetric $n \times n$ *adjacency matrix* $A = (a_{ij})$ in which the matrix entry $a_{ij} \geq 0$ captures the connectivity between modules v_i and v_j (so $a_{ii} \equiv 0$ is typically assumed). Generally, for the adjacency matrix is useful only when it is *sparse*, i.e., $a_{ij} > 0$ for a very small percentage of the matrix entries.

  To construct a graph from a netlist, the *clique* net model is often used: a signal net e contains $|e|$ pins and will induce an edge between every pair of its pins; each edge has a weight that is a function of $|e|$ (multiple edges with total weight W connecting a pair of modules are contracted into a single edge with weight W). The "standard" clique net model [133] assigns uniform weight $\frac{1}{|e|-1}$ to each clique edge, although many other weighting functions have been proposed.[6] It has been noted that the clique net model may enable a "finer-grain optimization" than the hypergraph

[6]Ideally, no matter how modules of the clique are partitioned, the cost should be one, corresponding to a single cut of a net. Ihler et al. [106] prove that such a "perfect" clique net model is impossible to achieve. In addition, Lengauer [133] shows that no matter what weighting function is used, there exists a bipartitioning with deviation $\Omega(\sqrt{|e|})$ from the desired cost of cutting a single net. The standard clique model ensures that for every signal net cut in a partitioning solution, the total weight of cut edges for that net will be at least one. However, since a cut net can contribute up to $\frac{|e|^2}{4} \cdot \frac{1}{(|e|-1)}$ to the partitioning objective, large nets are less likely to be cut than smaller nets. A weight of $\frac{4}{|e|(|e|-1)}$ was proposed by D. J.-H. Huang and adopted by [3] so that the *expected* weight of a cut signal net would be one. Hadley et al. [79] propose a weighting scheme that is a function of both $|e|$ and k which guarantees that for the cost of splitting e into k clusters will be bounded above by one. For the *maximum* cost of a net cut in the transformed graph to be one, Donath [58] shows that the appropriate edge weighting is $\frac{4}{|e|^2-(|e| \bmod 2)}$ since there are $\frac{|e|^2-(|e| \bmod 2)}{4}$ edges crossing between two clusters when half the modules of e are in each cluster (also see [184]). Donath further shows that when there are more than two clusters, $\frac{4}{|e|^2-(|e| \bmod 2)}$ remains the correct uniform weighting to achieve this upper bound. Many other net models are motivated by 1- and 2-dimensional cell placement. For example, the standard $\frac{1}{|e|-1}$ weight is inspired by linear placement into fixed slots separated by distance one [40]: since the minimum wirelength of a $|e|$-pin net must be at least $|e|-1$, the weighting should be inversely proportional to $|e|-1$ so that the objective does not "try too hard" to place the modules into an impossibly small span of slots (also see [81]). If the span of a net is exactly $|e|-1$, then D. J.-H. Huang showed that a weighting function of $\frac{6}{|e|(|e|+1)}$ gives the total wirelength $|e|-1$ for the clique representation. In a 2-dimensional layout, [91] assumes that the net will be a spanning tree with $|e|-1$ edges; hence, if the weight for these edges is evenly distributed among the clique edges, the weight function should be $(|e|-1)/\binom{|e|}{2} = \frac{2}{|e|}$. Frankle and Karp [68] proposed the uniform weight $(\frac{2}{|e|})^{\frac{3}{2}}$ for linear placement with minimum squared wirelength: if the span of a net is normalized to one and edges have weight w, the total edge cost may vary from $\frac{|e|}{2} \cdot w$ to $(\frac{|e|}{2})^2 \cdot w$ depending on the distribution of pins; thus, Frankle and Karp set $w = (\frac{2}{|e|})^{\frac{3}{2}}$ to minimize the worst-case deviation of cost from the square of the span (this model has also been used by [37]). Tsay and Kuh [183] propose edge weight $\frac{2}{|e|}$ for minimizing squared wirelength and $(\frac{2}{|e|})^3$ for minimizing Manhattan wirelength.

model, e.g., Shih [172] reports that solution quality of Fiduccia-Mattheyses bipartitioning [65] can improve when run on the graph representation of the netlist instead of the original hypergraph. However, the clique model can destroy the natural sparsity of the netlist since $\binom{|e|}{2}$ nonzeros will be inserted in A for every net e. For example, a 1000-pin clock net will induce 499,500 nonzero entries in A. Thus, some existing methods discard large nets to maintain sparsity [37] [79]. An undirected graph can also be induced from a netlist by constructing random spanning trees, paths, or cycles over the pins in every signal net. Alternatively, dummy modules may be inserted [106], e.g., the "star" model adds a dummy node for each net and connects every pin of the net to this dummy node (cf. the discussion of [99] in Section 5.2).

- **Intersection Graph:** One can view the netlist partitioning problem in terms of partitioning nets instead of modules. Given such a perspective, a useful netlist representation is the *intersection graph* $G'(V', E)$, whose vertices correspond to the signal nets of the original netlist hypergraph, i.e., $V' \equiv \{e_1, e_2, \ldots, e_m\}$. Two vertices $e_i, e_j \in V'$ are adjacent if and only if $e_i \cap e_j \neq \emptyset$, i.e., the nets share a module. Early uses of this representation were in both partitioning [112] and placement [146]. Note that the intersection graph is typically quite sparse due to module fanout limits [80].

- **Dual Hypergraph:** A slightly different net-oriented representation is the *dual hypergraph* $H(V', E')$, which has the same vertex set $V' \equiv \{e_1, e_2, \ldots, e_m\}$ as the intersection graph. However, the connections are slightly different in that modules are mapped to hyperedges: For each $v \in V$, the hyperedge $N(v)$ is added to E' (note $|E'| = n$) – recall that $N(v)$ is the set of all nets incident to v. Observe that the intersection graph can be derived by applying the clique net model to the dual hypergraph. Yeh et al. [194] used this representation for their Primal-Dual FM based algorithm (see Section 3.5), and [49] used a combination of the intersection and dual representations.

- **Directed Graph:** Signal flow direction can be directly integrated into a graph representation, e.g., by creating directed edges $(S(e), w)$ for every $e \in E$ and every $w \in D(e)$. This specific construction is called the *directed tree* representation. A directed graph is particularly useful for flow, uni-directional cut, timing, and replication formulations. Sometimes this construction yields a *directed acyclic graph* (DAG), e.g., for combinational logic networks [50] [107]. in such a case, the set of *primary inputs* is denoted by $PI = \{v \in V \mid \forall\ e \in N(v), S(e) = v\}$ and the set of *primary outputs* is denoted by $PO = \{v \in V \mid \forall\ e \in N(v), S(e) = v\}$.

These five models are illustrated in Figure 1. In the remainder of the discussion, we will assume that the circuit is represented as a hypergraph. Of course, any method for hypergraph partitioning can be also applied to the graph representation, as well as to the intersection and dual representations if a scheme is provided to transform the resulting net partitioning into a module partitioning.

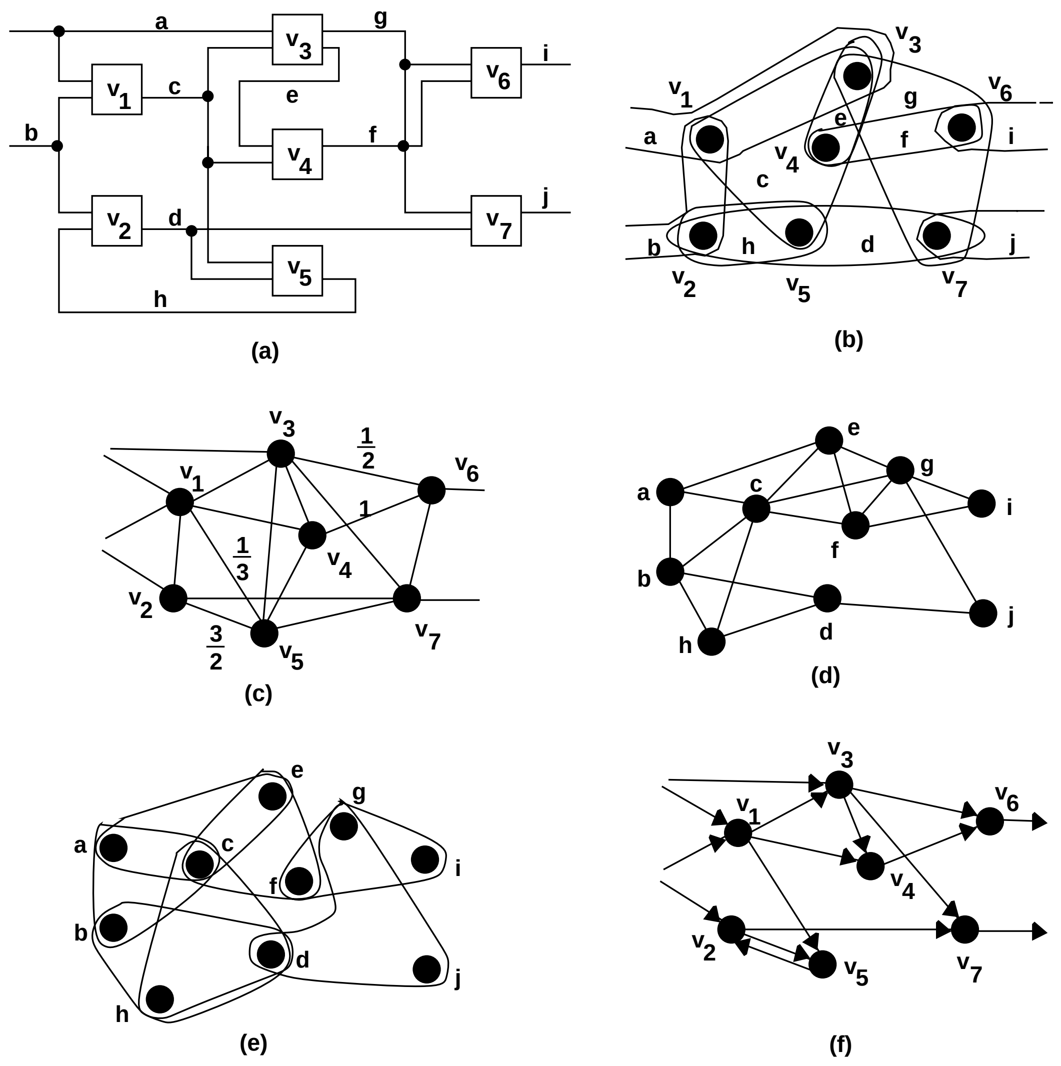


Figure 1: Representations of a circuit with 7 modules and 10 signal nets: (a) circuit diagram with all inputs on the left side of the modules and all outputs on the right side; (b) the hypergraph representation; (c) the weighted graph representation using the standard clique net model with uniform edge weight $\frac{1}{|e|-1}$ (only some edge weights are shown); (d) the intersection graph; (e) the dual graph; and (f) the directed graph (assuming a directed-tree hyperedge model). To enhance readability, not all edges have been labeled.

2.3 Bipartitioning Formulations

The min-cut bipartitioning problem seeks to divide V into two clusters such that the number of hyperedges cut by the clusters is minimized:

Min-Cut Bipartitioning: Minimize $F(P^2) = |E(C_1)| = |E(C_2)|$ such that $C_1 \neq \emptyset$, $C_2 \neq \emptyset$.

Min-Cut Bipartitioning can be solved by converting the hypergraph to a flow network, computing a certain set of $n-1$ flows and applying the max-flow min-cut theorem [66] to obtain a minimum cut. Many algorithms are known which can solve the max-flow problem in polynomial time; see [75] [2] for surveys. Note however that finding a minimum cut does not necessarily require a maximum flow; fast techniques to find minimum cuts [92] [114] [141] are noted in Section 5 below.

Optimal solutions to Min-Cut Bipartitioning will often be quite unbalanced and are thus not useful within a hierarchical design methodology; however, these solutions may form the basis of a useful heuristic [190] [191]. A more practical formulation seeks minimum-cut bipartitionings with bounds on module cardinality or total module area within each cluster. The *Min-Cut Bisection* problem seeks two equal-weight clusters:

Min-Cut Bisection: Minimize $F(P^2) = |E(C_1)|$ such that $|w(C_1) - w(C_2)| \leq \epsilon$.

If all modules have unit weight, then $\epsilon = 1$. When move-based heuristics are applied, it is usually convenient to allow the cluster weight imbalance ϵ to vary up to the largest module weight (otherwise the number of possible moves may become too limited). Min-Cut Bisection is NP-complete [70], as are all of the other size-constrained formulations that we discuss since they reduce to Min-Cut Bisection.

This formulation may be unnecessarily restrictive, and relaxing the size constraints may permit a much better solution while still maintaining relatively balanced clusters.

Size-Constrained Min-Cut Bipartitioning: Given prescribed lower and upper cluster size lower and upper bounds L and U, minimize $F(P^2) = |E(C_1)|$ such that $L \leq w(C_h) \leq U$ for $h = 1, 2$.

This formulation has become popular in the recent literature, both for its greater practical relevance and as an added basis for algorithm comparisons, e.g., [156] [190] present results using unit module areas and $L = \frac{9n}{20}$, $U = \frac{11n}{20}$, and [188] [84] present results using actual module areas and $L = \frac{1}{4}\sum_i w(v_i)$, $U = \frac{3}{4}\sum_i w(v_i)$ (see Section 7).

Rather than minimizing cutsize subject to cluster size constraints, the cutsize and balance criteria can be smoothly intergrated into the partitioning objective. The concept of *ratio cut* partitioning was introduced in [132] and first applied to circuit partitioning by Wei and Cheng [187].

Minimum Ratio Cut Bipartitioning: Minimize $F(P^2) = \frac{|E(C_1)|}{w(C_1) \cdot w(C_2)}$.

The numerator favors a low cutsize while the denominator favors more balanced cluster sizes. Figure 2 contrasts the optimal solutions for the Min-Cut Bipartitioning, Min-Cut Bisection and Minimum Ratio Cut Bipartitioning objectives.

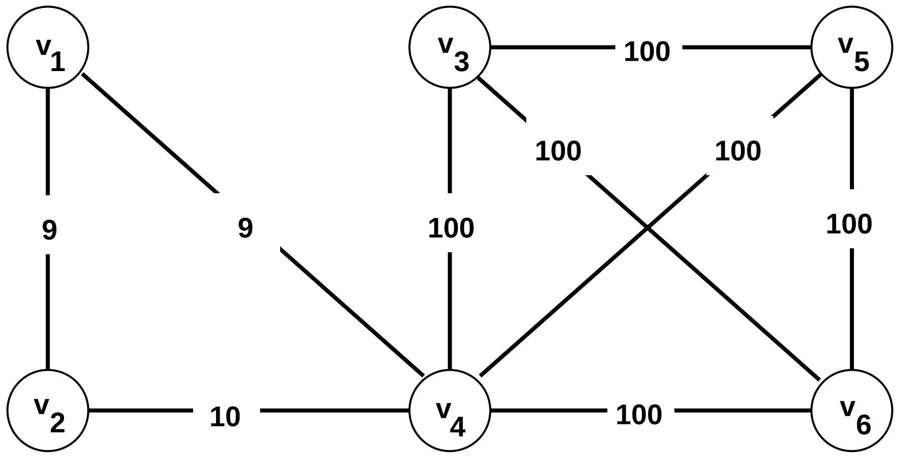


Figure 2: Optimal bipartitionings of an edge-weighted graph with six modules. The min-cut bipartitioning $\{\{v_1\}, \{v_2, v_3, v_4, v_5, v_6\}\}$ will have cutsize 18, but is very unbalanced. The min-cut bisection $\{\{v_1, v_2, v_4\}, \{v_3, v_5, v_6\}\}$ has cutsize 300, much worse than the arguably more natural solution $\{\{v_1, v_2\}, \{v_3, v_4, v_5, v_6\}\}$ which has cutsize 19 and corresponds to the optimal ratio cut of $\frac{19}{8}$.

2.4 Multi-Way Partitioning Formulations

A *multi-way* partitioning is a k-way partitioning with $k > 2$. A standard formulation is:

Min-Cut k-Way Partitioning: Given lower and upper cluster size bounds L and U, minimize $F(P^k) = \sum_{h=1}^{k} E(C_h)$ such that $L \leq w(C_i) \leq U$ for all $h = 1, \ldots, k$.

Exact cluster size balance is achieved by setting $L = \frac{1}{k}\sum_i w(v_i) - \epsilon$ and $U = \frac{1}{k}\sum_i w(v_i) + \epsilon$, where $\epsilon > 0$ may again be required for move-based algorithms to maintain feasible solutions. A similar constrained formulation used by, e.g., [18] requires that $w(C_h) = m_h$ for user-prescribed cluster sizes $m_1 \geq m_2 \geq \ldots \geq m_k$. Notice that the Min-Cut k-Way Partitioning objective sums the nets cut by each cluster: a net that is cut by h clusters is counted h times in the sum. As noted by [159] [179], this objective is preferable to simply counting the number of nets cut (i.e., $F(P^k) = |E(P^k)|$) since signal nets that span more clusters can consume more I/O and timing resources. Note that the two objectives are identical for undirected graphs (which have $|e| = 2$ for all $e \in E$).

As with bipartitioning, it is possible to integrate cutsize and cluster size balance within a single objective. To this end, Chan et al. [37] and Yeh et al. [193] respectively proposed the *Minimum Scaled Cost* and *Minimum Cluster Ratio* objectives.

Minimum Scaled Cost: Minimize

$$F(P^k) = \frac{1}{n(k-1)} \sum_{h=1}^{k} \frac{|E(C_h)|}{w(C_h)}.$$

Minimum Cluster Ratio: Minimize

$$F(P^k) = \frac{|E(P^k)|}{\sum_{h=1}^{k-1} \sum_{l=i+1}^{k} w(C_h) \cdot w(C_l)}.$$

Both of these objectives are k-way generalizations of the ratio cut objective, and are exactly equivalent to Minimum Ratio Cut Bipartitioning when $k = 2$. Scaled Cost seems more useful since it penalizes nets that are divided among more than two clusters; Cluster Ratio is also more difficult to evaluate, and the $O(k^2)$ terms in the denominator do not give immediate intuition regarding the cutsize-balance tradeoff.

Other k-way partitioning objectives have been proposed in the context of circuit clustering, i.e., when k is large with respect to $|V|$ (see Section 6). The *DS* objective [48] is:

DS: Maximize

$$F(P^k) = \frac{1}{n} \sum_{h=1}^{k} \frac{degree(C_h)}{separation(C_h)}$$

where $degree(C_h)$ is the average number of nets incident to each module of the cluster that have at least two pins in the cluster, and $separation(C_h)$ is the average length of a shortest path between two modules in C_h ($= \infty$ if the cluster is disconnected). Since DS requires $O(n^3)$ time to evaluate, it is more useful for comparison rather than optimization of clustering solutions.

The *Absorption* objective [179] measures the sum of the fractions of nets "absorbed" by the clusters:

Absorption: Maximize

$$F(P^k) = \sum_{h=1}^{k} \sum_{e \in E | e \cap C_h \neq \emptyset} \frac{|e \cap C_h| - 1}{|e| - 1}$$

e.g., net e incident to cluster C_h adds absorption zero if e has only one pin in C_h, and adds absorption one if all of e's pins are in C_h.

The *Density* objective [101] maximizes the sum of cluster *densities*, where the density of a cluster C is the number of hyperedges completely contained in C, divided by the the weight of C:

Density: Maximize

$$F(P^k) = \sum_{h=1}^{k} \frac{|\{e \in E \mid e \subseteq C_h\}|}{w(C_h)}$$

Density differs from Absorption in that only nets completely absorbed in the cluster are counted. Without this denominator, the objective is equivalent to minimizing the total number of cut nets (cf. the "Clustering" problem in [133]), since it can be rewritten as $E - E(P^k)$.

A very important clustering-related formulation captures the problem of timing-driven k-way partitioning (see Section 5.1 below). A DAG representation is assumed, and we seek to minimize the longest delay over all paths from primary inputs to primary outputs of the circuit, subject to satisfying the size constraint $w(C) \leq U$. Each module $v \in V$ has delay $\delta(v)$, and an edge (v_i, v_j) between two modules has delay one if v_i and v_j are in different clusters, and delay zero if v_i and v_j are in the same cluster. In other words, the cost(delay) of a path p from v_i to v_j can be written as $cost(p) = \sum_{v \in p} \delta(v) + K(p)$, where $K(p)$ is the number of intercluster edges in p.

Min-Delay Clustering: Given a DAG $G(V, E)$, module delays $\delta(v_i)$ and cluster size bound U, minimize $F(P^k) = \max_{\text{all paths } p} cost(p)$ such that $w(C_h) \leq U$ for all $h = 1, \ldots, k$.

As noted in [151], this formulation can capture delay between adjacent modules v_i and v_j by inserting a dummy module v_m on edge (v_i, v_j) with $w(v_m) = 0$ and edge delay $\delta(v_m)$; such a transformation only increases the size of the netlist by a constant factor. Applications of Min-Delay Clustering abound in the performance-driven design of multi-chip module (MCM) or multiple-FPGA systems. For example, in the latter application a technology-mapped circuit will have been decomposed into configurable logic block (CLB) equivalents, and these must be partitioned onto k FPGA devices subject to hard I/O limits for each device.

Finally, there is the class of *layout-driven* multi-way partitioning formulations, which are generally motivated by the link between partitioning and placement in typical physical design methodologies. Vijayan [186] proposed an abstract formulation that requires the netlist modules to be mapped onto an underlying k-node tree structure. Each node of the tree has a prescribed capacity of netlist modules, and the cost of the k-way partitioning is the sum of the costs of routing each net on the underlying tree structure. Thus, a tree consisting of two nodes, each with capacity $\frac{n}{2}$, captures Min-Cut Bisection. Similarly, a star topology with k leaf nodes each having capacity $\frac{n}{k}$, along with a dummy central vertex having zero capacity, captures balanced Min-Cut k-Way Partitioning. Vijayan notes many applications of his formulation, including seeding of functional blocks, partitioning within nonrectangular regions, and residual logic partitioning.

For layout-driven applications, a general graph topology rather than a tree structure can better incorporate information about the layout geometry [182]. For example, if modules are assigned to 12 identical devices that are arranged in a 3×4 grid, the cost of a cut net might correspond to the routing tree cost over the devices that contain pins of the net. This formulation applies to partitioning for multiple-FPGA systems, MCM designs, and general floorplanning. As with Vijayan's formulation, edge-weighting of the underlying topology can be used to model distance, signal delay, routing congestion or other layout parameters. Roy and Sechen [159] have integrated both the perimeter of a net's bounding box and penalties for wirelength of critical-paths into an objective function for MCM partitioning. Similar formulations are given in [158] [171]. The *quadrisection* problem [178] is a classic special case of the same formulation for standard-cell placement: the underlying graph is a 2×2 grid, and the objective is to minimize the number of nets crossing the horizontal and vertical middle gridlines.

2.5 Constraint-Driven (Satisficing) Formulations

With the increasing complexity of system design, CAD optimizations are becoming constraint-driven, i.e., *satisficing*, meaning that the design problem is expressible as a decision question. If the answer to the decision question is yes, then there exists a *feasible* solution that satisfies all the constraints, and all feasible solutions are equally good. For example, a design that can achieve a given system clock speed, fit into a given gate array, meet given I/O constraints on logic blocks, etc. will indeed be a good solution. Constraint-driven partitioning formulations are most prominent in the design of multiple-FPGA systems for rapid prototyping or system emulation. Because FPGA gate density is low, and because logic (CLB) and I/O resources both have hard upper bounds, FPGA partitioning is virtually a canonical constraint-driven application. Thus, we use the term "FPGA partitioning" to exemplify the more general case of partitioning with area, I/O and perhaps timing constraints. In FPGA partitioning, a cluster corresponds to an FPGA device (i.e., a chip), so that the circuitry of the modules and the connections in the cluster must be mappable onto the chip. If all the devices are of the same type, Kužnar et al. [125] proposes finding a feasible solution that minimizes the number of devices.

Definition: A cluster C is *feasible* with respect to FPGA device type D if $w(C) \leq w(D)$ and $E(C) \leq E(D)$ where $w(D)$ is the capacity and $E(D)$ is the I/O limit of the device type D.

Single-Device FPGA Partitioning: Given an FPGA device type D, find $P^k = \{C_1, C_2, \ldots C_k\}$ such that every $C_h \in P^k$ is feasible with respect to D and $F(P^k) = k$ is minimized.

This formulation has also been studied in, e.g., [44] [100]. Notice that this formulation is easily stated as a decision question – "Given a value k and a device type D, does there exist a feasible P^k with respect to D?" More generally, a library of different device types with varying size capacities and I/O limits may be available. Each device type D has an associated cost, denoted by $cost(D)$. Let $\Delta = \{D_1, D_2, \ldots, D_r\}$ denote a library of devices and let $dev : 2^V \rightarrow \Delta$ be a partial mapping of possible clusters to the lowest-cost device types for which they are feasible. The multiple-device FPGA partitioning problem [125] is:

Multiple-Device FPGA Partitioning: Given a library of devices Δ, find $P^k = \{C_1, C_2, \ldots C_k\}$ and a partial mapping dev such that every $C_h \in P^k$ is feasible with respect to $dev(C_h)$ and $F(P^k) =$

$\sum_{h=1}^{k} cost(dev(C_h))$ is minimized.

For satisficing problems, it is often the case that many solutions can achieve the same cost. In practice, other criteria can be used to distinguish superior solutions. For example, if $w(C)$ is very close to $w(dev(C))$, then the FPGA device may not be routable; one may seek to balance "slacks" $w(dev(C)) - w(C))$ among the clusters in order to yield the highest possibility of routing. As another example, it might be desirable to minimize the number of pins used, $\sum_{i=1}^{k} |E(C_i)|$, since this quantity represents total interconnect between FPGAs. This pin-minimization objective was proposed by Woo and Kim [189], who also studied a more rigid capacity constraint: each module has an associated "cell type" and each FPGA device can only contain a fixed number of modules of each cell type.

2.6 Replication Formulations

The final class of formulations that we preview removes the assumption that clusters are disjoint, i.e., a module can belong to multiple clusters [122]. Replicating modules can reduce the cutsize, and is particularly useful for FPGA partitioning since many device architectures seem more I/O-limited or interconnect-limited than logic-limited. Replication can also reduce the number of interchip wires along a given path, increasing system performance. There are three forms of replication in the literature, respectively involving directed graphs [103], hypergraphs with source and destination information [122], and functional information [126] (i.e., the actual logic functions must be known in addition to the circuit topology).

Consider the directed graph shown in Figure 3(a), in which module v represents an N-input decoder circuit. The cut shown has size 2^N, but if the decoder v is replicated as in (b), every one of these 2^N edges will become uncut (however, N new edges will be cut). The following rules are used to modify the edge set E when $v \in C_h$ is replicated into $v' \in C_l$.

Directed Graph Replication Rules: [103]

- For each (v, w) with $w \in C_l$, replace (v, w) with (v', w).
- For each $(w, v) \in E$, add (w, v') to E.

Notice that the 2^N edges were removed in Figure 3(b) according to the first rule, and the addition of N edges followed from the second rule.

These rules can be extended to hypergraphs, where source and destination information are known [122]. The following rules are used to modify the edge set E when $v \in C_h$ is replicated into $v' \in C_l$.

Hypergraph Replication Rules:

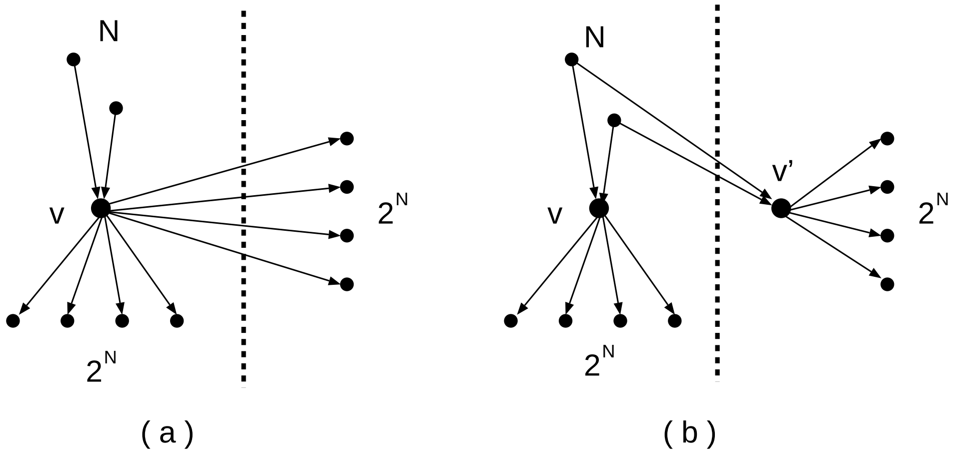


Figure 3: Directed graph corresponding to an N-to-2^N decoder v. Replication of v reduces the cut size from 2^N in (a) to N in (b).

- For every net e with $v = S(e)$ and for every $w \in D(e) \cap C_l$, remove w from e; create a new net e' with $S(e') = v'$ and $D(e') = \{w \in D(e) \cap C_l\}$. Delete any resulting 1-pin nets from the hypergraph.
- For every net e with $v \in D(e)$, add v' to $D(e)$.

Note the difference between this second rule and the corresponding rule for directed graph replication. Figure 4 shows an example with inputs (outputs) on the left (right) side of each module. By the first rule, replicating v adds the new net e_2' containing v' and w to the set of hyperedges; by the second rule, v' is added as another destination module for net e_1. Thus, e_1 is still cut only once, but if a directed tree representation and the directed graph rules were applied (as in [103]), both the edges $(S(e_1), v')$ and $(S(e_1), w)$ would cross the cut.

The third replication formulation is due to Kužnar et al. [126] and assumes functional knowledge of the circuit. The key idea is that if one knows the actual function of a module, then not all of the incoming signals may be required by both copies of the replicated module. The first hypergraph replication rule above is applied to construct new nets and remove any resulting 1-pin nets. Then, the second hypergraph replication rule is modified (again, when replicating v to v') to:

Logic-Dependent Replication Rule: Let $E^* = \{e \mid S(e) = v'\}$. For every net e such that $v \in D(e)$, if $\exists\, e^* \in E^*$ that depends on e, add v' to $D(e)$.

Consider the module v shown in Figure 5(a), with three inputs e_1, e_2, e_3 and two outputs e_4, e_5. From the dotted lines within the modules, we see that output e_4 depends only on inputs e_1 and e_3, while output e_5 depends only on inputs e_2 and e_3. Three nets cross the cutline in (a), but only two

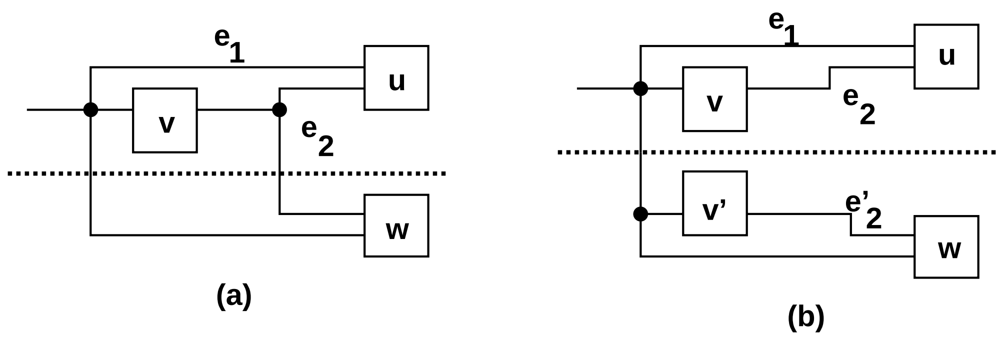


Figure 4: (a) A circuit with two nets crossing the cut, and (b) the same circuit with module v replicated, yielding only one cut net.

nets are cut after v has been replicated to v' in (b). Applying the first replication rule deletes e'_4 (since $D(e'_4) = \emptyset$) while e'_5 remains. Applying the second rule using $E^* = \{e'_5\}$ shows that v' needs connections to e_2 and e_3 only, since e'_5 does not depend on e_1. This formulation is germane to FPGA system synthesis, e.g., CLBs of such popular device families as the Xilinx 4000-series indeed have multiple outputs with differing dependencies on the input variables.

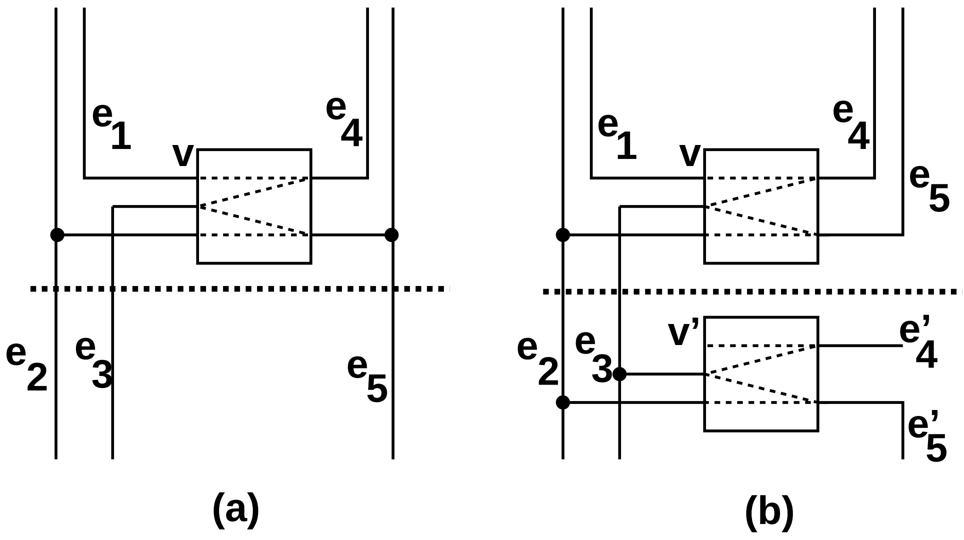


Figure 5: (a) A module v with three nets crossing the cut; (b) replication requires only two nets to cross the cut since the net corresponding to input e_1 is not needed by v'.

3 Move-Based Approaches

A partitioning approach is *move-based* if it iteratively constructs a new candidate solution based on two considerations: (i) a *neighborhood structure* that is defined over the set of feasible solutions, and (ii) the previous history of the optimization. The first consideration requires the notion of a *local* perturbation of the current solution; this is the heart of the move-based paradigm. The type of perturbation used determines a topology over the solution space, known as the *neighborhood structure*. For the objective function to be "smooth" over the neighborhood structure, the perturbation (also known as a *neighborhood operator*) should be small and "local". Typical neighborhood operators for partitioning include swapping a pair of modules, or shifting a single module, across a cluster boundary. For example, two partitioning solutions are neighbors under the pair-swap neighborhood structure if one solution can be derived from the other by swapping two modules between clusters. In general, the solution space is explored by repeatedly moving from the current solution to a neighboring solution. With respect to previous history, some approaches are "memoryless", e.g., a simple greedy method might rely only on the current solution to generate the next solution. On the other hand, such methods as Kernighan-Lin or Fiduccia-Mattheyses implicitly remember the entire history of a "pass"; hybrid genetic-local search or tabu search approaches must also remember lists of previously seen solutions.

Move-based approaches dominate both the literature and industry practice for several reasons. First, they are generally very intuitive – the logical way of improving a given solution is to repeatedly make it better via small changes, such as moving individual modules. Second, iterative algorithms are simple to describe and implement; for this reason, the bipartitioning method of Fiduccia-Mattheyses [65] and the multi-way partitioning method of Sanchis [164] are standards against which nearly all other heuristics are measured. Third, the move-based approach encompasses more sophisticated strategies for exploring the solution space – e.g., simulated annealing, tabu search, and genetic algorithms – which yield performance improvements over greedy iterative methods while retaining the intuitiveness associated with local search. Finally, the move-based approach is independent of the nature of the objective function that is used to measure solution quality. While other approaches might require the objective to be of a particular form, or a relatively simple function of solution parameters, the move-based approach can flexibly incorporate arbitrary constraints (e.g., on critical path delays or I/O utilization). Thus, the move-based approach has been applied to virtually every known partitioning formulation.

The remainder of this section is organized as follows. We first discuss the "classic" greedy iterative improvement algorithms, which move from the current solution to the *best* neighboring solution. We also discuss implementation issues that can greatly affect the performance of these methods. We then discuss three *stochastic hill-climbing* methods – simulated annealing, tabu search, and genetic algorithms – which can move to higher-cost neighboring solutions in order to escape local minima. Finally, the section concludes by reviewing the numerous works that have each adapted some basic move-based paradigm in addressing some variant partitioning formulation.

3.1 Iterative Improvement

Iterative improvement algorithms are based on the greedy strategy: start with some feasible solution and iteratively move to the best (improving) neighboring solution. The process terminates when the algorithm reaches a local minimum, i.e., a solution for which all neighbors have greater cost. Early greedy improvement methods apply simple pair-swap or single-move neighborhood operators, and quickly reach local minima corresponding to poor solutions. By contrast, the strategies discussed in this subsection all rely on extended neighborhood structures which effectively allow hill-climbing out of local minima (even though each strategy is greedy with respect to its neighborhood operator). Due to these methods, iterative improvement remains a viable strategy, particularly when clustering techniques are integrated (see Section 6).

3.1.1 The Kernighan-Lin (KL) Algorithm

In 1970, Kernighan and Lin [117] introduced what is often described as the first "good" graph bisection heuristic. Their algorithm uses a pair-swap neighborhood structure and proceeds in a series of *passes*. During each pass of the algorithm, every module moves exactly once, either from C_1 to C_2 or from C_2 to C_1. At the beginning of a pass, each module is *unlocked*, meaning that it is free to be swapped; after a module is swapped it becomes *locked*. KL iteratively swaps the pair of unlocked modules with the highest *gain*, where the gain of swapping $v_i \in C_1$ with $v_j \in C_2$ is given by $F(\{C_1, C_2\}) - F(\{C_1 + v_j - v_i\}, \{C_2 + v_i - v_j\})$. In other words, the gain is the decrease in solution cost that results from the pair-swap. For a weighted undirected graph and the min-cut objective, the gain for these modules is given by

$$gain(v_i, v_j) = \sum_{v_k \in C_2} (a_{ik} - a_{jk}) + \sum_{v_k \in C_1} (a_{jk} - a_{ik}).$$

The swapping process is iterated until all modules become locked, and the lowest-cost bisection observed during the pass is returned. Another pass is then executed using this bisection as its starting solution; the algorithm terminates when a pass fails to find a solution with lower cost than its starting solution. An advantage of KL is that it can in some sense climb out of local minima, since it always swaps the pair of modules with highest gain even if this gain is negative. However, if we consider all solutions reachable within a single pass of the algorithm to be "neighbors" of the current solution, the KL algorithm is still seen to be greedy.

A simple implementation of KL requires $O(n^3)$ time per pass since finding the highest-gain swap involves evaluating $O(n^2)$ swaps. In practice, this complexity is reduced to $O(n^2 \log n)$ by maintaining a sorted list of gains. The number of passes is clearly bounded by $|E|$ for unweighted graphs (since the cost must improve with each pass); in practice, significantly fewer passes will be needed to reach a local minimum. Recently, Dutt [61] presented the *Quick Cut* algorithm which reduces the complexity of a single KL pass to $O(\max\{|E| \cdot \log n, |E| \cdot deg_{max}\})$ where deg_{max} is the maximum module degree. The speedup is based on the observation that it is not necessary to search more than a certain subset of $(deg_{max})^2$ module pairs to find the pair with highest gain. Although the original KL algorithm and Quick Cut apply only to undirected weighted graphs, Schweikert and Kernighan [169] extended KL to hypergraphs and a similar extension appears possible for Quick Cut.

3.1.2 The Fiduccia-Mattheyses (FM) Algorithm

Fiduccia and Mattheyses [65] presented a KL-inspired algorithm which reduced the time per pass to linear in the size of the netlist. The main difference between KL and FM is the neighborhood structure: a new bipartitioning is derived by moving a single module either from C_1 to C_2 or from C_2 to C_1. Since intermediate solutions considered by FM must violate the strict bisection constraint, the solution is permitted to deviate from an exact bisection by the size of the largest module. The gain associated with module $v \in C_i$ is $|E(C_i)| - |E(C_i - v)|$. Like KL, the FM algorithm performs passes wherein each module moves exactly once, returns the best solution observed during a pass, and terminates when a pass fails to improve the cost function. However, FM permits a much faster $O(|E|)$ implementation on undirected graphs and an $O(p)$ implementation on hypergraphs, where p is the number of pins in the netlist. The key to the speedup is the *gain bucket* data structure shown in Figure 6, which allows constant-time selection of the module with highest gain and fast gain updates after each move.

The efficient management of gain buckets is possible because (i) all module gains are integers, and (ii) every gain is bounded above by deg_{max} and below by $-deg_{max}$. Even when variations of the pure min-cut objective are permitted (e.g., non-integral net weights), there is usually a tight bound on the range of possible gain values. At the beginning of a pass, the gains for each of the n possible module moves are computed in $O(p)$ time, and each move is inserted into the data structure according to its gain. The modules with highest gain are stored in the bucket with gain value $MAXGAIN$. During an FM pass, a module is selected from this bucket and deleted from the linked list; the module is moved from its current cluster and the gains of unlocked modules incident to the moved module are updated. The updating of a module can be accomplished by removing it from its gain bucket list and inserting it at the head of the bucket list indexed by its new gain value. If one of these modules has new gain that is larger than $MAXGAIN$, then $MAXGAIN$ is updated to this new value. If the bucket indexed by $MAXGAIN$ becomes empty, then $MAXGAIN$ is decreased until it indexes a non-empty bucket.

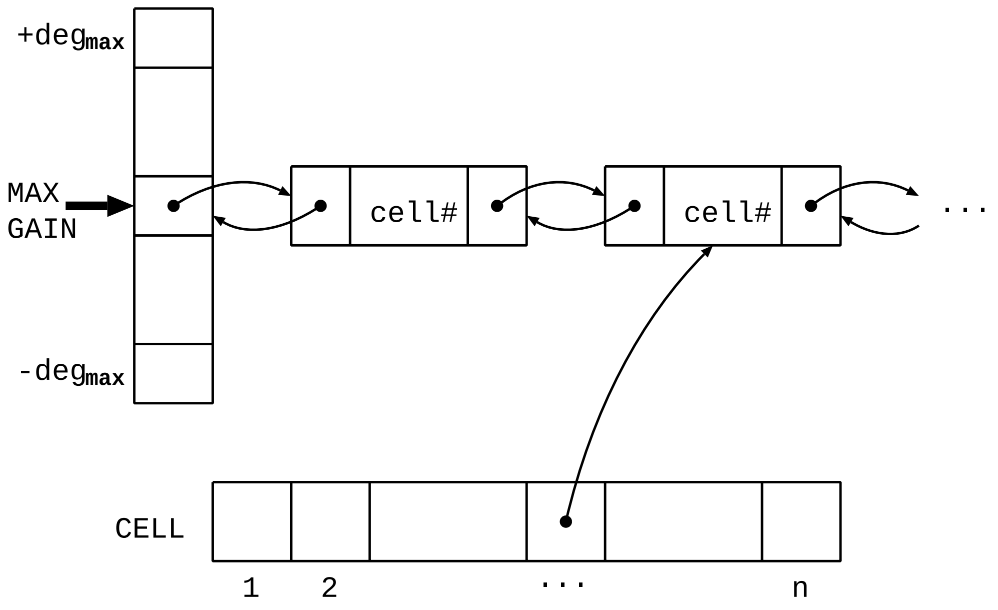


Figure 6: The gain bucket list structure as shown in [65].

3.1.3 Tie-Breaking Strategies

A frequently cited enhancement to FM is that of Krishnamurthy [123]. He suggested that the lack of an "intelligent" tie-breaking mechanism from among the many possible modules in the highest-gain bucket could cause FM to make "bad" choices. Hagen et al. [86] observe that 15 to 30 modules will typically share the highest-gain value at any time during an FM pass on the Primary1 MCNC benchmark (833 modules). As a tie-breaking mechanism, Krishnamurthy introduced a *gain vector*, which is a sequence of potential gain values corresponding to numbers of possible moves into the future. Thus, the r^{th} entry in the gain vector looks r moves ahead, and ties are broken lexicographically by 1^{st}-level gains, then 2^{nd}-level gains, etc. Gain vectors can be integrated into the FM gain bucket structure, increasing the complexity to $O(pr)$, where r is the maximum number of lookahead moves stored in the gain vector.

Krishnamurthy defines the *binding number* $\beta_{C_i}(e)$ of a signal net e with respect to cluster C_i as the number of unlocked modules in $C_i \cap e$, unless $C_i \cap e$ contains a locked module, in which case $\beta_{C_i}(e) = \infty$. Intuitively, the binding number measures how many modules must be moved from C_i in order to move all the pins in e out of C_i. This is impossible when a module in $C_i \cap e$ is locked, so the binding number is infinite. The r^{th}-level gain $\gamma_r(v)$ for $v \in C_1$ is given by

$$\begin{aligned}\gamma_r(v) = \quad & \mid \{e \in E(\{v\}) \text{ s.t. } \beta_{C_1}(e) = r, \beta_{C_2}(e) > 0\} \mid \\ & - \mid \{e \in E(\{v\}) \text{ s.t. } \beta_{C_1}(e) > 0, \beta_{C_2}(e) = r - 1\} \mid\end{aligned} \tag{3.1}$$

The first term counts how many nets with binding number $r - 1$ are "created" by the move, and the second term counts how many nets with binding number $r - 1$ are "destroyed" by the move (i.e., have new binding number $= \infty$). Hence, the r^{th}-level gain counts the additional number of nets that may possibly become uncut following r moves. Note that the 1^{st}-level gain is identical to the gain in the FM algorithm. Although the benefit of higher-level gains is well-documented (e.g., [94]), small modifications might improve performance. For example, instead of combining the positive and negative gains into a single term, once could store both terms separately to afford a two-tiered r^{th}-level comparison.

Even with Krishnamurthy's gain vector, ties may still occur in the 1^{st}- through r^{th}-level gains. In this case, the implementation of the gain bucket data structure will determine which module is selected. The original FM algorithm uses a linked list for each bucket; from Figure 6, reproduced from [65], we may infer that modules are removed and inserted at the head of the list, i.e., the bucket organization corresponds to a Last-In-First-Out (LIFO) stack. The authors of [65] do not mention why a LIFO organization was chosen, but one can speculate that it was an "obvious" choice. However, a First-In-First-Out (FIFO) organization which supports the same update efficiency could also be implemented, simply by removing modules from the head of the linked list and inserting modules at the tail of the linked list. Alternatively, a random organization could be implemented, though the time complexity would increase slightly since bucket selection could not be accomplished in constant time.

Recently, the authors of [86] observed that Sanchis [164], and most likely Krishnamurthy [123] also, used random selection from gain buckets in their implementations. Furthermore, [86] observed that a LIFO gain bucket organization yields considerably superior solutions than either the FIFO or random bucket organization, for both the FM and Krishnamurthy algorithms. (This result seems quite surprising: if anything, "folklore" would have it that introducing randomness can improve solution quality by increasing the range of the neighborhood search.) One possible explanation for the success of LIFO is that it enforces "locality" in the choice of modules to move, i.e., modules that are naturally clustered together will tend to move sequentially. Hagen et al. [86] use this idea of locality to propose an alternative formula for higher-level gains which also improves performance. Given that slight modifications to the bucket organization or gain formula can so dramatically affect the solution quality obtained by these iterative approaches, exploring other implementation choices seems to be a very promising direction for future work. The importance of such research is heightened by the present widespread use of the FM and Krishnamurthy algorithms.

3.1.4 Sanchis' Multi-Way Partitioning Algorithm

Sanchis [164] extended the FM algorithm, together with Krishnamurthy's lookahead scheme, to multi-way partitioning. The algorithm is generally straightforward, although there are several knotty implementation issues. Sanchis' detailed explanation of these issues accounts for her algorithm's wide use in practice, as well as its present status as a benchmark against which multi-way partitioning heuristics are compared.

Sanchis extends the definition for the binding number of a net e with respect to cluster C_i as $\beta'_{C_i}(e) = \sum_{j \neq i} \beta_{C_j}(e)$. In other words, $\beta'_{C_i}(e)$ is the sum of the Krishnamurthy binding numbers for e with respect to every cluster except C_i, and measures the difficulty of removing the pins in e from C_i. The analog of equation (3.1) for the r^{th}-level gain of moving module $v \in C_i$ to C_j is

$$\begin{aligned} \gamma_r(v) = \quad & \mid \{e \in E(\{v\}) \ s.t. \ \beta'_{C_j}(e) = r, \beta'_{C_i}(e) > 0\} \mid \\ - \quad & \mid \{e \in E(\{v\}) \ s.t. \ \beta'_{C_i}(e) > 0, \beta'_{C_j}(e) = r - 1\} \mid \end{aligned} \tag{3.2}$$

Equation (3.2) assumes that the objective is $F(P^k) = |E(P^k)|$. In [165], Sanchis shows how to modify this formula to handle the objective function which assigns cost $j - 1$ to a net that spans j clusters (this is very similar to $F(P^k) = \sum_{h=1}^{k} |E(C_h)|$). For this objective, the r^{th}-level gain is

$$\begin{aligned} \gamma_r(v) = \quad & \mid \{e \in E(\{v\}) \ s.t. \ \beta_{C_i}(e) = r, \beta_{C_j}(e) > 0\} \mid \\ - \quad & \mid \{e \in E(\{v\}) \ s.t. \ \beta_{C_j}(e) > 0, \beta_{C_i}(e) = r - 1\} \mid \end{aligned}$$

3.2 Simulated Annealing

Simulated Annealing (SA) was popularized by Kirkpatrick et al. [119] as an alternative to greedy approaches, which are quickly trapped in local minima since they can only make *downhill* moves. Given a neighborhood structure and a current solution, SA picks a random neighbor of the current solution and moves to this new solution if it represents a downhill move. Even if the new solution represents an uphill move, SA will move to it with probability $e^{\frac{-\Delta}{T}}$ (termed a "Boltzmann acceptance rule") and otherwise retain the current solution; here, Δ is the cost of the new solution minus the cost of the current solution, and T is the current value of a *temperature* parameter which guides the optimization. To control the rate of convergence and the strategy for exploring the solution space, the user typically establishes a *temperature schedule* by which T varies, e.g., as a function of the number of moves made. The SA algorithm enjoys certain theoretical attractions [72] [88] (see also [127]): using Markov chain arguments and properties of Gibbs-Boltzmann statistics, one can show that SA will converge to a globally optimum solution given an infinite number of moves and a temperature schedule that *cools* to zero sufficiently slowly. The use of terms such as "cooling" and "temperature schedule" are due to SA's analogy to physical annealing of a material into a ground-state energy configuration.

For the Min-Cut Bisection problem, Johnson et al. [110] conducted an extensive empirical study of simulated annealing versus iterative improvement approaches, using various random graphs as a testbed (see Section 7.1). The authors of [110] conclude that SA is a competitive approach, outperforming KL for uniform and geometric random graphs. However, they suggest that multiple runs of KL with random starting solutions may be preferable to SA for sparse graphs that have local structure (a description that applies to circuit netlists). They also make a number of interesting observations, including:

- Starting SA with a good solution as opposed to a random solution may be advantageous, particularly if the good solution can identify and exploit special structure of the instance.
- Spending long periods of time at high temperatures is not necessary.
- Geometric cooling schedules (i.e., setting $T = \alpha^M T_0$ where M is the number of moves made and T_0 is the initial temperature) seem at least as effective as nonadaptive alternatives such as logarithmic or linear cooling. Adaptive cooling schedules which are modified during the execution of SA (e.g., based on solution quality, distribution of solution costs in the neighborhood, etc.) hold promise.
- Expanding the feasible solution space may be worthwhile (e.g., by loosening the size constraints for bisection); expanding the neighborhood structure allows good solutions to be found more quickly.

These conclusions are based on studies of random graphs which do not possess natural hierarchical structure. Although there is no *a priori* reason to doubt such conclusions will also hold for circuit netlists, similar studies using VLSI circuits would seem worthwhile. We note that annealing is generally not yet viewed as "practical" for VLSI partitioning applications: runtimes are simply too long. For this reason, there is only limited work that addresses the use of SA for VLSI partitioning. Parallel implementations, improved temperature schedules, and two-stage approaches which anneal at low temperature starting from a good heuristic solution have all been investigated for VLSI placement, where annealing has often outperformed other heuristics. Quite possibly, such enhancements will make SA more viable for partitioning applications in the future.

One reason for long SA runtimes is that at low temperatures, many candidate moves (i.e., neighbors of the current solution) might be generated and rejected before one is finally accepted. Greene and Supowit [78] proposed a "rejectionless" variant of SA in which no moves are rejected, since candidate moves are generated with probability proportional to their likelihood of being generated *and accepted* given the current solution. For bipartitioning, a "gain" must be stored for each module, and only the gains for modules in $M(v)$ need to be updated when v is moved. The Boltzmann acceptance rule maps the gain of v to a weight $w(v)$, and the probability that v is generated (and moved to) is $w(v)/\sum_{u\in V} w(u)$.

Recently, Roy and Sechen [159] used simulated annealing to implement a timing-driven MCM partitioning algorithm. Their cost function is the sum of the half-perimeters of each net spanning multiple chips on the MCM, plus a timing penalty for critical nets whose wirelength exceeds predefined constraints. Chatterjee and Hartley [41] presented an SA-based heuristic which performs partitioning and placement simultaneously. Their cost function is the sum of five components: conflict and capacity costs which combine to measure the feasibility of mapping a cluster onto the chip, a wasted space cost that penalizes unused resources on the chip, a half-perimeter net cost, and a pin cost. These works illustrate the ease with which SA can address relatively arbitrary objective functions (of course, simple objective functions are preferred since evaluation of solution cost dominates the runtime). Sun and Sechen [179] have also used SA to optimize Absorption in a clustering preprocessor within the TimberWolf placement tool.

Mean Field Annealing

Mean Field Annealing (MFA) is a technique similar to SA which also has a physical analogy to systems of particles in thermal equilibrium. Van den Bout and Miller [54] showed how MFA could be applied to graph partitioning. They use an indicator n-vector $\vec{x}$ to denote a bipartitioning solution, where $x_i = 0$ corresponds to membership of v_i in C_1 and $x_i = 1$ corresponds to membership of v_i in C_2. However, x_i can also take on any *real* value between 0 and 1. Initially, each x_i is set to be slightly larger than 0.5. Iteratively, a random v_i is chosen and the two solutions $\vec{x}(0)$ with $x_i = 0$ and $\vec{x}(1)$ with $x_i = 1$ are evaluated (the cost function is extended to non-discrete solutions). MFA then finds a "compromise" value for x_i,

$$x_i = (1 + e^{(F(\vec{x}(1)) - F(\vec{x}(0)))/T})^{-1}$$

and $\vec{x}$ is modified accordingly for the next iteration. The intuition behind this assignment is that x_i should migrate towards its natural value, e.g., if $F(\vec{x}(1)) = F(\vec{x}(0)$ then $x_i = 0.5$ and if $F(\vec{x}(1)) \ll F(\vec{x}(0)$ then $x_i \approx 1$.

This process of choosing a random v_i and computing a new x_i value is repeated until a stable solution is reached., The temperature T is then lowered and the algorithm repeated; this polarizes the x_i values further (to 0 or 1). Finally, a graph bipartitioning solution is obtained by rounding each x_i to its nearest discrete value. Bultan and Aykanat [33] have extended this basic approach to multi-way partitioning of hypergraphs. An earlier work of Peterson and Anderson [145] studied the performance of MFA for graph bisection; their formulation used $\vec{x}$ with $x_i = -1$ if $v_i \in C_1$ and $x_i = 1$ if $v_i \in C_2$, with maximization objective

$$F(\vec{x}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j - \frac{\alpha}{2}(\sum_{i=1}^{n} x_i)^2$$

where α is a user-chosen constant. The first term of the objective affords a positive weight a_{ij} if (v_i, v_j) is not cut, and weight $-a_{ij}$ if (v_i, v_j) is cut; the second term penalizes cluster size imbalance.

While MFA is a well-studied optimization paradigm, its application to VLSI CAD remains largely unexplored. Like many promising approaches (for instance, those based on eigenvectors, one-dimensional orderings, mathematical programming, fuzzy clustering, etc.), the MFA approach permits non-discrete partitioning solutions (i.e., a relaxation), and thus allows search over a larger solution space. As highlighted in Section 4, it is a reasonable strategy is to leave the feasible ("legal") region of the solution space, find a superior infeasible ("non-legal") solution, then find the feasible solution that is closest to the infeasible solution. However, like SA, MFA takes a long time to converge, though parallel implementations may reduce this time.

3.3 Tabu Search

Tabu search was proposed by Glover [74] as a general combinatorial optimization technique. Tabu search is similar to iterative improvement in that moves are sought which transform the current solution to its best neighboring solution. Tabu search also maintains a *tabu list* of its r most recent moves (e.g., pairs of modules that have been swapped recently), with r a prescribed constant; moves on the tabu list cannot be made. The tabu list exists to prevent cycling near a local minimum and also to enable uphill moves; in some sense, the tabu list can be viewed as an alternative to the locking mechanism in KL and FM. Occasionally, a tabu move may be made if the *aspiration function* permits it (essentially, the aspiration function will override the tabu list if the move "looks good enough"). A claimed advantage of tabu search over SA is that SA may waste time making poor random moves or cycling through previously visited regions of the solution space. Although tabu search can certainly behave similarly, it is designed to quickly find a local minimum, climb out of the "valley" surrounding this local minimum, and then move on to the next local minimum. In this sense, tabu search might explore the solution space more efficiently than SA.[7]

Tabu search has been applied to graph bisection by Tao et al. [180] and Lim and Chee [134]. In [180], a move consists of swapping a pair of modules, and the aspiration function is one less than the cost

[7]It is instructive to compare the motivations for tabu search with those of other methods in the optimization literature, e.g., Baum's *iterated descent* [21] or the "iterated Lin-Kernighan" strategy used by Johnson [109] to address the traveling salesman problem. Furthermore, one may ask whether certain problem classes possess relationships among local minima in the neighborhood structure which make tabu search more effective (e.g., see [25]).

of the best solution, i.e., a pair swap on the tabu list is accepted if the resulting solution has lower cost than every solution seen so far. The authors of [134] also adopt a pair-swapping neighborhood structure, though the aspiration function depends on the cost of the current solution. For each solution with cost F, the aspiration value $A(F)$ is the maximum decrease in cost that has been previously observed when moving from a solution of cost F. Thus, if a move from a solution with cost F results in a solution with cost F', then the aspiration function overrides the tabu list if $F - F' > A(F)$.

Areibi and Vannelli [9] first applied tabu search to hypergraph bipartitioning using the same aspiration function as [134], but with single module moves as the neighborhood operator. Andreatta and Ribeiro [7] applied tabu search to a DAG partitioning formulation designed for testing of combinational circuits. Areibi and Vannelli have also applied tabu search as a post-processing mechanism to initial partitioning solutions constructed using an eigenvector [10] and a genetic algorithm [8] [11]. In [8], the genetic algorithm is used to generate many good starting solutions, so that the tabu search can in some sense concentrate its efforts on the most promising regions of the solution space.

3.4 Genetic Algorithms

Genetic algorithms are motivated by Darwin's theory of natural selection in evolution, where "superior" members of a species produce more offspring in succeeding generations than "inferior" members [98]. A *genetic algorithm* (GA) starts with an initial *population* of solutions. This population evolves over generations, with the solutions in the current generation being replaced with a set of *offspring* solutions in the next generation. A GA implementation typically has *crossover* and *mutation* operators that determine candidate offspring for the next generation. The crossover operator is analogous to mating: two solutions are selected from the current population (based on some probabilistic *selection scheme*), and (their descriptors) are partially mixed to generate an offspring. The mutation operator enables small random perturbations to a given single solution. There exists some heuristic justification (cf. the "schema theorem" of [98]) for why such operators enable "good" solution characteristics to become more prevalent (and "bad" characteristics less prevalent) in the population with succeeding generations. The *replacement scheme* is the final component of a GA; it determines which offspring will replace which members of the current population. Designing each of these elements (crossover and mutation operators, selection scheme and replacement scheme) seems to be critical to the success of a genetic approach.

Inayoshi and Manderick [108] studied weighted graph bisection, representing solutions as indicator vectors having equal numbers of 0's and 1's. The Hamming distance between two solutions $\vec{x}$ and $\vec{y}$ is given by $d_H(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}||^2$, i.e., the number of entries in which $\vec{x}$ and $\vec{y}$ differ. The crossover operator applied to $\vec{x}$ and $\vec{y}$ constructs an offspring $\vec{z}$ in which $z_i = x_i$ if $x_i = y_i$; otherwise, if $d_H(\vec{x}, \vec{y}) = 2M$, the $2M$ coordinates of $\vec{z}$ in which $\vec{x}$ and $\vec{y}$ differ are randomly filled with M ones and M zeros. Thus, $\vec{z}$ inherits the entries common to its two parents, and has mean Hamming distance M to each of its parents. A mutation of $\vec{x}$ flips a random $x_i = 1$ to 0 and a random $x_j = 0$ to 1, yielding one of the $\frac{n^2}{4}$ neighbors of $\vec{x}$ according to the pair-swap neighborhood structure. The selection scheme is linear-rank based: solutions are ranked by their cost, and the probability of a solution being chosen for a mutation or crossover operation is linearly decreasing in its rank.

Ackley [1] proposed a GA for Min-Cut Bisection that combines crossover and mutation into a single operation. The crossover of $\vec{x}$ and $\vec{y}$ yields an offspring $\vec{z}$ for which each entry z_i has probability p of being set to a random value, and probability $\frac{1-p}{2}$ of being set to x_i or y_i, respectively. The selection scheme is purely random and the replacement scheme eliminates a random solution in the current population which has below-average quality. This GA may create unbalanced solutions that are not legal bisections, and therefore also incorporates a penalty term equal to $\frac{2}{5}(||\vec{x}||^2 - \frac{n}{2})^2$ in the cost function. The structure of the cost function reflects the preference of [1] for a continuous cost function over the solution space.

Bui and Moon have utilized GAs for graph bisection [32] and for ratio cut bipartitioning of hypergraphs [31]. Their linear selection scheme is a function of solution cost instead of rank (specifically, the best solution is four times as likely to be selected as the worst solution, and a solution of intermediate quality will be chosen with probability proportional to its cost difference from the worst solution). The replacement scheme is adaptive; in general, an offspring replaces its more closely related parent (in terms of Hamming distance) if it has lower cost than the parent. The mutation operator flips each x_i value in $\vec{x}$ with independent probability $= 0.015$, and crossover is accomplished via random *crossover points*. For example, if there are three crossover points $1 \leq c_1 < c_2 < c_3 \leq n$, then the offspring $\vec{z}$ derived from crossover of $\vec{x}$ and $\vec{y}$ has $z_i = x_i$ for $1 \leq i < c_1, c_2 \leq i < c_3$ and $z_i = y_i$ for $c_1 \leq i < c_2, c_3 \leq i \leq n$. Since this crossover operator makes it likelier for v_i and v_j to be in the same cluster if $|i - j|$ is small, ordering the problem encoding to capture the netlist structure can improve performance. The implementation of [31] adopts a weighted depth-first ordering; orderings based on the 1-dimensional representations discussed in Section 4 may be promising in this context.

Saab and Rao [162] proposed a *simulated evolution* bisection heuristic which has no crossover operator, but rather a more complex mutation operator. The authors of [162] define the "goodness" of a module v_i (e.g., in C_1) as $\frac{E_i}{I_i}$, where $E_i = \sum_{v_j \in C_2} a_{ij}$ is the cost of the external edges incident to v_i and $I_i = \sum_{v_j \in C_1} a_{ij}$ is the cost of the internal edges incident to v_i. If the goodness of v_i is less than a given random number between 0 and 1, then v_i is judged to be good; otherwise it is judged to be bad. Intuitively, a good vertex should remain in its current cluster since it is likely to have many internal connections; on the other hand, a bad vertex should be moved to the other cluster ([162]

also provide a second criterion for judging a vertex good or bad). The mutation operator essentially swaps large subsets of bad vertices. Finally, GAs for multi-way partitioning have been proposed by Chandrasekharam et al. [39] and Hulin [102].

Local Search Hybrids and Multi-Start

Because GAs are ill-equipped to search a prescribed region of the solution space for local optima, a given GA may take fairly long to find a good solution, if it finds one at all [31]. Thus, many researchers have successfully combined GAs with local optimization heuristics to form *GA-local search hybrids*, or *hybrid GAs*. For example, [108] applies KL to every member of the population; [31] [32] does the same with FM; and [8] do the same with tabu search.

Recent works have shed some light on the mechanism by which GAs, and GA hybrids in particular, can find good solutions. Boese et al. [25] and Inayoshi and Manderick [108] observe that many of the best local optima are "close" to other local optima according to natural measures of distance in the neighborhood structure. Figure 7 depicts 2500 runs of a greedy pair-swapping heuristic on a random graph in the class $G_{Bui}(100, 4, 10)$ (see Section 7.1). In (a), for each local minimum the average distance to the other local minima (in terms of number of pair-swaps) is plotted as a function of solution cost. The distribution indicates that the local minima with lowest cost are on average closer to all the other (2499) local minima, i.e., they are in the center of a "big valley" structure that governs the set of all local minima. Figure 7(b) plots for each local minimum the distance to the *best* (lowest-cost) local minimum; again we see that the local minima that are structurally most similar to the best minimum are also the next best solutions. Such correlations suggest that a crossover between two good solutions should lead to an even better solution, since it in some sense "averages" the good solutions and the cost surface is "convex". Indeed, GAs – in retaining common features of parents and searching only the "regions of disagreement" – implicitly assume that global optima will be located near good local minima.

Based on the "big valley" (Figure 7), Boese et al. [25] suggested the following *adaptive multi-start* (AMS) approach (originally in the context of the traveling salesman problem). First, AMS generates a set of random starting solutions and runs a greedy iterative algorithm from each solution to find corresponding local minima. Then, AMS generates new starting solutions by combining features of the t best local minima seen so far, with t being a parameter of the approach. New local minima are obtaining by running the greedy algorithm from these new starting solutions, and the process iterates until some stop criterion is met. AMS bears some resemblance to hybrid GAs, but differs in that many solutions (instead of just two) are used to generate the new starting solution. AMS also uses only the t very best solutions to construct the new starting solution, while the standard GA selection scheme

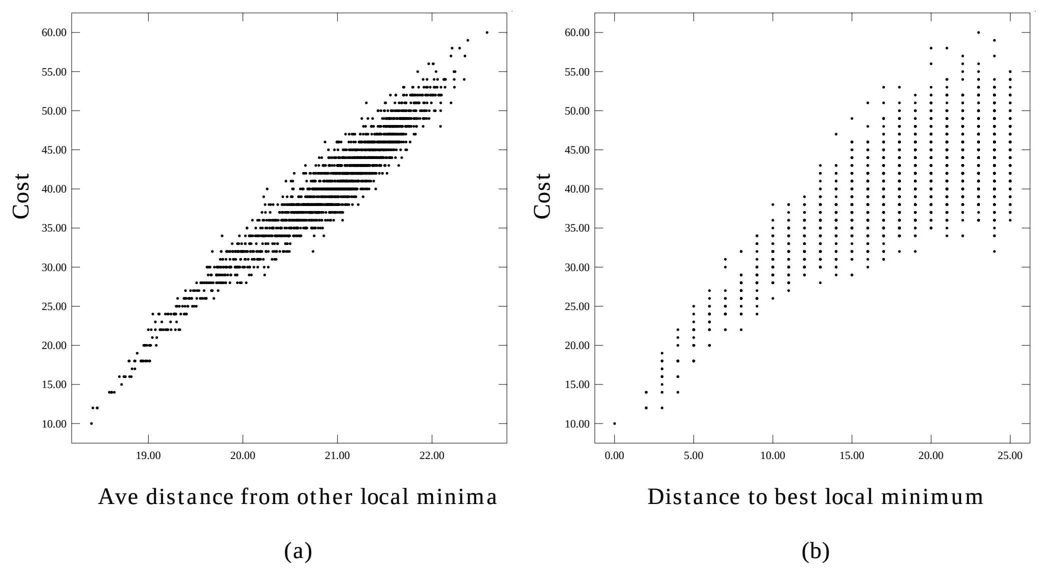


Figure 7: The distribution of 2500 random locally minimum bisections for a particular random graph in $G_{Bui}(100, 4, 10)$. The data represent 2343 distinct local minima.

will incorporate inferior solutions with nonzero probability.

3.5 Adaptations of Move-Based Algorithms

In practice, iterative improvement methods (i.e., KL, FM, Krishnamurthy and Sanchis) are the most commonly applied, due to their excellent runtimes, relatively high solution qualities, and simple implementations. Their basic paradigms are also flexible enough to adapt to objectives other than min-cut, as well as variant netlist representations, solution constraints, etc. We now review the "litany" of iterative adaptations that have appeared in the literature. We loosely categorize each approach by either its characteristics or its associated problem formulation.

3.5.1 Alternative Move Strategies

A possible weakness of the KL and FM strategies lies in the locking mechanism, e.g., a module v may be moved from C_1 to C_2 early in a pass, only to have many of its neighboring modules moved back to C_2, which causes v to be in the wrong cluster. To rectify this behavior, Hoffman [97] proposed a dynamic locking mechanism which behaves like FM, except that when v is moved out of C_i, every module in $M(v) \cap C_i$ becomes unlocked. This allows the neighbors of v in C_i to also migrate out of C_i. The algorithm permits a maximum of ten moves per module per pass. Dasdan and Aykanat

[52] propose a multi-way FM variant that allows a small constant number (e.g., three or four) module moves per pass.

Yeh et al. [194] gave an extension of Sanchis' multi-way partitioning algorithm that alternates "primal" passes of module moves with "dual" passes of net moves. A dual pass allows more than one module to be moved simultaneously, thereby expanding the neighborhood structure. For each net e, Yeh et al. define the *critical set* with respect to C_i as $e \cap C_i$ and the *complementary critical set* as $e - (e \cap C_i)$. A move in a dual pass consists of either moving the critical set out of C_i or moving the complementary critical set into C_i. The gain is the decrease in cost of the partitioning solution resulting from the move. Due to the complexity of the gain computation, a dual pass typically requires around 9-10 times the CPU of a primal pass.

Kring and Newton [122] extended FM to include module replication moves. The gain for replicating a module v is the change in cutsize resulting from applying the replication rules in Section 2.6. For hypergraphs with signal information, the gain is the number of cut nets for which v is a source, minus the number of uncut nets for which v is a destination. The authors of [122] observe that once a module v is replicated, v tends to remain replicated, so that modules in $M(v)$ tend to remain in their clusters. This behavior inhibits further moves, hence Kring and Newton restrict the number of replications by keeping three separate bucket structures for normal, replicating and unreplicating moves. Separate structures permit the unreplication of a module if its unreplication gain is higher than a prescribed threshold, even if better moves are available. [122] also prescribes a minimum replication gain threshold below which a module replication is prohibited, even if it is the best move available.

Sechen and Chen [170] propose another modification of the original KL and FM descriptions, that the Min-Cut Bisection objective (and the associated gain computation) are not ideal when the partitioning solution is to be used as the basis of hierarchical placement. If a net e has more than one pin in both C_1 and C_2, then e may cross the cut more than once after routing. For example, let $v_1, v_3, v_5 \in C_1$ and $v_2, v_4, v_6 \in C_2$ be the modules of a 6-pin net e, and assume that the cut between C_1 and C_2 corresponds to a vertical slice of the placement. If module $v_i \in e$ is placed in row i, then if e is routed to minimize vertical wirelength, e will cross the cut six times. Thus, for each net e, the authors of [170] consider every possible assignment of modules in e to rows $1, 2, \ldots, |e|$, compute the number of times e will cross the cut in routing size for each configuration, then average the results to derive an expected cutsize for e. This expected cutsize is used to compute module gains.

3.5.2 Relaxed Size Constraints

In practice, an exact bisection is not required, yet if cluster size constraints are removed, an unbalanced bipartitioning will result. This has motivated several alternative strategies. Wei and Cheng [187] proposed the following iterative approach to optimize the ratio cut objective. First, pick random modules v and w, form the bipartitioning $\{C_1, C_2\}$ where $C_1 = \{v\}$ and $C_2 = V - \{v\}$, and iteratively move modules from C_1 to C_2 to optimize ratio cut, until $C_1 = V - \{w\}$ and $C_2 = \{w\}$. This step is repeated with the roles of w and v reversed, and the best of the up to $2n - 2$ ratio cut bipartitionings created is taken as the initial solution. Then, an iterative shifting procedure is repeatedly applied: a right (left) shift iteratively moves modules from $C_1(C_2)$ to $C_2(C_1)$ to optimize ratio cut until $C_1(C_2) = \emptyset$. The solution with smallest ratio cut generated during a shift starts the next iteration, and left and right shifts are alternated. Finally, Wei and Cheng apply an FM variant in which (i) there are no size constraints, and (ii) if two modules have the same gain in terms of net cut, then the module which causes the larger reduction in ratio cut is selected to be moved. In [188], the same authors have also applied a hierarchical version of their algorithm to obtain clusterings for use within two-phase FM (see Section 6).

Park and Park [144] modify Sanchis' algorithm to handle relaxed balance constraints, using the cost function: $|E(P^k)| + R \cdot C_{Bal}(P^k)$ where

$$C_{Bal}(P^k) = \sum_{1 \le i < j \le k} \mid w(C_i) - w(C_j) \mid$$

and R is a user-defined parameter. The term $C_{Bal}(P^k)$ is minimized when cluster sizes are perfectly balanced and increases as clusters become unbalanced. Sanchis's algorithm is applied without lookahead gain vectors; The algorithm maintains two gain bucket structures, one for cutsize as in traditional FM and one for C_{Bal}.

3.5.3 FPGA Partitioning

FPGA partitioning poses qualitatively different challenges than Min-Cut partitioning due to hard size and pin constraints implicit in mapping onto prescribed devices. Woo and Kim [189] consider additional "cell-type" constraints for the devices: each FPGA device has an upper bound (capacity) on the number of modules of a given cell-type that can be assigned to the device. Woo and Kim proposed a k-way extension of FM in which the objective is to minimize the maximum number of I/Os used by the devices while satisfying type constraints and upper bounds on cluster size and I/O. Like FM, this algorithm seeks to move the highest-gain module, although many modules may have to be examined before finding a feasible move. Thus, the approach may be viewed as multi-way FM with a more complex gain function and certain moves being forbidden. A related satisficing formulation for MCM partitioning was addressed by Chen et al. [42]; their method solves the linear programming relaxation of an integer program, converts to a feasible solution, and then applies a KL-based post-processing step.

Kužnar et al. [125] recursively apply FM bipartitioning to address the Multiple Device FPGA problem. For a given library of devices and number of modules in the circuit, an integer linear program (ILP) can be solved to find a set of devices that yields a lower bound on cost. This device set would form the optimal solution if the circuit had no interconnections; however, it may not be possible to map the circuit onto these devices while satisfying I/O constraints. The algorithm picks the largest device D from this device set, and applies FM bipartitioning to yield $\{C_1, V - C_1\}$ where C_1 is feasible for D. The ILP is resolved for the remaining subcircuit $V - C_1$ and the largest device D' is chosen from the set and FM is run on $V - C_1$ to yield $\{C_1, C_2, V - C_1 - C_2\}$ with C_2 feasible for D'. This continues until the ILP yields a single device that is the smallest device onto which the remaining subcircuit can be feasibly mapped. Note that the objective function is not exactly min-cut, due to the existence of system I/O pins in the initial circuit description. For example, a net e that contains a system I/O pin on C_1 will require an additional terminal on the C_1 device if e has a pin in $V - C_1$. Kužnar et al. minimize an objective consisting of the number of nets cut times the number of terminals in the remainder of the subcircuit (the solution cost is infinite if device constraints are violated). They also give a slight, but perhaps promising, modification to the FM paradigm. During an FM pass, their implementation records both the best and the second-best solutions encountered. When a pass starting from the best solution does not yield any improvement, a second pass starting from the second-best solution is performed. Only if this pass also fails to yield improvement does the algorithm terminate. In [126], Kuznar et al. extend this algorithm to include functional replication (see Section 2.6).

3.5.4 Layout-Driven Formulations

To map modules onto an underlying tree structure, Vijayan [186] applies a variant of Sanchis' algorithm but with no lookahead gains. The overall procedure is the same, but the gain computation and update are complicated since the objective is the cost of routing nets on the tree structure. A speedup of Vijayan's heuristic was given in [182].

If the underlying structure is a 2×2 grid, a quadrisection formulation results. The motivation is that applying alternating horizontal and vertical cuts in hierarchical min-cut placement fewer cuts in the first direction chosen. The quadrisection formulation can trade off between vertical and horizontal routing resources according to a user-specified parameter. Suaris and Kedem [178] adapted FM to this problem: there are 12 possible ways to move modules from cluster C_i (four choices) to C_j (three choices), so 12 different gain bucket structures are used to store the move types. The authors of [178] construct a placement by using the terminal propagation technique of [60] and recursively quadrisecting the clusters.

Shih, Kuh, and Tsay [174] have applied a multi-way KL variant to MCM partitioning with cluster timing, area, thermal and pin constraints, where the objective is to minimize wirelength over the MCM configuration. An initial solution that satisfies timing constraints is constructed by merging modules with timing dependencies into "super-nodes". The super-nodes are then greedily packed into k clusters such that all the constraints are satisfied. Finally, multi-way KL is run on the clusters such that all the modules in a super-node must be moved together, and only moves which satisfy the constraints are permitted.

3.5.5 DAG Partitioning

Finally, we note variants which partition combinational Boolean networks. Since the input $G(V, E)$ is a DAG, a partitioning of V induces a *dependency graph* with k nodes, each corresponding to a cluster. The directed edge (C_i, C_j) is an edge in the dependency graph if $\exists u \in C_i, v \in C_j$ such that $(u, v) \in E$. Cong et al. [50] define a partitioning to be *acyclic* if it induces an acyclic dependency graph (which is desirable for pipelining or parallel circuit simulation applications). Since a random initial solution is not guaranteed to be acyclic, initial solutions are constructed using random topological sorts. Sanchis' algorithm is then applied, with moves restricted so as to not create a cycle in the dependency graph. The approach also uses MFFC clustering (see Section 6), within a two-phase methodology. An interesting approach of [22] applies FM and a resynthesis technique of [105] to partitioning Boolean networks with functional information. The signals directed from C_i to C_j are encoded in C_i. Since some of the signals might carry redundant information, this can reduce the number of signals from C_i to C_j. Corresponding logic must be added to decode (or reinterpret) these signals in C_j.

3.5.6 Perspectives on "Iterative Variants"

This subsection has surveyed a "litany of variants" arising from the core FM, KL, and Sanchis algorithms. Many of the modifications represent rather simple ideas, such as allowing locked modules to become unlocked, keeping the best and second best solutions, allowing net moves in addition to module moves, etc. More importantly, it can be unclear which of these modifications really improves the algorithm and which are "simply different". (For example, the partitioning study [94] concludes that dual passes are not worth the extra runtime.) As we have previously discussed, small changes in implementation can greatly affect performance, so further study of modifications to iterative methods is likely to be worthwhile; on the other hand, very detailed and systematic investigations are required if meaningful conclusions are to be drawn.

The "litany of variants" also reveals that applying an FM-based algorithm to yet another problem formulation can require varying degrees of innovation. For example, the balance term in the cost function of [144] is obvious: increase the penalty proportionally to the deviation from bisection. On the other hand, certain changes in formulation may appear simple (e.g., partitioning onto an underlying tree structure, by Vijayan [186]), but issues such as efficient gain updating can make the implementation differences non-trivial. Many new problem formulations seem very similar to previous ones (modulo some extra constraints or modified cost functions), hence care must be taken to discern the works which more substantially advance the field through their combination of relevance, non-obvious solution, and experimental methodology.

4 Geometric Representations

A geometric representation of the circuit netlist can provide a useful basis for a partitioning heuristic, since speedups and special "geometric" heuristics become possible. For example, computing a minimum spanning tree of a weighted undirected graph requires $O(n^2)$ time, in general, but only $O(n \log n)$ time for points in 2-dimensional geometric space [150]. Single-source shortest path, all-pairs shortest paths, and matching are other examples of problems that can be solved more efficiently for geometric instances. This section discusses partitioning approaches based on finding a geometric representation of a graph or hypergraph and applying "geometric" algorithms to find a partitioning solution. We focus on three primary types of representations:

- A **1-dimensional representation** or a *linear ordering* is a sequential list of the modules. Generally, modules that are closely connected should lie close to each other in the ordering, so that the ordering can reveal the netlist's structure. Indeed, problems in the sparse matrix computation literature such as finding minimum bandwidth, minimum profile, and minimum fill-in orderings exactly correspond to this type of ordering problem [147]. A linear ordering may also be viewed as a 1-dimensional placement, and vice-versa; consequently, some of the partitioning approaches we discuss below were originally designed for 1-dimensional placement

- A **multi-dimensional representation** is a set of n points in d-dimensional space with $d > 1$, where each point maps to (represents) a unique module. This representation implicitly defines a distance relation between every pair of modules, e.g., the Euclidean distance between their corresponding points. Geometric clustering algorithms may be applied to the set of points, possibly in conjunction with other graph-based algorithms. Such a representation is also common 2-dimensional cell placement.

- A **multi-dimensional vector space** representation can arise in two distinct ways. Using one approach, the vector space consists of indicator n-vectors (corresponding to bipartitioning solutions), and the problem becomes one of finding the direction of the best indicator vector [68]. Using the other approach, the n modules are mapped to n vectors in d-space and the vectors are clustered together to form both a vector partitioning and a module partitioning [12] [6] [37]. A major advantage of the vector space approach is that spectral methods can be used to construct a vector space that optimally captures the netlist information vis-a-vis partitioning, i.e., the optimal indicator vector direction or the optimal vector partitioning solution will map to the optimal graph partitioning solution.

Spectral methods are of primary importance in constructing geometric representations; their discussion requires the following notation. Assume the netlist is represented as a weighted undirected graph $G(V,E)$ with adjacency matrix $A = (a_{ij})$. The $n \times n$ *degree matrix* D is given by $d_{ii} = deg(v_i)$ with $d_{ij} = 0$ if $i \neq j$. The $n \times n$ Laplacian matrix of G is defined as $Q = D - A$. An n-dimensional vector $\vec{\mu}$ is an *eigenvector* of Q with *eigenvalue* λ if and only if $Q\vec{\mu} = \lambda\vec{\mu}$. We denote the set of eigenvectors of Q by $\vec{\mu_1}, \vec{\mu_2}, \ldots, \vec{\mu_n}$ with corresponding eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. We assume that the eigenvectors are normalized, i.e., for $1 \leq j \leq n$, $\vec{\mu_j}^T\vec{\mu_j} = ||\vec{\mu_j}||^2 = 1$. Let Δ_d denote the $d \times d$ diagonal *eigenvalue matrix* with entries $\lambda_1, \lambda_2, \ldots, \lambda_d$, and let $U_d = (\mu_{ij})$ denote the $n \times d$ *eigenvector matrix* with columns $\vec{\mu_1}, \vec{\mu_2}, \ldots, \vec{\mu_d}$. Notice that $Q = U_n \Delta_n U_n^T$. Some works use the eigenvectors of A instead of Q; however, the Laplacian has becoming more popular because it has the following following desirable properties (the first two due to Q being positive semi-definite) [139]:

- The eigenvectors are mutually orthogonal, and hence form a basis in n-dimensional space.
- Each eigenvalue λ_j of Q is real.
- The smallest eigenvalue $\lambda_1 = 0$ and has a corresponding eigenvector $\vec{\mu_1} = [\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}}]^T$.[8]
- If G is connected then $\lambda_2 > 0$. (In general, the multiplicity of 0 as an eigenvalue of Q is equal to the number of connected components of G.)

[8]This property combined with the orthogonality property implies that the entries of each $\vec{\mu_j}$ with $j > 1$ sum to zero. This does not necessarily hold for the spectra of A, and further the eigenvalues of A will generally be both positive and negative. However, in practice, a diagonal matrix B is often added to A to ensure positive semi-definiteness and the computability of eigenvectors, and also to obtain properties similar to Q (e.g., Donath and Hoffman [59] use $B = -D$, thereby obtaining the spectra of -Q). Currently, the relationship between the spectra of A and Q are not well-understood, though we believe that most theoretical results in the spectral literature can be equivalently derived with either Q or A. To this end, we discuss as many of the works in this section as possible in terms of Q, even if the original paper used A. For example, the discussion of Barnes's work [18] below is given in terms of Q, which allows us to establish the equivalence of lower bounds due to Barnes and to Donath and Hoffman. By "equivalence", we mean that following the theorem derivation of each work using the spectra of Q leads to the identical result.

The remainder of this section is organized as follows. First, we discuss the one-dimensional placement algorithm of Hall [89], and see how the second eigenvector $\vec{\mu_2}$ gives the optimal solution for minimum-squared wirelength placement. A linear ordering of modules is obtained by sorting the entries of $\vec{\mu_2}$, and a heuristic bipartitioning can be derived from splitting the ordering. Next, we describe an extension of Hall's multi-way partitioning approach due to Barnes [18], as well as some other extensions (e.g., [79] [37]). These approaches use k eigenvectors to derive a k-way partitioning; Barnes' approach explicitly tries to map each cluster to a single eigenvector. Third, we discuss an alternative approach that uses multiple eigenvectors to generate a bipartitioning. This *linear probe* technique [68] first constructs a vector space using the eigenvectors as a basis, then generates probes which search the vector space to find good bipartitionings. Fourth, we discuss utilizing this vector space to construct a vector partitioning problem that is equivalent to many multi-way graph partitioning formulations. Finally, we discuss techniques for deriving partitioning solutions from a linear ordering, and show how many of the approaches presented in this section can be used to construct such linear orderings.

4.1 Hall's Quadratic Placement

As defined in Section 2.1, we let $\vec{x}$ denote the $0-1$ indicator vector for a given bipartitioning P^2. The Min-Cut bipartitioning objective ($F(P^2) = E(C_1)$) can now be written as

$$2 \cdot F(P^2) = \vec{x}^T Q \vec{x} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}(x_i - x_j)^2 \tag{4.1}$$

All the $(x_i - x_j)^2$ terms in the double summation of equation (4.1) are zero, unless v_i and v_j are in different clusters. In this case, $(x_i - x_j)^2 = 1$, and the cost a_{ij} of each cut edge appears twice in the sum, so the double summation evaluates to $2 \cdot F(P^2)$. Also observe that equation (4.1) gives the squared wirelength of the 1-dimensional placement given by $\vec{x}$, even when the coordinates are allowed to be *non-discrete*, i.e., the x_i's are not restricted to integer values. Under the constraint $||x||^2 = \vec{x}^T\vec{x} = 1$, Hall [89] showed that $\vec{x} = \vec{\mu_2}$ gives the optimal nontrivial 1-dimensional placement, with squared wirelength equal to λ_2 (note that $\vec{\mu_1}$ gives the trivial zero-wirelength solution with all modules placed at coordinate $\frac{1}{\sqrt{n}}$).

The significance of Hall's result is that it provides the optimal non-discrete solution for Min-Cut Bipartitioning. Although a non-discrete solution for $\vec{x}$ is meaningless, this result suggests heuristically finding the discrete solution "closest" to $\vec{\mu_2}$. Given cluster size constraints $|C_1| = m_1$ and $|C_2| = m_2$, the closest discrete solution is obtained by sorting the coordinates of $\vec{\mu_2}$ – the m_1 modules with the highest coordinates are placed in C_1 and the m_2 lowest modules with the lowest coordinates are placed in C_2. The reverse assignment is also attempted, and one of these solutions is guaranteed to be closest to $\vec{\mu_2}$ as measured by the Frobenius norm. One can view this sorting algorithm as moving the smallest coordinates to C_1 (corresponding to $x_i = 0$) and the largest coordinates to C_1 (corresponding to $x_i = 1$).[9] This approach of finding the partitioning which best approximates the second smallest eigenvector was first used by Barnes [18] and is commonly known as *spectral bisection* (when $m_1 = m_2 = \frac{n}{2}$). This algorithm has also been widely used by the sparse matrix computation community; Pothen et al. [148] have used it as the basis of a vertex separator algorithm and Hendrickson and Leland [96] have extended it to partitioning onto hypercube or mesh architectures. Also, Hagen and Kahng [80] extended spectral bisection to ratio cut bipartitioning by choosing the best ratio cut that results from each possible split of the sorted coordinates of $\vec{\mu}_2$.

We observe that spectral bisection may perform arbitrarily worse than optimal, as illustrated by the following example. Consider the graph in Figure 8 in which each circle represents a clique, so that two $\frac{n}{4}$-cliques A and C are each connected by a single edge to the $\frac{n}{2}$-clique B. Since $\vec{\mu}_2$ gives the optimum 1-dimensional squared wirelength placement, sorting the entries in $\vec{\mu}_2$ yields the coordinates for modules in A that are followed by coordinates for those in B, that are in turn followed by coordinates for those in C. Consequently, spectral bisection will split B into equal halves, cutting $(\frac{n}{4})^2 = \frac{n^2}{16}$ edges. This solution is an $\Omega(n^2)$ factor worse than the optimum bisection cutsize of two edges, obtained by assigning A and C to one cluster and B to the other.

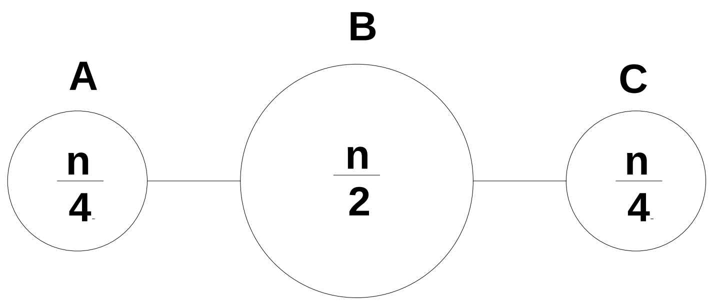


Figure 8: Pathological instance for spectral bisection. Each circle represents a clique of modules.

4.2 Mapping k Clusters to k Eigenvectors

Hall's approach can be extended to multi-way partitioning by associating each cluster with an eigenvector with small eigenvalue. The 1-dimensional placement solution $\vec{\mu}_2$ yields the minimum squared wirelength of λ_2, but $\vec{\mu}_3$ gives the next best solution with cost λ_3, etc. – again subject to the $||\vec{x}||^2 = 1$ constraint. For each of these high-quality 1-dimensional eigenvector placements, the closest indicator

[9]This interpretation is not exactly correct since $\vec{x} = \vec{\mu}_2$ has $\sum_{i=1}^{n} x_i = 0$ while the sum for an indicator vector of a bisection is $\frac{n}{2}$. Furthermore, the indicator vector for a bisection also has $||\vec{x}||^2 = \frac{n}{2}$ as opposed to $||\vec{\mu}_2||^2 = 1$. These inconsistencies are addressed by using coordinates $(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$ instead of $(0, 1)$ for the indicator vector. Note that these coordinates correspond to the ratioed assignment matrix R defined in the next subsection.

vector can be found, and the k indicator vectors can be used to construct a k-way partitioning.

Assume that the standard multi-way partitioning objective $F(P^k) = \sum_{h=1}^k |E(C_h)|$ applies, and that prescribed cluster sizes $m_1 \geq m_2 \geq \ldots \geq m_k \geq 0$ are given. Let M denote the $k \times k$ diagonal matrix with entries $m_1, m_2, \ldots, m_k$. Although X denotes the assignment matrix for P^k, sometimes it will be advantageous to represent P^k as the $n \times k$ *ratioed assignment matrix* $R = (r_{ih})$ where $r_{ih} = \frac{1}{\sqrt{|C_h|}}$ if $v_i \in C_h$ and $r_{ih} = 0$ otherwise. Notice that $\vec{R_h}$, the h^{th} column of R, has magnitude one, which satisfies the constraint in Hall's 1-dimensional placement formulation. Let $P = (p_{ij})$ be the $n \times n$ *partition matrix* with $p_{ij} = 1$ if v_i and v_j are in the same cluster and $p_{ij} = 0$ otherwise; observe that $P = XX^T = RMR^T$. Finally, define the $n \times n$ *ratioed partition matrix* P^R to be a scaled version of P with ij entry equal to $\frac{1}{|C|}$ if v_i and v_j are in the same cluster C, and 0 otherwise (so $P^R = RR^T$). Barnes noted that

$$||Q + P||^2 = ||Q||^2 + 2 \sum_{1 \leq i,j \leq n} q_{ij} p_{ij} + ||P^2|| = ||Q||^2 + 2F(P^k) + \sum_{i=1}^k m_i^2 \tag{4.2}$$

Since $||Q||$ and $\sum_{i=1}^k m_i^2$ are fixed constants, minimizing $||Q+P||^2$ and minimizing $F(P^k)$ are equivalent partitioning objectives. The Hoffman-Wielandt inequality states that if A and B are real $n \times n$ symmetric matrices with eigenvalues $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_n$ and $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_n$ respectively, then

$$||A - B||^2 \geq \sum_{i=1}^n (\alpha_i - \beta_i)^2.$$

It is not hard to show that the eigenvalues of $-P$ are $-m_1, -m_2, \ldots, -m_k, 0, 0, \ldots, 0$, which yields

$$||Q - (-P)||^2 \geq \sum_{i=1}^k (\lambda_i - (-m_i))^2 + \sum_{i=k+1}^n (\lambda_i - 0)^2 = \sum_{i=1}^k (\lambda_i + m_i)^2 + \sum_{i=k+1}^n \lambda_i^2 \tag{4.3}$$

Combining equation (4.3) with equation (4.2) yields the equivalent of the famous Donath and Hoffman [59] lower bound: $F(P^k) \geq \sum_{i=1}^k \lambda_i m_i$.[10] Actually, the lower bound of [59] is stronger, since it holds for not only the eigenvalues of Q, but also the eigenvalues of $D - A$ as long as the diagonal matrix D has $trace(D) = \sum_{i=1}^n deg(v_i)$. It is thus possible to varying D to increase $\sum_{i=1}^k \lambda_i m_i$ and improve the lower bound.

Observe that the Donath-Hoffman lower bound also implies that an optimal partitioning solution would be obtained if each $\vec{R_i}$ could be set to $\vec{\mu_i}$. However, setting $R = U_k$ generally does not yield a valid ratioed partition matrix. Barnes [18] justifies his approach to finding a valid R that best

[10] Combining the two equations yields

$$2F(P^k) \geq -||Q||^2 - \sum_{h=1}^k m_i^2 + \sum_{i=1}^k (\lambda_i + m_i)^2 + \sum_{i=k+1}^n \lambda_i^2 = -||Q||^2 + 2\sum_{i=1}^k \lambda_i m_i + \sum_{i=1}^n \lambda_i^2.$$

It is not difficult to show that $||Q||^2 = ||\Delta_n||^2$ which yields the desired result.

approximates U_k as follows. Since $Q = U_n \Delta_n U_n^T$ and $P = RMR^T$, equation (4.2) implies that the partitioning objective can be written as

$$||Q + P||^2 = ||U_n \Delta_n U_n^T + RMR^T||^2 = ||\Delta_n + U_n^T RM(U_n^T R)^T||^2 \tag{4.4}$$

If R could be chosen such that $U_n^T R = J$, where J is the $n \times k$ matrix having the $k \times k$ identity matrix I_k as its first k rows and all other entries zero, we would have equality in equation (4.3), and R would represent the optimal partitioning solution. Since such a choice of R is generally impossible, Barnes chooses to minimize the error $||U_n^T R - J||^2 = ||U_k - R||^2$ by setting up a transportation problem which can be solved efficiently (see Section 5.3.3). To minimize this error for a given vector $\vec{R_i}$ (representing C_i), $|\vec{\mu_i}^T \vec{R_i}|$ should be maximized; this occurs when the nonzero entries of $\vec{R_i}$ are the m_i largest (or smallest) coordinates of $\vec{\mu_i}$. Of course, it may not be possible to choose each $\vec{R_i}$ in this fashion, because a module might be assigned to more than one cluster; Barnes' transportation formulation optimally resolves this conflict. Notice that for $k = 2$, this formulation reduces to minimizing $||U_2 - R||^2$ and the optimal solution is derived by sorting the coordinates of $\vec{\mu_2}$ as discussed above.

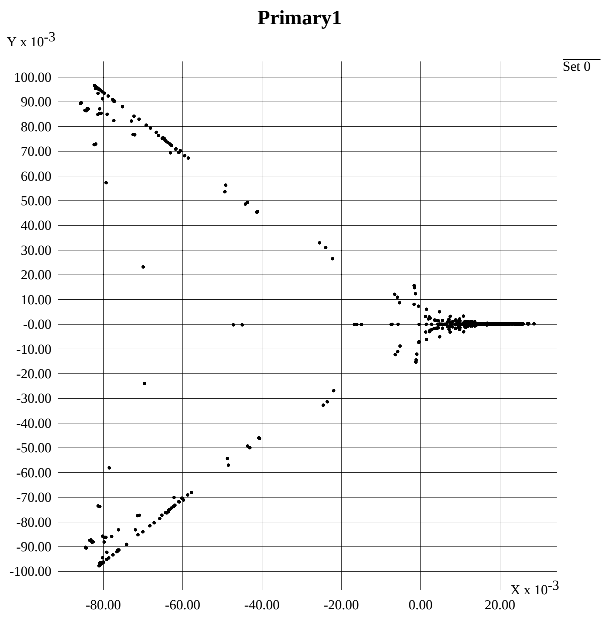
Barnes' algorithm was applied to VLSI circuits by Hadley et al. [79], who also incorporated a new clique net model to obtain a graph representation and FM post-processing. Vannelli and Rowan [185] also applied the algorithm to two variant constructions of the adjacency matrix, which respectively use $a_{ij} = |N(v_i)\backslash N(v_j)| + |N(v_i)\backslash N(v_j)|$ and $a_{ij} = (|N(v_i)\backslash N(v_j)| + |N(v_i)\backslash N(v_j)|)/(|N(v_i)| + |N(v_j)|)$; observe that these graph representations of the hypergraph do not preserve sparsity. Rendl and Wolkowicz [154] observe that Barnes' approach "only makes sense" if U_k is very close to an optimal solution. Consequently, [154] first relaxes the integer constraints of the assignment matrix X to find an X close to U_k, then uses the transportation formulation to find the closest legal solution to X. They also permit the diagonal entries of A to be perturbed before computing the eigenvectors, leading to tighter lower bounds than those given by Donath and Hoffman [59]. Further, they extend the average case bisection bound of Boppana [27] to $k > 2$ (see [133] for a more complete discussion of spectrally derived lower bounds).

Chan et al. [37] extend the Donath and Hoffman bound $F(P^k) \geq \sum_{i=1}^k \lambda_i m_i$ to Scaled Cost, proving for this objective that $F(P^k) \geq \sum_{i=1}^k \lambda_i$. They argue that since R is an approximation of U_k, than approximation for $P^R = RR^T$ should be given by $U_k U_k^T$, i.e., the matrix having as its ij^{th} entry the dot product of the i^{th} row of U_k with the j^{th} row of U_k. Viewing each module v_i as a vector in k-dimensional space with coordinates given by the i^{th} row of U_k, the ij^{th} entry of $U_k U_k^T$ is the angle or *directional cosine* between modules v_i and v_j. Chan et al. consider the directional cosine between two modules as a distance measure: the larger the directional cosine, the more likely the two modules should be placed in the same cluster. A directional cosine of zero between v_i and v_j implies that the ij^{th} entry in P_R is zero (since P_R should approximate $U_k U_k^T$), meaning that v_i and $v_j$should be assigned to different clusters. The KP algorithm of [37] finds an orthogonal basis with k "prototype vectors" and constructs k clusters by assigning each module to its closest prototype according to the directional cosines measure. Chan et al. [38] have also adapted their KP approach to FPGA partitioning by assigning each module to its closest prototype while observing size and I/O constraints on the clusters (FPGA devices).

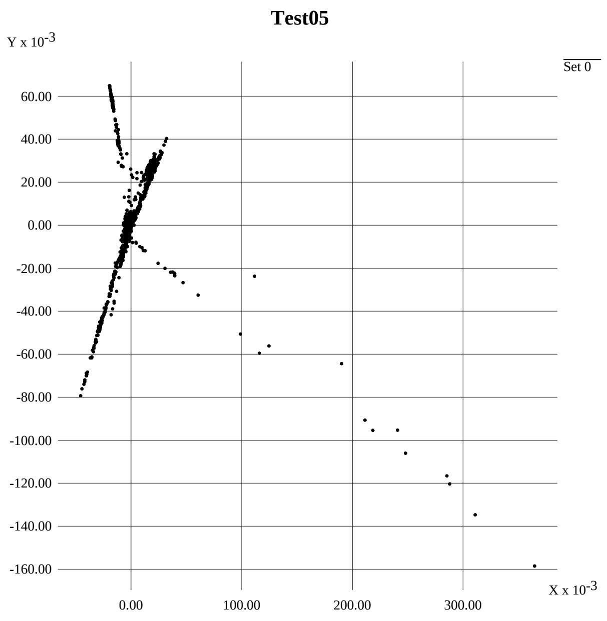
We believe that the Barnes and KP approaches may have limited performance potential. Both approaches assume that U_k can be perturbed to give a legal ratioed assignment matrix R: Barnes chooses R to minimize the error $||U_k - R||^2$, and Chan et al. explicitly assume that R is an approximation of U_k. Thus, each $\vec{R_i}$ representing cluster C_i should be close to $\vec{\mu_i}$ in the ideal case, but for real VLSI circuits $\vec{R_i}$ and $\vec{\mu_i}$ will generally be quite different. A given eigenvector will typically have a few outliers – modules with large coordinates – but a large majority of modules will have coordinates very close to zero; these coordinates would have to be perturbed considerably to achieve a valid ratioed assignment matrix. To see this, consider the coordinates for $\vec{\mu_2}$ (on the x-axis) and $\vec{\mu_3}$ (on the y-axis) for the Primary1 and Test05 benchmarks shown in Figure 9. Exactly 603 of the 833 modules for Primary1 have zero (to three decimal places) as their y-coordinate, and these modules are strongly clustered together. The embedding actually admits a very natural 3-way partitioning with sizes $681, 76, 76$ but does not reveal a natural bisection. This phenomenon is even more apparent for Test05, as 12 outliers with x-coordinate larger than 0.1 force a strong agglomeration of points around zero, again leading to a difficult clustering task. We believe that even when minimized, the error $||U_k - R||^2$ will usually be large, i.e., each $\vec{R_i}$ will not be close to its corresponding $\vec{\mu_i}$. This potentially large error may be magnified by squaring U_k: Chan et al. assume that $P^R = RR^T$ is an approximation of $U_k U_k^T$ (but their KP algorithm does not actually square U_k, since computing all n^2 entries of $U_k U_k^T$ would be too expensive).

Hall [89] also proposed a method for multi-way partitioning using the same eigenvector embedding as the authors of[37]. Since $\vec{\mu_2}$ gives the optimal 1-dimensional placement with squared wirelength, Hall reasoned that the other low-cost eigenvector placements could also be utilized. He suggested placing the the coordinates of d eigenvectors orthogonally to yield a d-dimensional geometric embedding of the netlist, as in Figure 9. Hall then proposed to "cluster" the embedding, but did not specify a heuristic. Alpert and Kahng [3] explored Hall's idea by applying geometric clustering algorithms to these spectral embeddings, using Euclidean distance as the dissimilarity measure between pairs of modules. In [5], they constructed a 1-dimensional ordering over the same d-dimensional embedding using a spacefilling curve heuristic to the traveling salesman problem [20]. The ordering was then split into a partitioning using dynamic programming (see Section 4.5).

We believe that neither directional cosines nor Euclidean distance is the proper similarity (or



(a)



(b)

Figure 9: The 2-dimensional embedding of the (a) Primary1 and (b) Test05 MCNC benchmarks, with $\vec{\mu_2}$ plotted on the x-axis and $\vec{\mu_3}$ plotted on the y-axis.

dissimiliarity) measure between spectrally embedded modules. The theoretical results described in the next two subsections indicate that the "proper" spectral embedding should be scaled by the eigenvalues, and that the module similarity measure should be a "vector sum".

4.3 Probes in Multi-Dimensional Vector Space

We have noted that a potential difficulty with the approaches of [18] [37], namely, that each cluster is associated with a single eigenvector, but the cluster's indicator vector may be far from its eigenvector. Blanks [24] also shared this intuition and illustrated (for placement, though it also holds for bipartitioning) that the closest legal solution to the eigenvector solution $\vec{\mu_2}$, i.e., the indicator vector that maximally projects onto $\vec{\mu_2}$, will generally not be optimum. Figure 10 gives an abstract visualization of this concept. The solution space consists of all n-dimensional vectors $\vec{x}$, the plane represents the subspace of all legal indicator vectors, and the optimum non-discrete solution for min-cut partitioning is given by the point $\vec{x} = \vec{\mu_2}$. The ellipsoids correspond to constant cost degradation, i.e., the smallest ellipsoid is the set of solutions with cost $\lambda_2 + \epsilon$, the second smallest ellipsoid contains solutions with cost $\lambda_2 + 2\epsilon$, etc; Blanks proved that these cost surfaces are indeed ellipsoidal. The optimum legal bipartitioning solution B is the intersection of the subspace of legal solutions with the smallest possible ellipsoid (such that this intersection is nonempty). The indicator vector $\vec{x}$ that maximally projects onto $\vec{\mu_2}$, labeled with A in the Figure, certainly does not have to be identical to B.

An alternative approach to finding the closest legal solution might be to combine several high-quality eigenvector solutions into a new solution that is closer to the subspace of legal solutions. The same figure shows a combined solution whose closest legal solution, C, is closer to optimal than A. Note that all linear combinations of the first d eigenvector solutions lie in the d-dimensional subspace spanned by these eigenvectors. This observation is the genesis behind the probing approach Frankle and Karp [68] [67]: they search this entire subspace for a good solution rather than associating a solution with a single eigenvector.

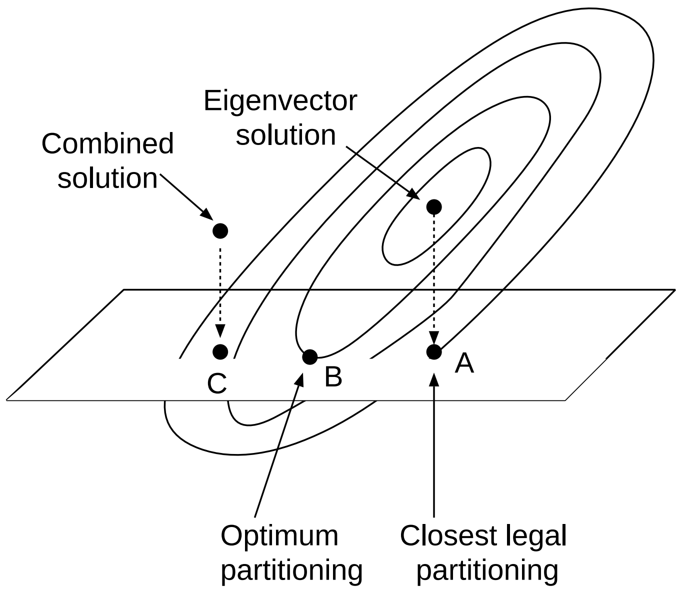


Figure 10: Abstraction of Hall's eigenvector approach as shown in [24].

Frankle and Karp [68] developed the following probing technique for finding such non-discrete combined solutions in the multi-dimensional eigenspace. Since the n eigenvectors of Q form an orthogonal n-dimensional basis, any indicator vector can be expressed as a sum of projections onto each of the eigenvectors, i.e., $\vec{x} = \sum_{j=1}^{n} \alpha_j(\vec{x})\vec{\mu_j}$ where $\alpha_j(\vec{x}) = \vec{x}^T\vec{\mu_j}$. Substituting this value for $\vec{x}$ in equation (4.1) yields

$$F(P^2) = \sum_{j=1}^{n} \alpha_j(\vec{x})^2 \lambda_j \tag{4.5}$$

If the cluster sizes $|C_1| = m_1$ and $|C_2| = m_2$ are fixed, then $\sum_{j=1}^{n} \alpha_j(\vec{x})^2 = m_2$ is a constant (since $\vec{x}$ is the indicator vector for C_2). This observation, along with equation (4.5), implies that the goal is to find an indicator vector $\vec{x}$ that projects maximally onto the eigenvectors with small eigenvalues: The eigenvalue λ_i is the coefficient for the $\alpha_i(\vec{x})$ term; hence, when λ_i is large $\alpha_i(\vec{x})$ should be small, and vice versa. If the large eigenvalues are ignored and only the first d terms in the sum are considered, the problem becomes one of finding the indicator vector with maximal projection onto the subspace spanned by the first d eigenvectors. Frankle and Karp express equation (4.5) as the following maximization problem:

$$\text{Maximize:} \quad Hm_2 - F(P^2) = H\sum_{j=1}^{n}\alpha_j(\vec{x})^2 - \sum_{j=1}^{n}\alpha_j(\vec{x})^2\lambda_j = \sum_{j=1}^{n}\alpha_j(\vec{x})^2(H-\lambda_j) \qquad (4.6)$$

for some $H \geq \lambda_n$. Let V_d denote the $n \times d$ *scaled eigenvector matrix* with column j equal to $\vec{\mu_j}\sqrt{H-\lambda_j}$. The right hand side of equation (4.6) can now be expressed as $||\vec{x}^T V_n||$. Given any n-dimensional vector $\vec{y}$, i.e., a *probe direction*, Frankle and Karp show that the legal indicator vector $\vec{x}$ that maximizes $\vec{x}^T V_n \vec{y}$ can be found efficiently by sorting and splitting the entries of $V_n\vec{y}$, as in spectral bisection. Here, the vector $\vec{y}$ serves as the combined solution point in Figure 10. Figure 11 illustrates this approach for two dimensions. The vector $\vec{x}^T V_2$ that projects maximally onto $\vec{y}$ gives the optimal solution vector for probe direction $\vec{y}$.

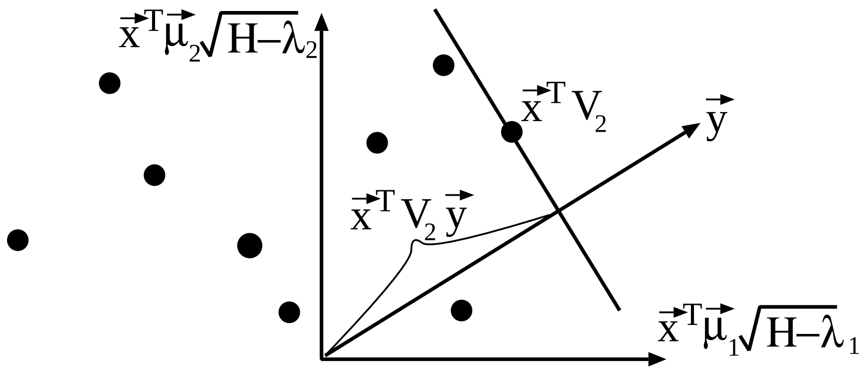

$\vec{x}^T\vec{\mu}_2\sqrt{H-\lambda_2}$
$\vec{x}^T V_2$
$\vec{y}$
$\vec{x}^T V_2 \vec{y}$
$\vec{x}^T\vec{\mu}_1\sqrt{H-\lambda_1}$

Figure 11: Illustration of the probing approach of [68].

Frankle and Karp heuristically search over various d-dimensional vector probes $\vec{y}$, and find for each the indicator vector $\vec{x}$ that maximizes $\vec{x}^T V_d \vec{y}$. Both randomized and iterated probing techniques are used to construct candidate probe vectors. By restricting $\vec{y}$ to d dimensions, only vectors that lie in the subspace spanned by the first d eigenvectors are considered. We believe that this is a sound strategy, given the equation (4.5) intuition that a good solution $\vec{x}$ should strongly project onto the first few eigenvectors. The problem reduces to finding a probe direction $\vec{y}$ that is as close as possible to a legal solution. Note that for $d = 2$, any probe vector will return the solution obtained by sorting and splitting the entries of $\vec{\mu_2}$.

4.4 Vector Partitioning

The probe approach can also be viewed another way [6]: for each module v_i, observe that the d-dimensional vector corresponding to the i^{th} row of V_d is actually the indicator vector for the single-module cluster $\{v_i\}$ after projecting onto the subspace spanned by the first d eigenvectors, and scaling the j^{th} coordinate by $\sqrt{H-\lambda_j}$, $1 \le j \le d$. If $\vec{x}$ is the 0-1 indicator vector for $\{v_i\}$, then its projection onto the scaled eigenspace is

$$\vec{y_i} = [\vec{x}^T\vec{\mu_1}\sqrt{H-\lambda_1}, \vec{x}^T\vec{\mu_2}\sqrt{H-\lambda_2}, \ldots, \vec{x}^T\vec{\mu_d}\sqrt{H-\lambda_d}]^T = [\mu_{i1}\sqrt{H-\lambda_1}, \mu_{i2}\sqrt{H-\lambda_2}, \ldots, \mu_{id}\sqrt{H-\lambda_d}]^T$$

i.e., the i^{th} row of V_d. Observe that if $\vec{Y} = \sum_{i=1}^{n} x_i \cdot \vec{y_i}$, then $\vec{Y} = \vec{x}^T V_d$. Recall that Frankle and Karp showed bipartitioning equivalent to maximizing $||\vec{x}^T V_n||^2$; we now have that bipartitioning is also equivalent to maximizing $||Y||^2$ when $d = n$. We therefore seek a subset of vectors $\vec{y_i}$ (corresponding to $x_i = 1$) whose sum has the largest possible magnitude. Graph bipartitioning exactly reduces to this vector subset problem for $d = n$ and $H = \lambda_n$.

Arun and Rao [12] also noted this vector partitioning formulation, but called it a "geometric clustering" problem. The derivation is somewhat different, relying on the factorability of the adjacency matrix, i.e., if A can be expressed as $A = CC^T$ where C is an $n \times d$ matrix, then A only has rank d. This implies that the columns of C span the same space as the columns (or rows) of A, hence the columns of C form a basis for this space. The indicator vector for $\{v_i\}$ becomes the i^{th} row of C when expressed in this basis. Viewing the rows of C as points in d-space centered at the origin, the bipartitioning problem reduces to finding a subset of points whose center is furthest from the origin.

To find an appropriate matrix C, the authors of [12] invoke a theorem from principal components analysis [152], that the best rank-d approximation to A with respect to both the spectral and Frobenius norms is $U^d \Delta_d U_d^T$. (Following [12], we use the eigenvectors of A so the columns of U_d are $\vec{\mu_1}, \ldots, \vec{\mu_d}$ with eigenvalues $\lambda_1 \ge \ldots \ge \lambda_n$.) This result implies that $C = U_d \Delta^{\frac{1}{2}}$ is the closest approximation to $A = CC^T$, and thus C is "equivalent" to V_d, meaning the reductions of [68] and [12] are also equivalent. Hence, Arun and Rao choose $C = U_d \Delta^{\frac{1}{2}}$. Like Barnes [18], they solve the 1-dimensional geometric clustering problem by sorting the entries of $\vec{\mu_1}$, and the 2-dimensional problem by testing all possible hyperplanes which divide the set of points into two clusters. In [13], they extend their 2-dimensional hyperplane algorithm to d dimensions, finding the optimal vector partitioning in $O(n^{d(d+3)/2})$ time. Frankle [67] has shown how to exhaustively search a d-dimensional vector space with probes, finding the optimal solution in $O(n^{d-1})$ time.

The min-cut bipartitioning reductions of [68] and [12] can be extended to many multi-way formulations, including minimum Scaled Cost, minimum total net cut, or maximum cluster I/O ($|E(C_h)|$ in FPGA partitioning), via a *vector partitioning* formulation [6]. Let S be the set of n vectors

$\{\vec{y_1}, \vec{y_2}, \ldots, \vec{y_n}\}$ which form the n rows of V_d. The authors of [6] showed that min-cut k-way partitioning reduces to the following *vector partitioning* problem: Find k mutually disjoint subsets of vectors $\{S_1, S_2, \ldots, S_k\}$, with $S_1 \cup S_2 \cup \ldots \cup S_k = S$, so as to maximize

$$\sum_{h=1}^{k} ||\vec{Y_h}||^2 \text{ where } \vec{Y_h} = \sum_{\vec{y_i} \in S_h} \vec{y_i}$$

If $d = n$ and $\{S_1, S_2, \ldots, S_k\}$ is the optimum vector partitioning solution, then $\{C_1, C_2, \ldots, C_k\}$ is the optimum vector partitioning solution where $C_h = \{v_i \mid \vec{y_i} \in S_h\}$.

Alpert and Yao propose the greedy MELO algorithm, which does not explicitly construct a vector partitioning, but instead generates a 1-dimensional ordering that is split into a clustering using dynamic programming (see Section 4.5). Given a d-dimensional vector partitioning instance, MELO starts with a set of vectors $S_1 = \emptyset$ and iteratively adds to S_1 the vector $\vec{y_i} \in S - S_1$ that maximizes

$$||\vec{y_i} + \sum_{\vec{y_j} \in S_1} \vec{y_j}||$$

If $\vec{y_i}$ is the j^{th} vector added to S_1 then v_i is the j^{th} module in the linear ordering. MELO can also be interpreted strictly in terms of equation (4.6). Recall that the indicator $\vec{x}$ should maximally project onto the d eigenvectors with smallest eigenvalues. Starting with $\vec{x} = \vec{0}$, MELO iteratively sets $x_i = 1$, for that i which maximizes $\sum_{j=1}^{d} \alpha_j(\vec{x})^2(H - \lambda_j)$, until $\vec{x} = \vec{1}$.

4.5 From Orderings to Partitionings

We have seen that spectral methods can be used to construct various geometric representations of the netlist, including multi-dimensional embeddings and 1-dimensional orderings. The latter type of representation has led to several intuitive and efficient heuristics (e.g., we have discussed constructing a bipartitioning from a 1-dimensional placement by sorting coordinates to obtain a linear ordering of modules, and then splitting this ordering into two clusters), and has received increased attention in recent years. We conclude this section by surveying various methods which construct partitionings from 1-dimensional orderings.

4.5.1 The Net-Based Approach

If a vertex ordering is constructed for the intersection graph or dual graph, then we will actually obtain an ordering of signal nets, rather than of modules. An approach which then splits the net ordering will require an additional step to construct a module partitioning from the resulting net partitioning. In other words, if we are given, say, a bipartition of signal nets, some modules will belong only to nets on one side of the partition and can be unambiguously assigned to that side, but other modules will be *shared* by nets on both sides of the partition (cf. terms such as "boundary graph" [112] [83] or "module contention" [49]). We must therefore seek a *completion* of the net partition which assigns each shared module to a single cluster, such that the partition cost is minimized. This general net-based partitioning approach (i.e., first obtain a net partition, then complete the net partition into a module partition) was established in [83] and extended in, e.g., [48] and [49] (see Section 5.2).

If the nets are partitioned into two sets N_1 and N_2, a module bipartitioning $P^2 = \{C_1, C_2\}$ can be derived by having the nets "vote" to determine each module assignment. The IG-Vote algorithm of Hagen and Kahng [83] begins with all nets in N_1 and all modules in C_1. Iteratively, a net is moved from N_1 to N_2 according to the net ordering. After each net is moved, if any incident module in C_1 has stronger connections to nets in N_2 than to nets in N_1, then this module is moved to C_2. This process generates at most $n-1$ module bipartitionings; the one with smallest ratio cut is chosen. The variant approach of [48] also considers each split of the net ordering, and for each completes the module bipartitioning by using a matching-based approach (see Section 5.2).

4.5.2 Dynamic Programming for Restricted Partitioning

To split a linear ordering into more than two clusters, Alpert and Kahng [5] proposed the DP-RP ("dynamic programming for restricted partitioning") algorithm, which optimally solves the restricted formulation where each cluster of P^k must be contiguous in the linear ordering. Notice that there are at most n^2 contiguous subsets of the ordering, and these represent every possible cluster that can belong to P^k. DP-RP begins by computing the costs for each of these possible clusters. These correspond to the optimal 1-way partitioning solutions for all contiguous subsets of the ordering. Dynamic programming is then applied to find all the optimal 2-way solutions, then all the optimal 3-way solutions, etc. DP-RP is optimal for any partitioning objective that is a *monotone* function of some intercluster cost metric. Such objectives include standard k-Way Minimum Cut, Scaled Cost, Absorption, and the DS metric. Although the complexity of DP-RP depends on the objective function, $O(nU + kn(U-L))$ implementations exist for all of these objectives except DS, with L and U denoting lower and upper bounds on cluster size.

The DP-RP result highlights the problem of finding 1-dimensional netlist representations which will lead to good multi-way partitioning solutions. Alpert and Yao [6] have applied DP-RP to their (MELO) orderings (see also the discussions of [4] [100] [101] in Section 6). Note that Frankle and Karp's [68] probe technique can be used to yield linear orderings as well, simply by sorting the entries of $V_d\vec{y}$ for a given probe direction $\vec{y}$; we believe that applying DP-RP to these orderings is a promising direction for future research.

4.5.3 Placement-Based Approaches

Another method which constructs a bipartition from a linear ordering is due to Riess et al. [156]. The premise of their PARABOLI approach is that the *squared* wirelength objective is not as useful as a *linear* wirelength objective for 1-dimensional placement; this insight derives from experiments with the GORDIAN placement package [120]. The authors of [156] begin with the 1-dimensional placement induced by $\vec{\mu_2}$; the ten modules with largest coordinates are fixed at location 1.0, and the ten modules with smallest coordinates are fixed at location 0.0. The remaining "free" modules are constrained to have center of gravity (i.e., mean coordinate) at 0.5; quadratic programming techniques are used to reposition these modules to minimize linear wirelength (see Section 5.3.4). The 5% of modules with largest coordinates are assigned center of gravity 0.95, and the 5% with smallest coordinates are assigned center of gravity 0.05. The free modules are again replaced, and the next 5% at the extreme right (left) are assigned center of gravity 0.90 (0.10). Note that throughout this process, only the original 20 extreme modules are fixed, with all others free to move but restricted by their centers of gravity. This process is repeated ten times (e.g., the next iteration uses centers of gravity 0.85 and 0.15), inducing ten distinct linear orderings. Riess et al. choose the best ratio cut bipartitioning among all possible splits of all ten orderings. The intuition behind PARABOLI is that after the first iteration, the leftmost and rightmost modules clearly belong at opposite ends of the linear ordering, but the proper locations for the middle modules remains unclear. Iteratively constraining only small fractions of the modules to the extremes of the linear ordering makes placement of the inner modules easier, while retaining flexibility. Riess and Schoene [158] have extended this approach to a layout-driven formulation in MCM partitioning, optimizing a given multi-chip layout alternately in vertical and horizontal directions.

We believe the success of the PARABOLI approach may have just as much to do with its module assignment technique as its linear wirelength objective. Thus, an interesting experiment would evaluate the quality of linear orderings derived using the same methodology, but with a quadratic objective. The successive over-relaxation placement approach of Tsay and Kuh [183] can also be used for this type of quadratic optimization with fixed module locations (however, centers of gravity cannot be used); see Section 5.3.4.

5 Combinatorial Formulations

The engineering workstations in a typical modern design environment have processing capabilities exceeding those of mainframes from only a few years ago, thereby permitting previously infeasible approaches to be applied to CAD optimizations. For example, spectral computations proposed in the early 1970's have become viable for netlist partitioning, and the quadratic programming iteration behind GORDIAN-type cell placement programs [120] now drives many of the latest industry placement packages. Many of these "rediscovered" combinatorial optimizations are well-studied and have either polynomial-time optimal solutions (e.g., network flow) or highly-developed heuristic tools (e.g., *Espresso II* for set covering). In addition, combinatorial approaches can capture complex formulations that incorporate timing constraints, preassignment of modules to clusters, or multiple cost functions (e.g., the authors of [174] force all paths to satisfy delay constraints within a quadratic boolean program). Quite possibly, the next frontier of optimization strategies for CAD applications will involve large-scale mathematical programming instances, including mixed integer-linear programs that require branch-and-bound search.[11] This trend toward combinatorial algorithmic approaches is typified by the works that we now discuss.

The approaches in this section share the common theme of the *transformation* of a partitioning problem into a different combinatorial formulation. Each formulation is of independent interest and typically has a long history and a large literature outside the scope of VLSI partitioning. Our discussion begins with graph labeling techniques for solving the Min-Delay Clustering problem. We then show how many formulations can be addressed in the context of network flows. Next, we discuss mathematical programming formulations such as quadratic and linear programming. These approaches have also been applied to one-dimensional placement; we discuss such works along with their potential applications to partitioning. Finally, we review fuzzy partitioning techniques and a set-covering approach to FPGA partitioning.

5.1 Min-Delay Clustering by Graph Labeling

Lawler et al. [130] first considered the Min-Delay Clustering problem (see Section 2.4) for combinational Boolean networks (DAGs) with no module delay, i.e., $\delta(v_i) = 0 \ \forall \ v_i \in V$. We call this the *unit-delay* clustering problem. This formulation assumes that module and intracluster delays (i.e., delays between modules in the same cluster) will be negligible compared to intercluster delay that results from placing clusters onto different chips. Lawler et al. propose the following labeling scheme to derive a clustering in which all clusters C must satisfy $w(C) \leq U$.

Each module v_i is associated with a label $\pi(i)$, i.e., $\pi : [1..n] \rightarrow [0..n]$. Module v_i is a *predecessor* of v_j if there is a directed path from v_i to v_j, and v_i is an *r-predecessor* of v_j if in addition $\pi(i) = r$. The set of all r-predecessors of v_j is denoted by $p_j(r)$; the total weight of modules in this set is $w(p_j(r))$.

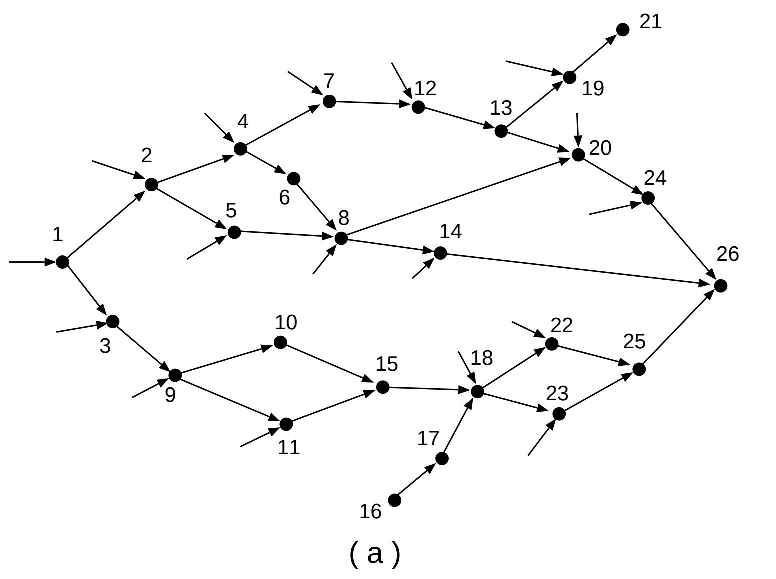
[11] A primal-dual approach with column generation has already been used to solve the linear programming relaxation of large integer programs in the context of global routing [36]; however, only small integer programs have been essayed in CAD, typically for high-level synthesis (see, e.g., [71]).

- For all primary inputs v_i (no incoming edges), set $\pi(i) = 0$.
- Find any unlabeled v_i, all of whose predecessors are labeled, and let r be the largest label of all predecessors of v_i. If $w(v_i) + w(p_i(r)) \leq U$, then set $\pi(i) = r$, otherwise set $\pi(i) = r + 1$.

The second step is repeated until all modules are labeled; two modules v_i and v_j with the same label $\pi(i)$ are placed into the same cluster if every module on the path from v_i to v_j also has label $\pi(i)$. If the netlist is a tree with a single primary output (a minor modification handles a single primary input), then this labeling scheme optimally solves the unit-delay clustering problem. If the netlist is a tree with multiple primary inputs and outputs, then the tree can be separated into a set of rooted subtrees, i.e., each subtree has a single primary input or primary output. The labeling scheme and clustering algorithm can applied to each subtree, and merging the resulting clusterings gives an optimal unit-delay clustering solution.

If the netlist is not a tree, this labeling scheme can still derive the optimal solution if module replication is permitted. For this case, Lawler et al. construct a cluster $C = \{v_i\} \cup p_i(\pi(i))$ for each v_i such that all of v_i's successors (modules v_j such that there is v_i-v_j path) have higher labels than v_i. This solution may assign some modules to more than one cluster, i.e., replicating these modules. In Figure 12, we reproduce (a) an example from [130], and (b) their min-delay clustering solution (with replication) assuming unit module weights and $U = 5$. The labeling scheme assigns label 0 to the modules in the 6 leftmost clusters, label 1 to the modules in the 4 middle clusters, and label 2 to module 26 in the rightmost cluster. The modules with successors that all have higher labels are $5, 6, 10, 11, 12, 14, 17, 21, 24, 25$ and 26. As an example of replication, note that the labeling scheme assigns module 8 to two clusters, since it is a 1-predecessor of modules 14 and 24; hence, the edges from 5 to 8 and from 6 to 8 are also replicated.

Murgai et al. [140] were the first to address the *variable-delay* clustering problem (with module delays $\delta(v_i) \neq 0$). Their generalization of the above labeling scheme retained optimality only under certain conditions. Rajaraman and Wong [151] later solved this problem optimally using a different labeling scheme. Consider, for a given v_i, the subgraph $N(i)$ induced by v_i and all of its predecessors. The delay from any $v_j \in PI$ to v_i in a given clustering solution is at least the delay from v_j to v_i in the optimal clustering of $N(i)$. This observation allows [151] to iteratively build optimal clusterings over $N(i)$ for each $v_i \in V$. Each $v_i \in PI$ is initially assigned the label $\pi(i) = \delta(i)$, each $v_i \notin PI$ will have a label no greater than the maximum delay of any path from a primary input to v_i. As in [130], any module whose predecessors are all labeled may be labeled next. When labeling v_i, a set $cluster(i) \subseteq N(i)$ is constructed which essentially consists of the predecessors of v_i that have the highest "label plus delay factor" values. Rajaraman and Wong show that there exists an optimal partitioning of $N(i)$ with $cluster(i)$ as a cluster. The label for v_i is the maximum delay over all paths



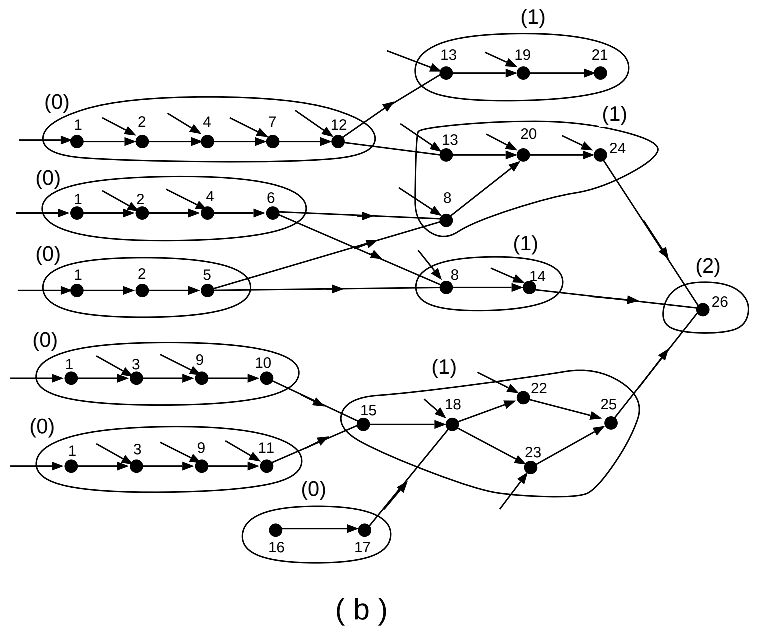


Figure 12: (a) An example DAG from [130] and (b) the clustering solution from Lawler et al.'s [130] labeling algorithm assuming unit module weight and cluster size bound $U = 5$.

from $v_j \in PI$ to v_i assuming that $cluster(i)$ exists, i.e., a delay of 1 is added to any path containing an edge from a module in $N(i) - cluster(i)$ to a module in $cluster(i)$. The partitioning solution P^k will be a subset of $\{cluster(i) \mid v_i \in V\}$. Initially, $P^k = \emptyset$ and $cluster(i)$ is added to P^k for every $v_i \in PO$. Next, the following rule is iteratively applied: for every v_i not contained in any cluster of P^k, such that there exists some v_j with $(v_i, v_j) \in E$ and $v_j \in C_h$ for some $C_h \in P^k$, add $cluster(i)$ to P^k. Like [130], this construction will also replicate modules.

A variant of the min-delay clustering problem was studied by Cong and Ding [45] for technology mapping in lookup table-based FPGA designs. They remove the area constraints $w(C_h) \leq U, 1 \leq h \leq k$ and replace them with the pin constraints $|In(C_h)| \leq K$ and $|Out(C_h)| = 1$, where $In(C_h) = \{v_j \in C_h \mid \exists v_i \notin C_h, (v_i, v_j) \in E\}$ and $Out(C_h) = \{v_i \in C_h \mid \exists v_j \notin C_h, (v_i, v_j) \in E\}$. These constraints allow each cluster to be mapped to a configurable logic block, i.e., a K-input lookup table that can implement any Boolean function of K variables. To address this formulation, Cong and Ding proposed a labeling and flow-based heuristic. Subsequently, Yang and Wong [191] proved that the same approach is not only optimal for these pin constraints, but also optimal for the more general pin constraints $|In(C_h) \cup Out(C_h)| \leq K+1$ under the unit-delay model. The authors of [191] combine the approaches of [45] and [151] into a heuristic for min-delay clustering under size and pin constraints $w(C_h) \leq U$, $|E(C_h)| \leq K+1$. The algorithm achieves optimal delay under either the pin constraint alone (as does [45]) or the size constraint alone (as does [151]).

5.2 Network Flows

New formulations such as replication and the increased usage of a directed netlist representation have recently made flow-based approaches more popular. We now explore a variety of partitioning formulations, and describe how network flows can be used to solve them (see [43] for many other examples).

5.2.1 Preliminaries

We first review several concepts from the theory of network flows [66]. A *flow network* $G = (V, E)$ is a directed graph in which each edge $(v, w) \in E$ has a positive *capacity* $c(v, w) > 0$; edges that are not present in the network implicitly have capacity zero. There are two distinguished nodes in G, a *source* $s \in V$ and a *sink* $t \in V$. A *flow* in G is a real-valued function $f : E \rightarrow \Re^+$ that satisfies the following properties:

1. **Capacity constraints:** For all $(v, w) \in E$, $f(v, w) \leq c(v, w)$; e is said to be *saturated* if $f(v, w) = c(v, w)$.
2. **Skew Symmetry:** For all $v, w \in V$, $f(v, w) = -f(w, v)$.
3. **Flow conservation:** For all $v \in V - \{s, t\}$, $\sum_u f((u, v)) = \sum_w f((v, w))$.

The *value* of a flow f is given by $|f| = \sum_{v \in V} f((s, v))$. For a given network, the network flow problem is to find a flow of maximum value. An *s-t cut* is a bipartitioning $\{C_1, C_2\}$ of G with $s \in C_1, t \in C_2$, and cutsize is given by $F(\{C_1, C_2\}) = \sum_{u \in C_1, v \in C_2} c(u, v)$. A well-known result from linear programming duality is the *max-flow min-cut* theorem [66]:

Max-Flow Min-Cut Theorem: Given a flow network G, the value of the maximum s-t flow is equal to minimum cost of any s-t cut. Moreover, all edges (v, w) with $v \in C_1, w \in C_2$ are saturated for some minimum s-t $\{C_1, C_2\}$.

The *min-cost flow* problem associates a cost (denoted $cost(v, w)$) with each $(v, w) \in E$, and the problem becomes that of finding a flow f with a prescribed value $|f| = z$ such that the total flow cost $\sum_{(v,w)} f(v, w) \cdot cost(v, w)$ is minimized.

Another formulation is the *uniform multicommodity flow* (UMCF) problem, which has been used to bipartition undirected edge-weighted graphs [138] [133]. For every $v, w \in V$, a special commodity $f_{v,w}$ is assigned, and we require exactly z units of this commodity to flow from v to w. The objective is to find a feasible flow with maximum z (s and t are unspecified). The total flow of all commodities along any given edge cannot exceed the capacity of the edge. Given a solution to the UMCF problem and a bipartitioning $\{C_1, C_2\}$, the flow from C_1 to C_2 is given by $\sum_{v \in C_1, w \in C_2} z = z \cdot |C_1| \cdot |C_2|$. The *free capacity* of $\{C_1, C_2\}$ (i.e., the total unused capacities on edges from C_1 to C_2) is the cutsize minus the flow from C_1 to C_2. Using the fact that the free capacity is nonnegative, Matula and Shahrokhi [138] establish the lower bound

$$z \leq \min_{\{C_1, C_2\}} \frac{F(\{C_1, C_2\})}{|C_1| \cdot |C_2|}$$

i.e., the the minimum ratio cut of G is an upper bound for z. Leighton and Rao [132] showed this bound is relatively tight by constructing a bipartitioning with ratio cut cost of $O(z \log n)$, where z is the optimal flow for this UMCF formulation. An approximation algorithm for multi-way partitioning [131] applies Leighton and Rao's algorithm recursively. A review of recent advances in multicommodity flows is given in [14].

5.2.2 The Min-Cut Replication Problem

Assume that we are given a directed graph $G(V, E)$ and a k-way partitioning $P^k = \{C_1, C_2, \ldots, C_k\}$ without replication, and let F be the multi-way min-cut objective. The *Min-Cut Replication Problem* of Hwang and El Gamal [103] seeks a collection of subsets of modules $\{C^*_{ij} \mid C^*_{ij} \subseteq C_i, 1 \leq i, j \leq k\}$ that minimizes $F(P^{k*})$, where P^{k*} is the partitioning that results when each subset C^*_{ij} is replicated from C_i to C_j. (Hwang and El Gamal implicitly assume each C_i contains a subset I_i of primary inputs that cannot be replicated.) We now show how to solve this formulation optimally for $k = 2$ [103]; the problem can be solved optimally for $k > 2$ by solving k independent 2-way replication instances [104].

Given a bipartitioning $P^2 = \{C_1, C_2\}$, we seek a subset $C^* = C^*_{12} \subseteq C_1 - I_1$ such that the number of edges from modules in $C_1 - I_1 - C^*$ to modules in $C^* \cup C_2$ is minimized. The new bipartitioning P^{2*} contains clusters C_1 and $C_2 \cup C^*$, so that edges from C^* to C_2 will not be cut (see Section 2.6 for replication rules). Edges from C_2 to C_1 and from I_1 to C_2 will still be cut in the new solution, and can be ignored; hence, only edges from $C_1 - I_1 - C^*$ to $C^* \cup C_2$ need to be considered. Hwang and El Gamal construct the following flow network $G'(V', E')$ with $V' = \{s, t\} \cup V$ and $E' = (E - E_{21}) \cup E'_s \cup E'_t$ where

- $E'_s = \{(s, v) \mid v \in I_1\}$.
- $E'_t = \{(v, t) \mid v \in C_2 \text{ and } \exists\ w \in C_1, (v, w) \in E\}$.
- $E_{21} = \{(v, w) \mid v \in C_2$.

The edge capacities are $c(e) = \infty\ \forall e \in E'_s \cup E'_t$, and $c(e) = 1\ \forall e \in E - E_{21}$; this ensures that only edges in $E - E_{21}$ will be part of the min-cut. Applying the Max-Flow Min-Cut theorem to G' yields the optimum min-cut bipartitioning $P'^2 = \{C'_1, C'_2\}$ with $\{s\} \cup I_1 \subseteq C'_1$ and $t \in C'_2$. Since no edges in E'_t are cut, $C_2 \subseteq C'_2$, and $C^* = C'_2 - C_2$ is the optimum set of modules to replicate from C_1. Optimality follows directly from the Max-Flow Min-Cut Theorem.

In [104], Hwang and El Gamal showed how to modify their flow network to solve min-cut replication for hypergraphs with signal information. While their formulation requires an input partitioning solution, it could be applied during the construction of a multi-way partitioning, and seems useful for pin-constrained (FPGA) formulations. For example, one might apply the algorithm to the bipartitioning $\{V - C, C\}$ when a cluster C is close to satisfying the pin constraint $|E(C)| \le IO$. As a heuristic, [104] integrated their min-cut replication solution with a modified form of FM. Yang and Wong [192] slightly enhanced Hwang and El Gamal's approach to solve the min-cut replication problem while minimizing $|C^*|$ as a secondary objective, breaking ties so that a min-cut with "minimum size" is found. Recently, Liu et al. [135] appear to have eliminated the need for a pre-existing bipartitioning by solving the *general replication* problem: find clusters C_1, C_2, and $C^* \subseteq C_1$ such that $P^2 = \{C_1 - C^*, C_2 \cup C^*\}$ minimizes the number of cuts with replication.

5.2.3 Max-Flow Min-Cut Approaches

The approaches in this section are linked by their use of the Max-Flow Min-Cut Theorem in finding one or more minimum $s - t$ cuts to construct a partitioning solution.

To find the global min-cut bipartitioning, $n - 1$ max-flow computations suffice via the Gomory-Hu cut tree construction [77]. Such a tree is formed by finding an $s - t$ cut $\{C_1, C_2\}$, and constructing the edge (C_1, C_2) between vertices C_1 and C_2 in the cut tree, with cost $|E(C_1)|$. The tree is expanded recursively by picking any two modules s and t in a cluster C and computing the minimum $s-t$ cut $\{C', C''\}$ in which all modules in $V-C$ are condensed into a single node. If this node is in C' then the vertex C in the tree is replaced with C' and the edge (C', C'') is added with cost $E(C')$. This process continues until every node in the tree corresponds to a singleton cluster, and the global min-cut is determined by the edge in the tree with lowest cost.

Hao and Orlin [92] also showed how to find a min-cut bipartitioning using $O(n)$ flow computations, with total time equivalent to to that of just one flow computation. They compute a sequence of $S-t'$ cuts where S is a set condensed into a single node and $S=\{s\}$ initially. After an $S-t'$ cut is computed, t' is added to S, a new t' is chosen and the process repeated until $S=V$. The lowest-cost cut observed in this sequence gives the optimal min-cut bipartitioning.

Hu and Moerder [99] showed how the Max-Flow Min-Cut duality can also be used to find a Min-Cut Bipartitioning in a hypergraph. For each net e, an extra *dummy module* is added to the graph, and a star is formed connecting the dummy module to each of the modules in e (see Figure 13(a)). The dummy module has unit capacity, and the other modules have infinite capacity. Hence, any minimum vertex separator (cutset of modules) of this network will be a subset of the dummy modules which will correspond to a subset of E. The minimum vertex separator can be found by maximum-flow after transforming this *node-capacitated* flow network into an *edge-capacitated* network $G'(V', E')$ using a technique due to Lawler [129]:

- For every $v \in V$, add v_1 and v_2 to V' and add (v_1, v_2) to E'. Assign $c(v_1, v_2)$ to be the node capacity of v.
- For every $(v, w) \in E$, add the edge (v_2, w_1) to E'.

If Hu and Moerder's star representation in Figure 13(a) is transformed into a directed graph by replacing each undirected edge (v, w) with directed edges (v, w) and (w, v) and applying the above construction, the graph in Figure 13(b) is the result. However, applying the first rule of Lawler's transformation to modules with infinite capacity is unnecessary. Hence, v_1 and v_2, u_1 and u_2, and w_1 and w_2 can be collapsed into u, v, and w, respectively, yielding the flow network in Figure 13(c). Hu and Moerder also show how to construct a modified Gomory-Hu cut tree to solve min-cut hypergraph bipartitioning with $n-1$ flow computations. Vannelli and Hadley [184] also discuss hypergraph bipartitioning based on Gomory-Hu cut trees.

Saran and Vazarani [166] use Gomory-Hu cut trees as the basis for a k-way partitioning heuristic for weighted undirected graphs with no cluster size constraints. They sort the cuts $g_1, g_2, \ldots, g_{n-1}$ on the Gomory-Hu cut tree by increasing weight, and find the minimum i such that $g_1 \cup g_2 \cup \ldots \cup g_i$ divides the graph into k components. This solution is guaranteed to yield a cutset within a factor of $2 - \frac{2}{k}$ of optimal and uses $n-1$ flow computations. The authors of [166] also proposed an alternative which hierarchically splits the cluster which has the smallest min-cut; hence, a Gomory-Hu cut must be computed for each cluster. This approach also achieves a performance ratio of $2 - \frac{2}{k}$ but requires $O(kn)$ flow computations. Neither heuristic uniformly dominates the other.

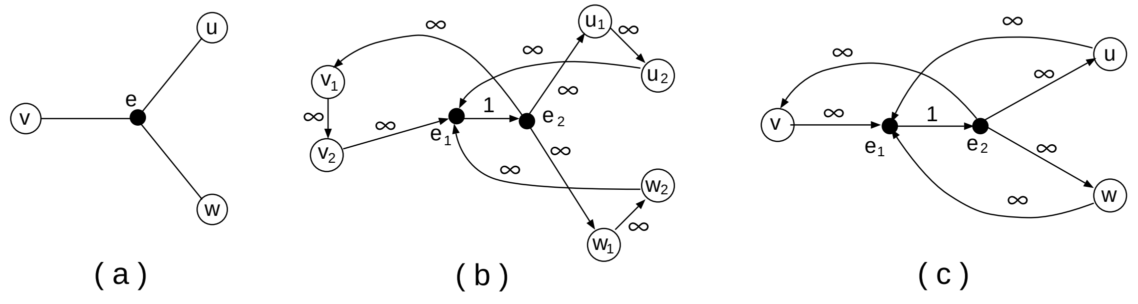


Figure 13: (a) The star representation for a 3-pin net, (b) the application of Lawler's transformation to (a), and (c) a more efficient representation of (b).

While a max-flow computation is guaranteed to return a minimum cut, the resulting cluster sizes may be very unbalanced. Consequently, many works propose contracting large subsets of modules to enforce balance constraints, e.g., Bui et al. [29] propose bisecting d-regular graphs by contracting modules in the neighborhoods of s and t before computing an $s-t$ cut. The *neighborhood* of a module v in a d-regular graph with n modules is defined to be the set of all modules within graph distance $\log_{(d-1)}(\sqrt{n} - 2)$ of v. Bui et al. compute a minimum $s-t$ cut in the contracted graph for every possible choice of s and t, then combine the cuts to form a bisection. In bipartitioning a DAG subject to cluster size constraints, Iman et al. [107] similarly use flow computations to find a minimum $s-t$ cut and then, if size constraints are violated, contract the smaller cluster C into a single module with weight $w(C)$. To force a different min-cut in the next iteration, the capacity of each edge in the min-cut is increased by a factor of $1 + \epsilon$ in the flow network. Yang and Wong [190] apply a similar scheme, along with the flow construction in Figure 13(c), for hypergraph bipartitioning with size constraints. However, instead of increasing capacities of edges in the cut, they pick an additional module v from the larger cluster and contract it with the smaller cluster C. This approach may be more likely to find the true min-cut (since it does not modify edge weights), and it also bounds the number of flow computations. Furthermore, the approach takes only $O(nm)$ time, i.e., the time of a single max-flow computation – since the max-flow can be computed in the new contracted network using the residue network (the network of remaining flow capacities) from the previous flow computation.

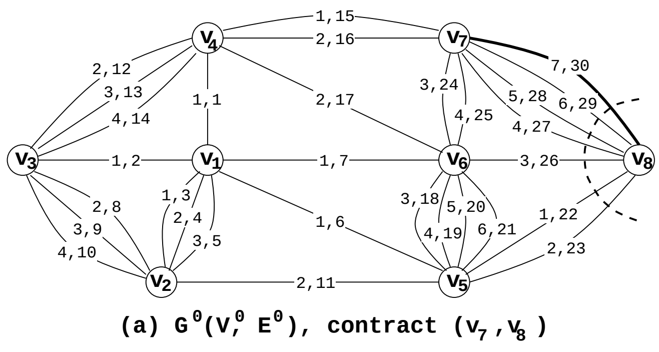
Alternative Optimal Min-Cut Methods

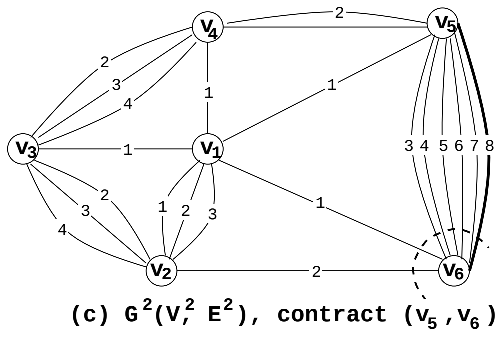
Nagamochi and Ibaraki [141] recently gave a very efficient algorithm which finds the optimal min-cut bipartitioning without using any flow computation. They assume an undirected, unweighted graph $G(V, E)$ as input, although multiple edges may exist between modules. The modules and edges are labeled using a maximum-adjacency scheme, i.e., the module with the most edges incident to the set of labeled vertices is ordered next. Initially, labels $\pi(v) = 0$ are assigned to each $v \in V$. The algorithm iteratively chooses the module v with the largest label (ties are broken arbitrarily), and then visits each unvisited edge (v, w), incrementing $\pi(w)$ by one and then setting the label of (v, w) to $\pi(w)$. The scheme requires $O(|E|)$ time to label all modules and edges.

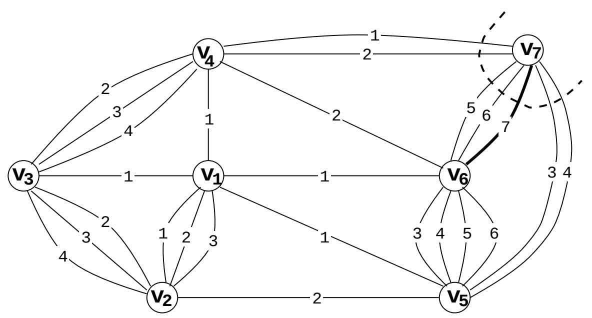
Let $E(i)$ be the set of all edges with label i. Interestingly, the edges in $E(i)$ form a forest that spans $G(V, E - E(1) - E(2) - \ldots - E(i-1))$. The algorithm begins with $G^0 = G$ and computes G^{r+1} from G^r as follows: first label the modules and edges of G^r using the above scheme, then pick a module v with $deg(v) = deg_{min}$ such that v is incident to an edge (v, w) with label deg_{min}, and finally contract v with this w. The resulting graph with self-loops removed is G^{r+1}, and the edges incident to v in G^r form the cutset. The smallest cutset observed during the contractions from G^0 down to G^{n-1} is a global min-cut for G. The authors of [141] also extended this approach to graphs with real-valued edge weights.

As an example, consider the multi-graph in Figure 14 (a). The modules are numbered according to a possible labeling order, and each edge e is marked with (i, j) where i is the label of e and j indicates the order in which the edges were visited. Many modules v have $deg(v) = deg_{min} = 7$, although only v_7 and v_8 are incident to an edge with label 7. Hence, v_7 and v_8 are contracted to yield the multi-graph in (b). The labeling is computed for this new graph (the order in which edges are labeled is not shown), and the contraction is repeated, yielding the graphs in (c) and (d). The graphs $G^4 - G^6$ are not shown. The smallest cutset observed during this process is of size 6 in (d); this corresponds to the optimal min-cut bipartitioning $\{\{v_1, v_2, v_3, v_4\}, \{v_5, v_6, v_7, v_8\}\}$.

A randomized min-cut approach was proposed by Karger [114]: iteratively contract a random pair of incident modules into a cluster until only two clusters remain. Karger showed that $O(n^2)$ executions of this algorithm will not only find an optimal bipartitioning with high probability, but will also find all optimal min-cut bipartitionings with high probability.


(a) $G^0(V^0, E^0)$, contract (v_7, v_8)


(c) $G^2(V^2, E^2)$, contract (v_5, v_6)


(b) $G^1(V^1, E^1)$, contract (v_6, v_7)

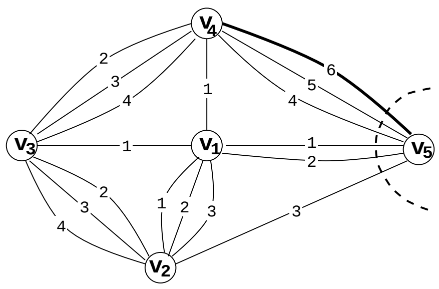

(d) $G^3(V^3, E^3)$, contract (v_4, v_5)

Figure 14: Example of the execution of the min-cut algorithm of [141].

5.2.4 Bipartite Flow

A *bipartite graph* $G'(V', E')$ has the property that V' can be partitioned into V_1 and V_2 such that every edge $(u, v) \in E'$ has $u \in V_1$ and $v \in V_2$. A *bipartite flow network* is a bipartite graph with additional source s and sink t added to V', and all edges in E' directed from nodes in V_1 to nodes in V_2. In addition, directed edges (s, v) $\forall v \in V_1$ and (w, t) $\forall w \in V_2$ are added to the flow network. We now discuss various partitioning formulations that can be solved with bipartite graphs and/or flow networks.

Huang and Kahng [101] use bipartite flow (specifically, the "provisioning" formulation from Lawler [129]) to find the maximum-density cluster (i.e., subhypergraph) C of a hypergraph $H(V, E)$, where $den(C) = \frac{|\{e \in E \mid e \subseteq C\}|}{w(C)}$ (see Section 2.4). The decision question "Does there exist a cluster with density larger than d?" can be answered using the bipartite graph $G'(V', E')$ with $V_1 = E$ and $V_2 = V$ and $E' = \{(e, v) \mid v \in e, v \in V, e \in E\}$.[12] A bipartite flow network is constructed by setting capacities $c(e, v) = \infty$ $\forall (e, v) \in E'$, $c(s, e) = 1$ $\forall e \in E$, and $c(v, t) = d \cdot w(v)$ $\forall v \in V$. The maximum $s - t$ flow in $G'(V', E')$ yields a min-cut bipartitioning $P^2 = \{C_1, C_2\}$ with $s \in C_1, t \in C_2$. The cluster $C = C_1 \cap V$

[12] This bipartite graph representation of a hypergraph is very natural, but has seen surprisingly little use in the CAD literature.

is guaranteed to have density at least d if such a cluster exists. If no cluster has density $\geq d$, then the computation will return $C = \emptyset$ (actually $C_1 = \{s\}$).

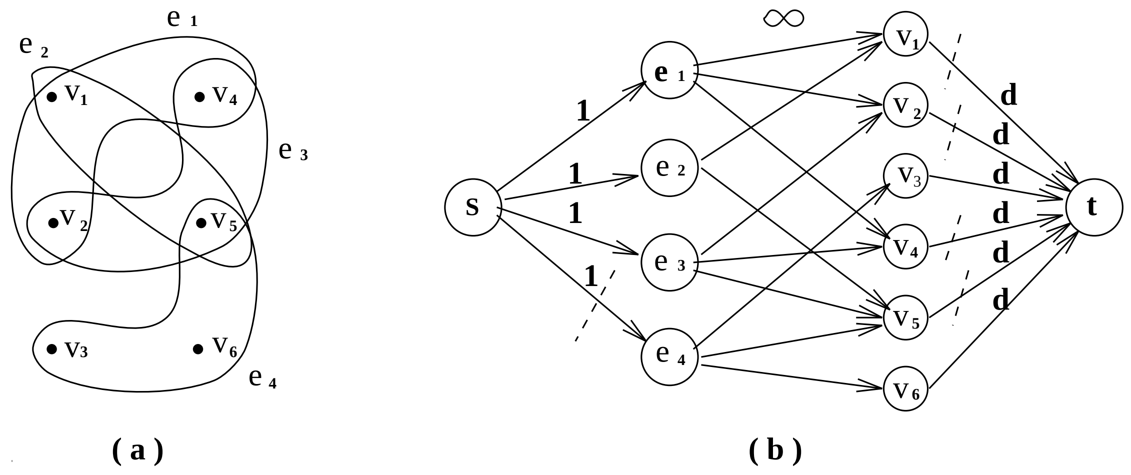


Figure 15: (a) An example of a hypergraph and (b) its density flow network assuming unit module weights.

Consider the hypergraph in Figure 15(a) with 6 modules and 4 nets; its density flow network assuming unit module weights is shown in (b). When $d = \frac{3}{4}$, the min-cut bipartitioning for this network is $P^2 = \{\{s, e_1, e_2, e_3, v_1, v_2, v_4, v_5\}, \{e_4, v_3, v_6, t\}\}$ with corresponding max-flow value $|f| = 4d + 1 = 4$. Hence, $C = \{v_1, v_2, v_4, v_5\}$, and the density of C is guaranteed to be at least $\frac{3}{4}$. Since the density decision question can be answered efficiently, binary search on values of d can yield the max-density cluster using $O(\log n)$ flow computations. As a heuristic for finding a k-way partitioning which maximizes the sum of cluster densities, Huang and Kahng propose iteratively finding the max-density cluster and removing it from the hypergraph. A post-processing step of constructs a linear ordering from the sequence of max-density clusters; DP-RP (see Section 4.5) is then applied.

Another problem that can be solved using bipartite flow is the *completion* (see Section 4.5) of a net bipartitioning $\{N_1, N_2\}$ into a module bipartitioning $\{C_1, C_2\}$ such that if all the nets incident to a given module belong to the same cluster N_i, then the module must be assigned to C_i. The problem may be viewed as assigning modules corresponding to edges in the bipartite "conflict subgraph", i.e., modules with incident nets in both N_1 and N_2, to C_1 and C_2 to minimize the number of cut nets. This bipartite graph $G'(V', E')$ has $V_1 = N_1$, $V_2 = N_2$ and $(e, e') \in E'$ if $e \in N_1, e' \in N_2$ and $e \cap e' \neq \emptyset$. Each edge in E' corresponds to the set of modules $e \cap e'$. Every net $e \in N_i$ will either have all of its modules in C_i (i.e., it will be uncut) or have some modules in both clusters (i.e., it will be cut). A *maximum independent set* (MIS) $S \subseteq V'$ for $G'(V', E')$ has the property that if $e \in S$ and $e' \in S$ then $(e, e') \notin E'$ and $|S|$ is maximal with respect to this property. Cong et al. [48] construct an MIS S using the algorithm of [93] and assign each contested module v to C_i if there exists an $e \in S \cap N(v)$

with $e \in N_i$. As an example, consider the bipartitioning of the intersection graph in Figure 16(a) with $N_1 = \{e_1, e_2, e_3, e_4\}$ and $N_2 = \{e_5, e_6, e_7, e_8\}$ and (b) its corresponding bipartite graph. The unique MIS for this graph is $\{e_1, e_2, e_3, e_4, e_7, e_8\}$ so the modules incident to these nets can all be assigned to the same cluster, and at most nets e_5 and e_6 will be cut. (Note that this approach is not optimal: vertices of the intersection graph not in the MIS and may yet end up being uncut, i.e., maximizing the size of the independent set of uncut hyperedges does not correspond to minimizing the number of cut hyperedges.)

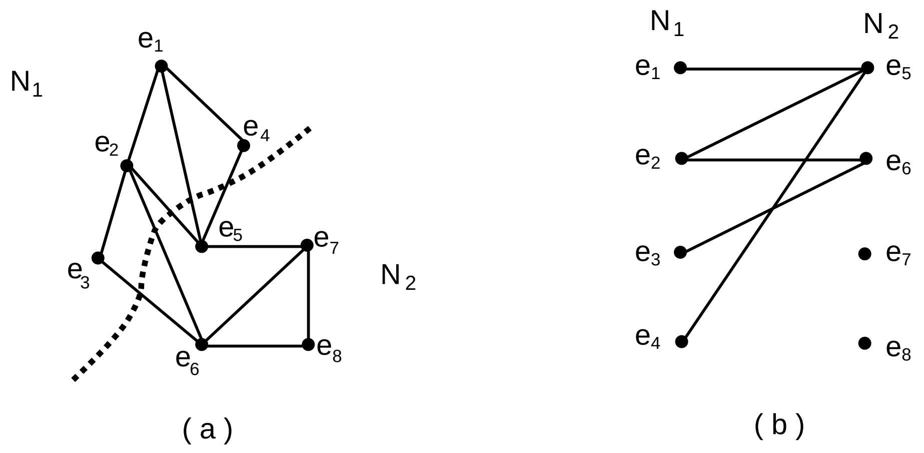


Figure 16: (a) An example of a net bipartitioning of the intersection graph and (b) the bipartite conflict graph it induces.

Kamidoi et al. [113] earlier proposed a similar technique for hypergraph bisection. They first transform the netlist into a graph $G(V, E)$ using the star net model of [99]. They then divide the nodes of this graph into module nodes V_M and net nodes V_N. Breadth-first search is used to find an *articulation set* (i.e., a vertex separator) $S \subseteq V_M$ that separates V into two clusters C_1 and C_2. S forms the set of contested modules and induces the bipartite graph with vertices $V_1 = C_1 \cap V_N$, $V_2 = C_2 \cap V_N$ and edges $(e, e') \in E$ if $e \cap e' \cap S \neq \emptyset$. Kamidoi et al. then resolve module contentions by finding an independent set of net nodes in this graph, using a greedy approach that integrates cluster sizes.

Finally, the *multi-way contention problem* seeks to convert a k-way net partitioning $\{N_1, N_2, \ldots, N_k\}$ into a k-way module partitioning $\{C_1, C_2, \ldots, C_k\}$. A module is in *contention* if it is incident to nets that are not all in the same N_i; uncontested modules with all nets in N_i become assigned to C_i. Cong et al. [49] construct a complete bipartite graph with V_1 being the set of contested modules and $V_2 = \{N_1, N_2, \ldots, N_k\}$ with an edge (v, N_i) from every $v \in V_1$ to every cluster of nets. Edges (s, v) with cost zero and capacity one are introduced $\forall v \in V_1$, and edges (N_i, t) with cost zero and capacity equal to the remaining size capacity of C_i are introduced $\forall N_i$. Finally, each edge (v, N_i) has capacity one and cost equal to $MAX - pref(v, N_i)$, where MAX is a constant larger than any *preference* value, and the preference of v for N_i is given by

$$pref(v, N_i) = \sum_{\{e \in N_i \ | \ v \in e\}} \frac{(|e| - Con(e))^2}{|e| \cdot Con(e)}. \tag{5.1}$$

Here $Con(e)$ is the number of modules of e that are in contention, i.e.,

$$Con(e) = |\{v \in e \text{ s.t. } \exists e', v \in e', \text{ with } e \in N_i, e' \in N_j, \text{ for some indices } i \neq j\}|$$

A min-cost flow of value $|V_1|$ will saturate n edge-disjoint paths from s to vertices N_i, which implies that there is exactly one saturated edge of the form (v, N_i) for each $v \in V_1$. For each such edge, v is assigned to cluster C_i. This algorithm guarantees the optimal module assignment in terms of maximum sum of preferences.

5.2.5 Shortest Path Clustering and Probabilistic Methods

Yeh et al. [193] proposed an algorithm based on the relationship between uniform multi-commodity flow and min-cut partitioning. Yeh et al. construct a "flow network" by assigning $c(e) = cost(e) = 1$ and $f(e) = 0$ for every net e. Two random modules in the network are chosen and the shortest path (i.e., path with lowest cost) p between them is computed where $cost(p) = \sum_{e \in p} cost(e)$. A constant $\Delta < 1$ is added to $f(e)$ for each net $e \in p$, and the cost for every $e \in p$ is reset to $cost(e) = e^{\alpha f(e)}$ for some constant α. (Yeh et al. choose $\Delta = 0.1$ and $\alpha = 10$.) Adjusting the cost penalizes paths through congested areas and forces alternative shortest paths. This random shortest path computation is repeated until all paths between the chosen pair of modules pass through at least one "saturated" net (i.e., with $f(e) = 1$).

The set of saturated nets induces a multi-way partitioning in which v and w belong to the same cluster if there is a path of unsaturated nets between v and w. For each of these clusters C, the *flux* $\frac{|E(C)|}{w(C)}$ is computed and the clusters are sorted based on their flux value. Yeh et al. begin initially with $P_1 = \{V\}$ and consider in turn each cluster from the sorted list to "peel" from V. For example, if C_1 is the first cluster in the list, the bipartitioning $P^2 = \{C_1, V - C_1\}$ results. Assume that the current solution is $P^k = \{C_1, C_2, \ldots, C_{k-1}, V - C_1 - C_2 - \ldots - C_{k-1}\}$, and the unseen cluster with highest flux is C^*. If $P^{k+1} = \{C_1, C_2, \ldots, C_{k-1}, C^*, V - C_1 - C_2 - \ldots - C_{k-1} - C^*\}$ has a smaller Cluster Ratio (see Section 2.4) than P^k, then P^{k+1} replaces P^k as the current solution, otherwise the next unseen cluster on the list is considered. This process repeats until all of the clusters in the sorted list have been seen. The SPC approach is attractive because the saturated nets are good candidates to be cut in a partitioning solution. However, SPC may tend to construct poorly balanced solutions since the clusters in the sorted list might all be very small. The basic principles behind this approach were adopted by Hauck and Borriello [95] for FPGA partitioning onto an underlying topology (cf. the discussion of layout-driven formulations in Section 2.4).

An alternative taxonomy might classify the SPC approach along with the random-walk clustering algorithm of Hagen and Kahng [82] and the compaction algorithm of Karger [114]. All of these approaches rely on randomization to uncover the circuit structure. For example, the nets between dense clusters are likely to be in many shortest paths, hence with high probability these nets will become saturated during the execution of the SPC algorithm. Similarly, a random walk that visits a dense cluster will likely not leave the cluster until many of the cluster's modules have been visited. Finally, contracting random nets is likelier to contract a dense cluster than a sparse cluster; this allows denser clusters to remain uncut.

5.3 Mathematical Programming

Mathematical programming optimizes an objective function subject to inequality constraints on the variables. (An equality constraint can be captured by two inequality constraints). A *linear program* (LP) requires every equation to be linear in terms of each variable. An LP can be solved in average-case polynomial time using the simplex method; interior methods such as that of Karmarkar have polynomial worst-case complexity; see e.g., [115] for a review. An *integer linear program* (ILP) is an LP with the additional constraint that the variables must take on integer values; solving general ILP instances is NP-Hard. A *quadratic program* (QP) is an LP with an objective that is quadratic in the variables, and a *quadratic boolean program* (QBP) additionally restricts the variables to $0-1$ values. Some QP and QBP formulations have polynomial solutions while others are NP-hard (see [73]).

5.3.1 Quadratic Formulations

The multi-way graph partitioning formulation with objective $F(P^k) = \sum_{i=1}^{k} |E(C_i)|$ and fixed cluster sizes $m_1 \geq m_2 \geq \ldots \geq m_k$ (where $|C_i| = m_i$) can be expressed as a QBP. Let $B = (b_{ij})$ be the $k \times k$ matrix with $b_{ii} = 0$ and $b_{ij} = 1$ if $i \neq j$, and let $X = (x_{ij})$ be the $n \times k$ assignment matrix. The QBP minimizes

$$F(P^k) = \sum_{i,j=1}^{n} \sum_{h,l=1}^{k} a_{ij}(x_{ih} b_{hl} x_{jl}) \tag{5.2}$$

subject to

$$\begin{array}{lll} \sum_{i=1}^{n} x_{ih} = m_h, & 1 \leq h \leq k, & \text{(prescribed cluster sizes)} \\ \sum_{h=1}^{k} x_{ih} = 1, & 1 \leq i \leq n, & \text{(each module belongs to exactly one cluster)} \\ x_{ij} \in \{0,1\}, & 1 \leq i,j \leq n & \text{(legal assignment matrix)} \end{array}$$

To see why equation (5.2) holds, consider whether the edge (v_i, v_j) is cut. If it is, there will be a unique h and l such that $v_i \in C_h$ and $v_j \in C_l$, and only for these values will both $x_{ih} = 1$ and $x_{jl} = 1$, hence $a_{ij} \sum_{h,l=1}^{k}(x_{ih} b_{hl} x_{jl}) = a_{ij}$; the cost of (v_i, v_j) appears once in the sum. If the edge (v_i, v_j) is not cut, then v_i and v_j are in the same cluster, so $h = l$ and when both $x_{ih} = 1$ and $x_{jl} = 1$, we have $b_{hl} = 0$ which implies that $a_{ij} \sum_{h,l=1}^{k}(x_{ih} b_{hl} x_{jl}) = 0$.

Shih and Kuh [173] extended this formulation to handle timing and area constraints. In this case, the matrix B captures the cost of wiring between clusters; previously we had assumed that this cost is uniformly one, but it may be generalized (e.g., for MCM partitioning). The area constraints $w(C_h) \leq U_h, 1 \leq h \leq k$ can be incorporated by replacing the constraints $\sum_{i=1}^{n} x_{ih} = m_h$ by $\sum_{i=1}^{n} w(v_i) x_{ih} \leq m_h$. To capture timing constraints, the authors of [173] use the $n \times n$ matrix $D^C = (D^C_{ij})$ to store the maximum allowable signal routing delay between every pair of modules, and the $k \times k$ matrix $D = (D_{ij})$ to store the cost of routing between every pair of clusters. Hence, the timing constraints are $D_{hl} \leq D^C_{ij}, 1 \leq h, l \leq k$ and $1 \leq i, j \leq n$ whenever $x_{ih} = x_{jl} = 1$. Finally, the $n \times k$ matrix $P = (p_{ih})$ stores the cost of assigning a module v_i to C_h. Shih and Kuh's proposed objective is

$$\alpha \sum_{i=1}^{n} \sum_{h=1}^{k} p_{ij} x_{ij} \quad + \quad \beta \sum_{i,j=1}^{n} \sum_{h,l=1}^{k} a_{ij}(x_{ih} b_{hl} x_{jl}), \tag{5.3}$$

where the first term gives the cost of assigning modules to clusters, and the second term gives the wiring costs. The coefficients α and β can be used to trade off between the terms, but in practice both are set to one. When the entries in B are permitted to take on any values, the formulation in equation (5.2) becomes the quadratic assignment problem (QAP). Shih and Kuh show how to adapt a QAP heuristic due to Burkard and Bonniger [34] to their formulation. The same heuristic was also applied by [62] to an opto-electronic formulation with size, connection, power, and interconnect constraints.

Shih, Kuh, and Tsay [175] also applied the formulation of equation (5.3) to am MCM partitioning formulation, setting $\alpha = 1$ and $\beta = 0$ to obtain an ILP. The method assumes a good initial partitioning, although timing and area constraints may be violated. The matrix P is used to capture the cost of perturbing the current solution into a new solution. If $x_{ih} = 1$ in the current solution, then $p_{il} = \frac{n \cdot w(v_i)}{w(V)} dist(C_h, C_l)$ for $1 \leq l \leq k$, where $dist(C_h, C_l)$ is the Manhattan distance between C_h and C_l in the MCM; this is the cost of moving v_i from C_h to C_l. The objective is then to satisfy timing and area constraints while minimizing the perturbation of the current solution. Shih et al. apply a *constraints decoupling* technique, separating the problem into one with only area constraints and one with only timing constraints. The algorithm iteratively solves the two problems separately and uses the two solutions to reformulate for the next iteration, until the solutions converge.

We note one final quadratic (boolean) formulation due to Khan and Madisetti [118] which integrates area, pin, and yield constraints for MCM partitioning. They apply a linearization technique which approximates the QBP by an ILP, adding a new constraint for each variable in the objective. This ILP is heuristically solved using branch and bound and linear programming relaxation.

5.3.2 Retiming

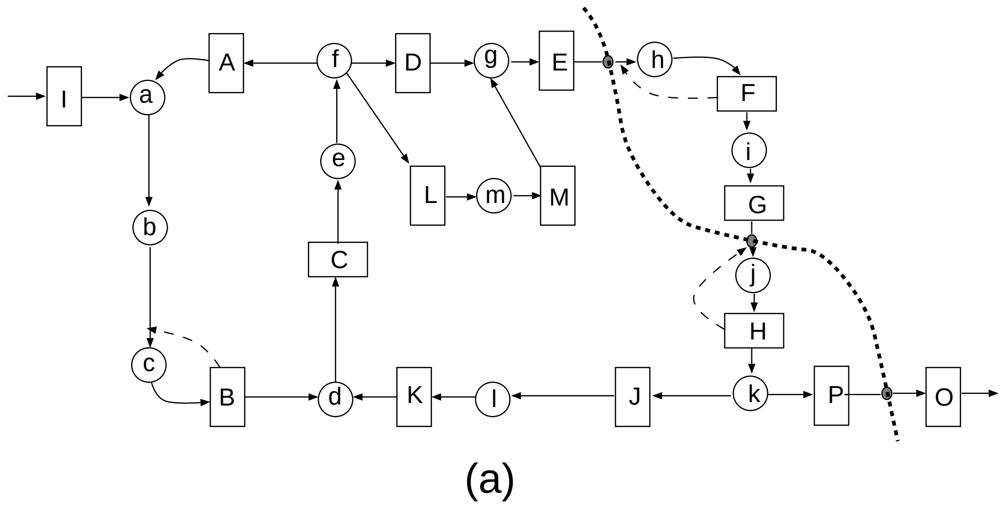
Liu et al. [137] propose a timing formulation similar to the Min-Delay Clustering problem of Section 2.4, except that the cost of a path p is extended to loops in the circuit. Assume that V consists of a set of registers $R(V)$ and a set of combinational logic blocks $C(V)$, and assume that $\delta(r) = 0 \forall\ r \in R(V)$. Liu et al. define the *iteration bound* as $L = \max_{loops\ l} (\lceil \frac{cost(l)}{r(l)} \rceil)$ where $r(l)$ is the number of registers in loop l. The clock cycle T is the maximum delay between consecutive registers on any path. L gives a lower bound on T: in the loop l with maximum L, $cost(l)$ is the delay of a loop and $r(l)$ is the number of registers, hence the longest delay between consecutive registers must be at least $\frac{cost(l)}{r(l)}$. The *latency* on a path p from $v_i \in PI$ to $v_j \in PO$ is defined as the minimum number of clock cycles that pass between a signal arrival at v_i and its first effect on the signal output at v_j. The latency is determined by the *critical path* p between v_i and v_j with the fewest registers; hence the *latency bound* for the circuit is defined as $\max_{paths\ p} cost(p)$ over all PI-to-PO paths. The *retiming problem* seeks a rearrangement of registers along their paths, such that the clock cycle and circuit latency are as close as possible to the iteration and latency bounds.

As an example, consider the circuit in Figure 17(a) in which registers are represented by rectangles and combinational blocks by circles. Assume the intercluster delay is 2 instead of 1 (following [137]) and that $\delta(v) = 1, \forall\ v \in C(V)$. The shortest loop in the circuit that crosses the cut contains 9 combinational blocks and crosses the cut twice for a delay of 13 units. Since this loop contains 7 registers, the iteration bound is $\lceil \frac{13}{7} \rceil = 2$, which is also the iteration bound for the entire circuit. We have $T = 3$ before retiming since the path from E to F crosses the cut once (2 units) and then passes through the combinational block h (1 unit). The paths from A to B and from G to H also force a 3-unit clock cycle; however, after retiming (moving B, F,and H as designated) T is reduced to 2 units, which is optimal. The latency after retiming is 18 since there are 9 registers on the longest path from I to O; however, in (b) a different cut plus retiming allows the removal of register P, reducing the latency to 16 units.

Liu et al. [136] add clock and latency constraints into the above QBP formulation. They only consider bipartitioning; thus, the objective in equation (5.2) reduces to

$$\sum_{i,j=1}^{n} a_{ij}(x_{i1}x_{j2} + x_{i2}x_{j1})$$

since $b_{ij} = 0$ if $i = j$. Liu et al. also adapt the same area constraints $\sum_{i=1}^{n} w(v_i)x_{ih} \leq U_h, 1 \leq h \leq k$ and cluster membership constraints $x_{i1} + x_{i2} = 1, 1 \leq i \leq n$. If an iteration bound of L is required,



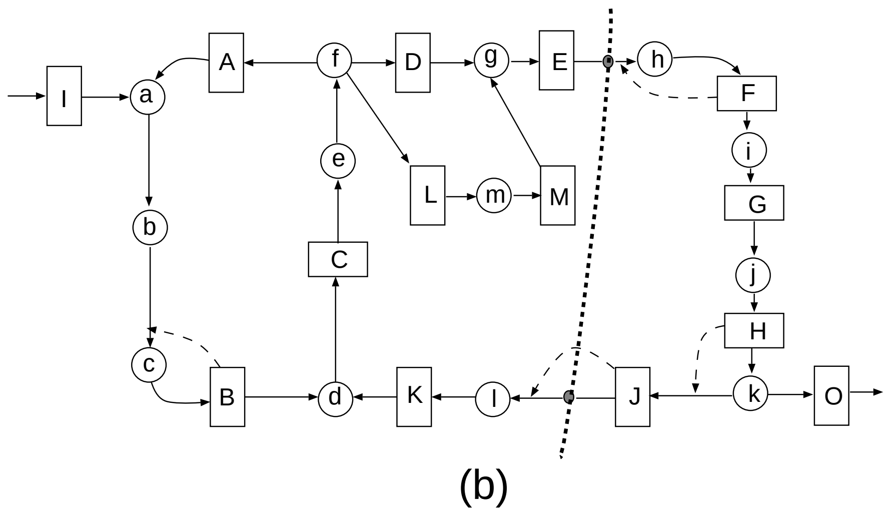


Figure 17: (a) A bipartitioning $T = 3$ before retiming, and $T = 2$ after retiming (with latency 18), and (b) a bipartitioning with $T = 2$ and 16 units latency after retiming.

the constraints

$$\frac{1}{r_l}\left(\sum_{v_i \in l} \delta(v_i) + \sum_{(i,j) \in l} (x_{i1}x_{2j} + x_{i2}x_{j1})\right) \leq L, \quad \forall \text{ loops } l$$

are added. The term in large parentheses gives the total delay for a signal to traverse loop l, so the entire left term gives the iteration bound. To enforce a latency bound H, the QBP adds constraints

$$\sum_{v_i \in p} \delta(v_i) + \sum_{(i,j) \in p} (x_{i1}x_{2j} + x_{i2}x_{j1}) \leq H, \quad \forall \text{ IO-critical paths } p$$

Liu et al. then decompose the problem into primal and dual subproblems; the primal problem is solved using the heuristic of [34], and the dual problem is solved using a subgradient method.

5.3.3 A Transportation Problem

Barnes et al. [19] use the QBP discussed at the beginning of this subsection, except that they *maximize* the number of uncut edges, rather than equivalently minimizing the number of cut edges. This objective can be expressed using equation (5.2) with B the identity matrix. The objective becomes:

$$\sum_{i,j=1}^{n}\sum_{h,l=1}^{k} a_{ij}(x_{ih}b_{hl}x_{jl}) = \sum_{i,j=1}^{n}\sum_{h=1}^{k} a_{ij}(x_{ih}x_{jh}) = \sum_{h=1}^{k}\sum_{i,j=1}^{n} x_{ih}a_{ij}x_{jh} = \sum_{h=1}^{k} \vec{X_h}^T A\vec{X_h} \tag{5.4}$$

where $\vec{X_h}$ is the indicator vector for cluster C_h. Notice the similarity between this equation and Hall's quadratic formulation in equation (4.1). Since optimizing equation (5.4) is NP-complete, Barnes et al. try to approximate it as a *transportation problem*, a specific type of ILP that can be solved efficiently. Let $A = QQ^T$ denote a Cholesky factorization of A into the product of lower- and upper-triangular matrices, and let $\vec{q_i}$ be the i^{th} row of Q. If we let $\vec{r_h} = \frac{1}{m_h}\sum_{i=1}^{n}\vec{q_i}x_{ih}$ then the following equality can be established:

$$\sum_{h=1}^{k}\sum_{i=1}^{n}(||\vec{q_i} - \vec{r_h}||^2 - ||q_i||^2)m_h x_{ih} = -\sum_{h=1}^{k}\vec{X_h}^T A\vec{X_h} \tag{5.5}$$

This equation appears to be a linearization of equation (5.4) except that $\vec{r_h}$ depends on $\vec{X_h}$. Barnes et al. propose to fix the $\vec{r_h}$ vectors and optimize the transportation problem that is linear in the x_{ih} variables. Thus, the algorithm begins with an initial assignment matrix X corresponding to P^k and computes the $\vec{r_h}$ vectors from X. These vectors are then fixed for solving the transportation problem in equation (5.5), and the $\vec{r_h}$ vectors are then recomputed for the new X. Barnes et al. prove that the each partitioning solution corresponding to X will have cost no higher than the previous one, and that their algorithm will eventually converge.

Recall from Section 4.2 that Barnes [18] also proposed another transportation problem that approximates the multi-way partitioning objective. He tried to minimize the error $||U_k - R||^2$ which is equivalent to minimizing

$$\sum_{i=1}^{n}\sum_{h=1}^{k}(\frac{\mu_{ih}}{\sqrt{m_h}})x_{ih}$$

subject to the same constraints as in equation (5.2).

5.3.4 Programming and Placement

We now discuss applying mathematical programming techniques for the 1-dimensional placement problem. Recall that a bipartitioning can be represented by a $0-1$ indicator vector $\vec{x}$, but $\vec{x}$ can also represent a 1-dimensional placement if its entries take on real values. A linear placement also induces a linear ordering, and as seen in Section 4.5, linear orderings and placements can serve as the basis for a partitioning algorithm.

The placement formulation of Tsay and Kuh [183] assumes that the coordinates of n legal *slots* $s_1 \le s_2 \le \ldots \le s_n$ are given, and a solution consists of an assignment of module locations $\{x_i\}$ to slots $\{s_j\}$. The objective is to minimize the squared wirelength ($\vec{x}^T Q \vec{x}$ in equation (4.1)) of the placement (i.e., the slot assignment). For example, if the placement is to correspond to a bipartitioning into clusters of sizes m_1 and m_2, then $s_1 = s_2 = \ldots = s_{m_1} = 0$ and $s_{m_1+1}, s_{m_1+2}, \ldots s_n = 1$. Tsay and Kuh enforce the slot constraints via a set of nonlinear equations:

$$\begin{array}{ll}
\sum_{i=1}^n x_i = \sum_{i=1}^n s_i = c_1 & \text{first-order constraint} \\
\sum_{i=1}^n x_i^2 = \sum_{i=1}^n s_i^2 = c_2 & \text{second-order constraint} \\
 & \ldots \\
\sum_{i=1}^n x_i^n = \sum_{i=1}^n s_i^n = c_n & n^{th}\text{-order constraint}
\end{array}$$

We have already seen that the second eigenvector $\vec{\mu_2}$ of Q gives the optimal solution if only the first-and second-order constraints are considered. The difficulty lies in the 3^{rd}- through n^{th}-order constraints. Like Blanks [24], Tsay and Kuh observe that the closest legal solution to $\vec{\mu_2}$ can be obtained by sorting the entries of $\vec{\mu_2}$, although this solution will generally be suboptimal. They propose the SCAN postprocessing algorithm which improves on the $\vec{\mu_2}$ solution by gradually including higher order constraints and resolving. In their implementation, subsequent solutions are functions of $\vec{\mu_2}, \vec{\mu_3}$ and $\vec{\mu_4}$, but in general, the higher-order constraints cannot be directly mapped to an eigenvector-based solution.

Another formulation of Tsay and Kuh [183] ignores slot constraints and assumes that α modules have already been assigned to fixed locations, while the other $\beta = n - \alpha$ modules are free to move. Let $\vec{x_\beta} = [x_1, x_2, \ldots, x_\beta]^T$ be the coordinates of movable modules and let $\vec{x_\alpha} = [x_{\beta+1}, x_{\beta+2}, \ldots, x_n]^T$ be the coordinates of the fixed modules. The optimal locations of the movable modules can be determined by solving $Q_{\beta\beta}\vec{x_\beta} + Q_{\beta\alpha}\vec{x_\alpha} = 0$ where

$$Q = \left[\begin{array}{cc} Q_{\beta\beta} & Q_{\beta\alpha} \\ Q_{\alpha\beta} & Q_{\alpha\alpha} \end{array} \right].$$

Tsay and Kuh recommend solving this sparse system of linear equations using a successive over-relaxation (SOR) technique. Since $Q_{\beta\beta}$ is real, symmetric, positive definite and diagonally dominant, convergence is guaranteed; in practice, SOR converges in time nearly linear in the number of modules.

Kleinhans et al. [120] applied quadratic programming to a hypergraph representation in their GORDIAN cell placement program. Each net e has an associated location $x_e = \frac{1}{|e|}\sum_{v_i \in e} x_i$ which is the "center of gravity" of all of its pins. Assuming a star topology, the squared wirelength objective becomes $\Phi_q(\vec{x}) = \sum_{e \in E}\sum_{v_i \in e}(x_i - x_e)^2$, and setting x_e to be the center of gravity is optimum for this objective. Sigl et al. [177] improved GORDIAN by incorporating a linear wirelength objective, which can be written as

$$\Phi_l(\vec{x}) = \sum_{e\in E}\sum_{v_i\in e}|x_i - x_e| = \sum_{e\in E}\sum_{v_i\in e}\frac{(x_i-x_e)^2}{|x_i-x_e|} = \sum_{e\in E}\sum_{v_i\in e}\frac{(x_i-x_e)^2}{g_{ie}} \quad \text{where} \quad g_{ie} = |x_i - x_e|$$

This objective can be approximated by minimizing

$$\Phi_l(\vec{x}) \approx \sum_{e\in E}\frac{1}{g_e}\sum_{v_i\in e}(x_i-x_e)^2 \quad \text{where} \quad g_e = \sum_{v_i\in e} g_{ie} = \sum_{v_i\in e}|x_i - x_e| \tag{5.6}$$

This approximation reduces the influence of nets with many pins. In addition, it reduces the influence of a module v_i that is close to an incident net's location x_e (i.e., induces a large $\frac{1}{g_{ie}}$ term); the substituted $\frac{1}{g_e}$ term is generally proportional to the span of e in a linear placement. This approach has led to both MCM [158] and FPGA [157] partitioning methods.

The quadratic programming formulation can be efficiently solved using a conjugate gradient method when each g_e is constant. Let g_e^r denote the value for g_e during the r^{th} iteration of the solution. In the first iteration, $g_e^1 = 1$ and the quadratic program of equation (5.6) is solved. Given the placement $\vec{x}$ from the $r-1^{st}$ iteration, $g_e^r = \max\{w_0, \sum_{v_i\in e}|x_i - x_e|\}$ in the r^{th} iteration, where $w_0 > 0$ is a small constant that is used to avoid numerical error. This process of solving the quadratic program and adjusting the g_e^r terms is repeated until $\sum_{e\in E}|g_e^{r-1} - g_e^r| < \epsilon$, for some choice of ϵ. Notice the similarity between this approach and that of [19] in which the $\vec{r_h}$ vectors are adjusted after each iteration until convergence.

Hagen and Kahng [81] approximated the linear objective function by squaring it and ignoring the numerous mixed terms, yielding $\sum_{1\le i,j\le n} a_{ij}^2(x_i - x_j)^2$, i.e., the traditional quadratic placement objective with squared coefficients. The more general question of finding a value c such that $a_{ij}^c(x_i - x_j)^2$ best approximates linear wirelength was also raised. All of these placement techniques ([183] [177] [81]) can be used to iteratively generate a bipartitioning by solving the formulation, peeling off and fixing the extreme coordinates, and integrating these fixed coordinates into the next iteration. The PARABOLI scheme of Riess et al. [156] (see Section 4.5) uses the GORDIAN placement in exactly this manner, but no analogous work has used orderings based on SOR or squared coefficients. In addition, none of these orderings have been used with the dynamic programming algorithm of [5] to yield multi-way partitionings. We believe that constructing partitionings from placements derived via mathematical programming is a promising research direction.

5.4 Fuzzy Partitioning

The Fuzzy k-Means (FKM) algorithm is a well-known optimization technique for clustering problems that arise in such fields as geological shape analysis, medical diagnosis, image analysis, irrigation design, and automatic target recognition [35] [196]. The problem formulation generally involves clustering data points in multi-dimensional space, although the paradigm can be applied more generally.

A fuzzy partitioning can partially assign a module to several clusters, e.g., $\frac{1}{2}$ to C_1 and $\frac{1}{4}$ to C_2 and to C_3. The assignment matrix X is still used to represent a partitioning, with entries in X now real-valued; however, for any row i of X, we still have $\sum_{h=1}^{k} x_{ih} = 1$, i.e., v_i is assigned to a total of one cluster. There are k variable cluster centers, denoted by $w_1, w_2, \ldots, w_k$. Assume that a distance function $dist(v_i, w_h)$ is defined between modules and cluster centers (e.g., the distance function is obvious if the modules and cluster centers have physical locations). FKM begins with an initial fuzzy partitioning X, then iteratively modifies X to optimize the objective:

$$\sum_{i=1}^{n}\sum_{h=1}^{k}(x_{ih})^c \cdot dist(v_i, w_h)^2$$

where $c > 1$ is a user-chosen scalar. If we let $(v_{i1}, v_{i2}, \ldots, v_{id})$ and $(w_{h1}, w_{h2}, \ldots, w_{hd})$ be the respective coordinates (assuming locations in d-space) for v_i and w_h, then the coordinates for w_h are computed to be the weighted averages of the coordinates for modules with partial membership in C_h. This expression is given by

$$w_{hj} = \frac{\sum_{i=1}^{n}(x_{ih})^c v_{ij}}{\sum_{i=1}^{n}(x_{ih})^c} \qquad h = 1, 2, \ldots, k, \quad j = 1, 2, \ldots, d$$

For all $i = 1, 2, \ldots, n$ and $h = 1, 2, \ldots, k$, if the coordinates for v_i and w_h are the same, then x_{ih} is set to 1 and x_{ij} is set to 0 for all other $j \neq h$. Otherwise, the new fuzzy partitioning has entry

$$x_{ih} = \left(\sum_{l=1}^{k}\left(\frac{dist(v_i, w_h)}{dist(v_i, w_l)}\right)^{2/(c-1)}\right)^{-1} \qquad i = 1, 2, \ldots, n, \quad h = 1, 2, \ldots, k$$

The computation of cluster center coordinates and a new fuzzy partitioning is alternated until X no longer changes by a significant amount. Then, rounding to its closest discrete assignment is used to to derive a non-fuzzy partitioning solution.

Ball et al. [16] have shown how to use FKM in conjunction with GORDIAN to derive 2-dimensional placements, and Razaz [153] earlier did the same by first modifying FKM to handle graph partitioning without specified coordinates. Razaz uses the i^{th} row $\vec{a_i}$ of the adjacency matrix A to represent the coordinates for v_i, so the cluster centers are given by the n-vectors $\vec{w_1}, \vec{w_2}, \ldots, \vec{w_k}$. The distance between module v_i and cluster center w_h is simply $dist(v_i, \vec{w_h}) = ||\vec{a_i} - \vec{w_h}||$. Given a fuzzy partitioning X, the new cluster centers are given by

$$\vec{w_h} = \frac{\sum_{i=1}^{n}(x_{ih})^2 \vec{a_i}}{\sum_{i=1}^{n} x_{ih}} \qquad h = 1, 2, \ldots, k$$

The use of 2 for the exponent in the numerator and 1 for the exponent in the denominator seem to be an arbitrary choice of [153].

5.5 Boolean Set Covering

A new approach to multi-way partitioning was recently proposed by Chou et al. [44] to address the the Single-Device FPGA Partitioning problem. Chou et al. initially construct a multi-way partitioning with each cluster satisfying the device's size and pin constraints, except that module *overlapping* is permitted, i.e., a module can be a member of more than one cluster.[13] This feasible solution (i.e., one that satisfies all constraints) with overlapping is then transformed into a feasible solution without overlapping. That such a transformation can always be achieved is due to the FPGA Complementary Theorem, which states that if $C_1, C_2, \ldots, C_k$ are k feasible FPGA clusters, then there exits a permutation $\pi : [1..k] \rightarrow [1..k]$ such that

$$C_{\pi(1)}, C_{\pi(2)} \backslash C_{\pi(1)}, C_{\pi(3)} \backslash (C_{\pi(1)} \cup C_{\pi(2)}), \ldots, C_{\pi(k)} \backslash (C_{\pi(1)} \cup C_{\pi(2)} \cup \ldots \cup C_{\pi(k-1)})$$

are mutually disjoint, feasible clusters. (This result was extended to Multiple-Device FPGA Partitioning in [100].) Thus, one may find a minimum *set covering* of the modules, i.e., a minimum set of feasible clusters that covers (contains) all the modules, and this partitioning can always be transformed into a non-overlapping partitioning with the same number of clusters. Although the set covering problem is NP-Complete, the *Espresso II* tool [28] provides a well-developed set-covering heuristic used in minimizing Boolean expressions. Chou et al. adapt the main ideas behind Espresso into their set covering algorithm. This approach is attractive because a highly developed tool for a very different problem was utilized for module partitioning.

6 Clustering Approaches

Clustering (i.e., k-way partitioning for large $k = \Theta(n)$) is rarely a goal in and of itself. Rather, a clustering solution is typically used to induce a smaller and more tractable problem instance. Many clustering algorithms utilize a *bottom-up* approach: each module initially belongs to its own cluster, and clusters are gradually merged or grown into larger clusters until the desired decomposition is found. We classify bottom-up approaches as *agglomerative* if new clusters are formed one at a time and *hierarchical* if several new clusters may be formed simultaneously. Other intuitive approaches involve random walks, iterative peeling of clusters, vertex orderings, and simulated annealing. Another set of approaches are specific to (acyclic) combinational Boolean networks. This section begins with motivations for clustering, and then discusses the various clustering approaches. We conclude by surveying methods which integrate clustering into a move-based partitioning strategy.

[13] In the Single-Device FPGA formulation, the constraints for each cluster can be evaluated independently of the other clusters; hence, overlapping allows modules to be assigned to multiple clusters though this may have no physical meaning. Note the difference between overlapping and replication, in which the netlist is modified according to specified rules.

6.1 Motivations for Clustering

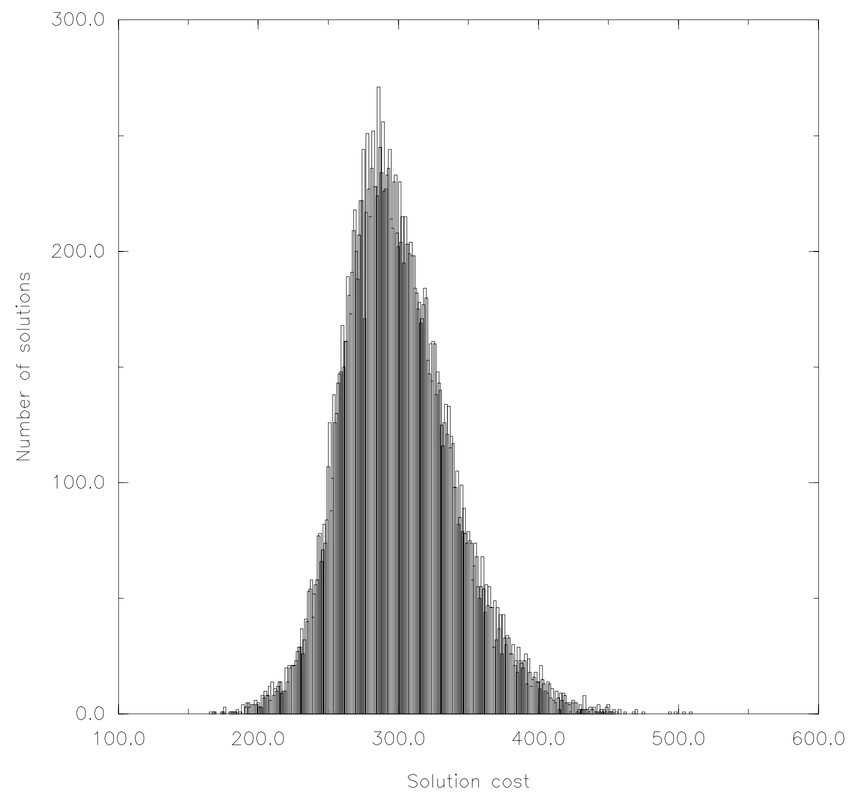


Figure 18: Distribution of 21,203 FM bisection solutions (i.e., local minima) for SIGDA Layout Synthesis benchmark Primary2 (3014 modules). Each solution was generated from a new random starting point.

As noted above, move-based approaches, and iterative improvement in particular, are the most common partitioning algorithms in current CAD tools. A common weakness of these approaches is that the solution quality is not "stable", i.e., predictable. Figure 18 shows the FM solution cost distribution for the Primary2 benchmark; we see that the distribution is roughly normal and that the average FM solution is significantly worse than the best FM solution. This result reflects the "central limit catastrophe" [116] of move-based methods, i.e., that most local optima tend to be of only average quality. In hopes of hitting the "good tail of the distribution", many practical implementations of FM use a *random multi-start* approach, i.e., the algorithm is run many times from random starting solutions, and returns the best solution found. However, as discussed in Section 3.4 and as shown in

Figure 18, hundreds of runs may be necessary to achieve stable performance.

Given a clustering solution $P^k = \{C_1, C_2, \ldots, C_k\}$, we can reduce the instance size from n to k by constructing the *contracted* hypergraph $H'(V', E')$ with $V' = \{C_1, C_2, \ldots, C_k\}$. For every $e \in E$, there is a hyperedge $e' \in E'$ with $e' = \{C \mid \exists v \in e \cap C\}$ (unless $|e'| = 1$), i.e., each cluster in e' contains some module that has a pin of e. The contraction process will also decrease the sparsity of the netlist, i.e., the average degree $deg_{avg} = \sum_{e \in E} |e|/|V|$ will be higher for the contracted hypergraph. Goldberg and Burstein [76] claim that deg_{avg} is between 1.8 and 2.5 for real circuits, and that an FM-based algorithm performs "relatively poorly" in this range, but "nearly optimally" when $deg_{avg} > 5$. Lengauer [133] also conjectures that for graphs with high density and large minimum degree, there will be few local optima that are not global optima. Finally, the analysis of Saab and Rao [163] also suggests that KL performance improves with increasing graph density. A bipartitioning is said to be *m-optimal* if swapping any m modules from C_1 with m modules from C_2 cannot decrease the cost, e.g., KL is a 1-optimal algorithm. Saab and Rao give a performance bound which holds for any 1-optimal heuristic, and becomes tighter as the number of edges in the graph increases. Specifically, if $(\frac{1+\epsilon}{1+2\epsilon}) \cdot (\frac{n(n-1)}{2})$ edges are present, then the heuristic bisection width is suboptimal by a factor of at most ϵ (e.g., if $\frac{2}{3} \cdot (\frac{n(n-1)}{2})$ edges are present, a 1-optimal bisection can have cost at most twice optimal). All of these works imply that a move-based heuristic should be more effective on a contracted netlist not only because of the smaller solution space, but also because of the increased density.

These considerations have motivated the "two-phase" application of move-based algorithms [76]: first execute the given algorithm on a contracted netlist, then re-expanded the resulting solution into the starting point of a second algorithm run on the flat (original) netlist. The two-phase approach has been applied with many clustering algorithms, e.g., [3] [4] [29] [30] [51] [82] [94] [143] [176]; see Section 6.6 for a discussion of methods which integrate clustering into move-based approaches.

6.2 Agglomerative Clustering

An *agglomerative* clustering algorithm begins with the n-way clustering $P^n = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$ and iteratively constructs P^k from P^{k+1} as follows:

- Choose two clusters C_h and C_l from P^{k+1}.
- Construct P^k from P^{k+1} by removing C_h and C_l, and adding the merged cluster $C = C_h \cup C_l$.

Agglomerative clustering was first proposed by S. C. Johnson [111] for weighted complete graphs. The criterion for choosing clusters C_h and C_l is what distinguishes among agglomerative variants, e.g., [111] merges the two clusters that minimize the diameter of the new cluster C. This minimum-diameter criterion was used for circuit bipartitioning in [3]: modules were first mapped to points (i.e., singleton clusters) in multi-dimensional space (see Section 4), the diameter criterion was used to merge clusters, the resulting clustering was used within two-phase FM.

For undirected graphs, Karger [114] proposed a heuristic which begins with all vertices as isolated clusters. Iteratively, a random edge is chosen and its incident clusters are contracted into a single cluster (see Section 5.2). The approach can be extended to hypergraphs by picking a random net (perhaps with size-dependent probability) and contracting two random incident clusters (or all incident clusters). An alternative greedy approach would be to simply merge the two clusters with highest connectivity, i.e., choose C_h and C_l such that $N_{hl} = |E(C_h) \cap E(C_l)|$ is maximum. Schuler and Ulrich [168] observe that such a criterion ignores the number of nets cut resulting from the merged cluster. Such a criterion will generally also fail to construct balanced clusterings. Thus, the authors of [168] maximize a merging objective of form:

$$f(w(C_h))\frac{N_{hl}}{|E(C_h)| - N_{hl}} + f(w(C_l))\frac{N_{hl}}{|E(C_l)| - N_{hl}} \tag{6.1}$$

where f is a function of the cluster weights. The resulting cluster hierarchy is used to form a 1-dimensional placement. More recently, Shin and Kim [176] proposed a slightly different approach: given P^{k+1}, they merge C_h and C_l which maximizes

$$\frac{N_{hl}}{\min\{E(C_h), E(C_l)\}} - \alpha\frac{k \cdot (w(C_h) + w(C_l))}{w(V)}.$$

The first term captures the connectivity between clusters, with the denominator favoring the selection of "outlier" clusters (e.g., a cluster with connections to only one other cluster) for merging. The second term implies a penalty for making a cluster too large; if the merged cluster has average weight in the resulting P^k, the term reduces to α. The authors of [176] proposed this clustering approach for use with two-phase FM; their particular scheme makes many runs on the contracted netlist before the netlist is flattened for the second FM phase.

Ng et al. [143] propose an agglomerative algorithm based on Rent's rule; Rent's rule describes an average-case phenomenon in "good" circuit placements, i.e., that there is a power-law relationship $|E(C)| = d(C) \cdot |C|^r$ between the number of edges incident to a cluster and the size of the cluster (see [128] and [85] for reviews). Here, $d(C) = \frac{1}{|C|}\sum_{v \in C} deg(v)$ is the average degree of modules in the cluster. Because the layout has finite size, the power-law relationship eventually breaks down when the number of clusters is small. Feuer [64], Donath [57] and others have established that a lower Rent parameter r corresponds to lower total wirelength in the layout. However, the connection between such results for placement (where the netlist is embedding in the plane) and the clustering domain (where the netlist is "free") remains unclear (cf. [85]). The merging step chooses the pair of clusters C_h and

C_l that minimize the Rent parameter $0 \leq r \leq 1$ of the merged cluster C, where

$$r = 1 + \frac{\ln(E(C)) - \ln(d(C))}{\ln |C|},$$

The merging process continues until the prescribed number k of clusters is reached, or until r exceeds a prescribed constant.

6.3 Hierarchical Strategies

Generally, agglomerative methods will not be very efficient: finding the best pair of clusters to merge may require $O(k^2)$ time, unless a list of cluster merging costs is stored and updated (which will likely require $O(n^2)$ space). An alternative strategy is to find many good clusters to merge, then perform all merges simultaneously; we call this a *hierarchical* strategy. The difference between agglomerative and hierarchical strategies is illustrated for the 8-module example in Figure 19. In (a), the dendogram reveals the order in which clusters are merged; each dotted horizontal line is a level in the hierarchy, and an agglomerative algorithm will have $n - 1$ levels. Figure 19(b) shows a hierarchical algorithm that simultaneously merges as many cluster pairs as possible, yielding a hierarchy with $\lceil \log n \rceil$ levels.

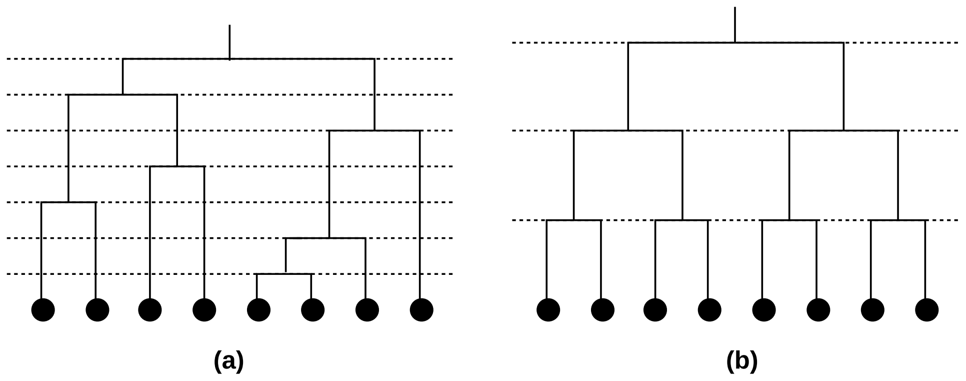


Figure 19: An 8-module example of (a) an agglomerative and (b) a hierarchical construction.

In [29] [30], Bui et al. propose *matching-based compaction* as a clustering strategy. A *matching* in the netlist is a set of disjoint pairs of modules, such that each pair shares a common net. Bui et al. find a random maximal matching and merge each pair of modules into a cluster: this will ideally result in a set of $\frac{n}{2}$ clusters; matchings can be iteratively computed on the contracted netlist, yielding a hierarchy of clusterings with size $\frac{n}{4}$, $\frac{n}{8}$, etc. A similar approach was proposed earlier by [76]. With the random matching approach, all clusters will have roughly the same size, although it is not known whether this is preferable, e.g., for two-phase bipartitioning.

Feo and Khellaf [63] proposed contracting edges according to a maximum-weight matching. Tight performance bounds were shown for this approach on graphs with edge weights that satisfy the triangle inequality, but the matching construction generally requires $O(n^3)$ time. Roy and Sechen [159] also proposed a deterministic contraction approach; they first construct a weighted undirected graph using the standard clique model, then contract all edges with weight higher than a given threshold. This approach differs from the matching-based approaches in module pairs are contracted more selectively.

Instead of merging pairs of modules, entire nets may be contracted. In the placement heuristic of [181], nets are first sorted in nondecreasing order by size, and a net is picked from the list and contracted into a cluster as long as none of its modules belong to a contracted cluster. In other words, this approach greedily selects a maximal subset of mutually disjoint nets, then contracts all the nets in this subset. This procedure is iteratively repeated on the contracted netlist until certain "fractal" criteria are met.

Another scheme, proposed by Hauck and Borriello [94], uses randomness and the merging criterion of [168]. Initially, each module belongs to its own singleton cluster; the clusters are then visited in a random order and each cluster is associated with the neighboring cluster with which it has highest connectivity, i.e., that maximizes equation (6.1) with $f(C_h) = \frac{1}{w(C_h)}$. If the current cluster in the order has already been associated with another cluster, it is ignored. Once each cluster has been visited, clusters are transitively merged with their associated neighbors.

6.4 Intuitive Cluster Properties

We now turn to a group of clustering approaches whose main common bond is that each tries to identify certain *intuitive* cluster properties, such as having dense cliques or cycles. Some of these intuitions are explicitly captured in a clustering objective that can be optimized, e.g., Absorption or ratio cut. We identify each approach by its underlying clustering intuition.

(K,L)-connectivity: Modules u and v are (K,L) *-connected* if there exists K edge-disjoint paths of length $\leq L$ between u and v. Garbers et al. [69] proposed using the transitive closure of the (K,L)-connectedness relation for clustering into dense subgraphs, (i.e., having large $\frac{|E|}{|V|^2}$ ratio). However, the method may not be that intuitive, e.g., in a 4-cycle (v_1, v_2, v_3, v_4), v_1 and v_3, along with v_2 and v_4, are $(2,2)$-connected, yet the clustering $\{\{v_1, v_3\}, \{v_2, v_4\}\}$ will cut all four edges. In addition, determining (K,L)-connectivity is NP-Complete for $L \geq 5$, with NP-Completeness for $L = 4$ still an open question. In practice, [69] ensures clusters are connected by placing two modules in the same cluster if K edge-disjoint paths between them can be found with at least one path having length one.

Cycles in Random Walks: A *random walk* in a netlist is a random sequence of modules $\{w_1, w_2, \ldots, w_K\}$ such that $N(w_i) \cap N(w_{i+1}) \neq \emptyset, \forall\ i = 1, \ldots, K-1.$. In other words, every consecutive pair of modules share a common net. Cong et al. [47] defined a *cycle* $\{w_p, w_{p+1}, \ldots, w_q\}$ as a contiguous subsequence of the random walk with $w_p = w_q$ and all w_i distinct, $i = p, p+1, \ldots, q-1$. The *maximum cycle* for v_i is the longest cycle that begins and ends at v_i; all maximal cycles are found in $O(N)$ time and then used to identify clusters. (Intuitively, a maximal cycle should identify a good cluster: if it contained any natural subcluster, a module in this subcluster would have likely appeared more than once within the maximal cycle.) Hagen and Kahng [82] assign v_i and v_j to the same cluster if their *sameness* is larger than zero, where sameness is computed as follows. Let $CC(i,j)$ be the number of times that v_j occurs in a cycle beginning at v_i. If $CC(i,j) = 0$ or $CC(j,i) = 0$, then the sameness of v_i and v_j is 0; otherwise, it is set to $2(CC(i,j) + CC(j,i))$ and an adjustment of $4CC(i,l) - CC(j,l)$ $(4CC(j,l) - CC(i,l))$ is added for each v_l such that $CC(j,l) > CC(i,l)$ $(CC(i,l) \geq CC(j,l))$. The intuition behind the adjustment term is that if v_l appears roughly the same number of times in cycles beginning with v_i and v_j, then v_i and v_j are similar; however, if v_l appears many more times in one module's cycle (i.e., by more than a factor of 4) then v_i and v_j are dissimilar. These random-walk approaches have been utilized within two-phase FM bipartitioning.

Cliques: A clique is a densest-possible subgraph, hence it forms a very intuitive cluster. Cong and Smith [51] propose to find and collapse cliques in the undirected graph constructed from the netlist by applying the clique net model with edge weight $\frac{2}{|e|}$ to all nets with $|e| \leq 5$. The algorithm searches for cliques of size r_0 and $r_0 + 1$ where r_0 is likely to be the size of the largest clique in the graph. A clique is contracted into a cluster as long as it satisfies the size and cardinality constraints $4w(C) \leq w(V), 3|C| \leq n$ as well as the density constraint that total edge weight is at least $\alpha \frac{2|E|}{n(n-1)}$ for some parameter α. The density constraint prevents single nets from forming clusters. While the approach is amenable to parallel processing, finding dense cliques by enumeration is computationally expensive. Also, by discarding medium-size and large nets to maintain sparsity, the algorithm may lose important clustering information.

Ratio Cut: Chou et al. [44] and Wei and Cheng [188] have constructed clusterings based on the ratio cut objective. The algorithm of [44] applies the ratio cut algorithm of [187] to construct a bipartitioning $\{C_1, C_2\}$ such that $C_1 \leq U$ for some small U, e.g., $U = 50$. Cluster C_1 is then removed from the circuit and the algorithm is repeated until the circuit has less than U modules. The strategy of finding a good cluster and "peeling" it from the circuit has used in [101] for the Density objective (see Section 5.2). Chou et al. apply their clusterings to the FPGA Set Covering formulation discussed in Section 5.5. The ratio cut clustering approach of Wei and Cheng [188] iteratively applies the bipartitioning algorithm of [187] to the largest remaining cluster; the top-down divisive process ends when the largest cluster contains less than 2% of the modules. This method seems to be the only "top-down clustering" approach in the literature. The clustering solutions were originally integrated within two-phase FM, and later used in a simulated-annealing based placement algorithm [90].

Absorption: Sun and Sechen [179] believe that clusterings that maximize the Absorption objective are best suited for use in the TimberWolf simulated annealing-based placement program. They used $L = 3Y$ and $U = 30Y$ for the cluster size constraints $L \leq w(C_h) \leq U, 1 \leq h \leq k$ where $10Y$ is the designated mean cluster size, but did not specify how Y is determined. TimberWolf's neighborhood structure is based on swapping clusters, hence the need for size constraints: too much variance in cluster sizes may make the placement solution infeasible. In [179], the process itself also uses simulated annealing, with neighborhood structure based on moving a single module from one cluster to another. The authors claim that $100n$ moves are sufficient to optimize Absorption.

Ordering Contiguity: Alpert and Kahng [4] proposed constructing a linear ordering of the modules, then constructing a clustering by splitting the ordering. The intuition is that if the ordering can appropriately traverse the circuit, then the clustering will be of high quality. Their WINDOW ordering scheme begins with an empty cluster C and iteratively adds the module with highest *attraction* to the cluster, where the attraction function is defined to reflect a given objective. The authors of [4] have given attraction functions for a variety of objectives, including Scaled Cost and Absorption. In addition, they showed how attraction can capture breadth-first and depth-first orderings, along with "max-adjacency" and "min-perimeter" criteria (cf. the Nagamochi and Ibaraki [141] in Section 5.2). Since C eventually becomes much larger than is desired for any single cluster, only a *window* of the most recently ordered modules are used to compute the attraction; the size of the window reflects cluster size constraints. The ordering is split into a clustering using the DP-RP algorithm discussed in Section 4.5.

6.5 Clustering of Boolean Networks

The final three approaches all exploit the directed structure of combination Boolean networks.

Corollas: Dey, Brglez and Kedem [55] [56] construct a directed "star" representation for a combinational Boolean network by introducing a dummy *stem* module $v(e)$ for each net e. For every $e \in E$, the DAG contains edges $(S(e), v(e))$ and $(v(e), w)$ for every $w \in D(e)$. Observe that stems are the only modules with multiple fan-outs, i.e., a stem $v(e)$ may have edges $(v(e), u), (v(e), w), u \neq w$. If there exist two disjoint paths from a stem to another module, the stem is called a *reconvergent fanout stem*, and the module is a *reconvergent module*. A *petal* is defined as the set of all modules located on all paths from a reconvergent fanout stem to all of its reconvergent modules; the petal is the basic primitive of the clustering algorithm. Dey et al. construct a clustering by first finding all petals, then merging overlapping petals into *corollas*, with corollas being maximal with respect to the overlapping property. Each individual corolla may be resynthesized using algebraic/boolean factoring (e.g., *Espresso* [28]) and if the new factorization will likely reduce the layout area, the resynthesized corolla is "glued" back in place of the original one. Works by the same authors show the implications of corolla-based clustering for logic synthesis, delay reduction, testability, etc.

MFFCs: Cong and Ding [46] use the standard DAG representation of a combinational network to find clusters for FPGA technology mapping. For every $v \in V$, the authors of [46] inductively define a *fanout free cone of* v, $FFC_v \subseteq V$, as follows: (i) $v \in FFC_v$, and (ii) if $u \in FFC_v$, $u \neq v$, and $(u, w) \in E$ for some w, then $w \in FFC_v$. In other words, and FFC is a single-output subnetworks. A *maximum fanout-free cone* for v ($MFFC_v$) is an FFC_v such that if $w \notin PI$ and $w \in MFFC_v$ then for every $(u, w) \in E$, $u \in MFFC_v$. The intuition is that MFFCs are naturally suited for technology mapping onto lookup table-based logic blocks which have a single primary output and a fixed number of primary inputs. In [50] MFFC clusters are used for acyclic, multi-way partitioning (see Section 3.5); here, the cluster sizes are heuristically thresholded since an MFFC can be quite large.

Cones: Given $v \in PO$ for a given DAG, Saucier, Brasen and Hiol [167] define the *cone* of v to be the set of all $w \in V$ such that there is a path from w to v. The authors of [167] use cones as a building block for a multi-way FPGA partitioning algorithm. For each $v \in PO$, the cone Λ_v is constructed, and clusters C are formed by overlapping cones and extracting the "coarsest common cluster", i.e., C is a cluster if for every cone Λ_v, $C \cap \Lambda_v = C$ or $C \cap \Lambda_v = \emptyset$. A new clustering is then agglomeratively constructed by iteratively finding C_h with maximum $\frac{|E(C_h)|}{w(C_h)}$ and merging it with C_l that maximizes $\frac{|E(C_h \cup C_l)|}{w(C_h \cup C_l)}$. The merging continues as long as all the clusters satisfy prescribed size and pin constraints. Saucier et al. applied this method to the Single-Device FPGA Partitioning problem, and gave a modification which handles critical path delays. Each critical path lies entirely within a single cone, so by not splitting the cone the path will be contained in a single cluster.

6.6 Integration of Clustering into a Bipartitioning Heuristic

Many of the above methods were originally proposed for use within an FM-based bipartitioning approach. The simplest way to incorporate a clustering solution is via the two-phase approach, i.e., run FM on the contracted netlist, and use the result as the starting solution of a second run on the flattened netlist. However, more sophisticated techniques may be preferable.

For example, Shin and Kim [176] run a variant of FM on the clustered netlist ten times on each of five different clustering solutions (with $k = \frac{n}{12}, \frac{n}{11}, \frac{n}{10}, \frac{n}{9}, \frac{n}{8}$). The FM variant is then run only once on the flattened netlist, using the best of the 50 clustered bipartitionings as a starting solution. Note that the FM variant actually consists of multiple passes where the cluster size constraints become tighter within each pass; [176] states that this procedure allows closely coupled cells to settle first, with freer cells moving closer to an optimum bipartitioning in later passes. Hence, this scheme makes hundreds of FM passes on clustered netlists before making a single pass on a flattened netlist. It is difficult to evaluate whether this method is the most effective use of FM passes: certainly, since FM passes come "cheaply" (in linear time), they can be integrated within a clustering scheme in many creative ways.

Hauck and Borriello [94] discuss two *unclustering* methods which integrate cluster hierarchy into the application of FM. The *iterative unclustering* method [51] runs FM on the netlist contracted from the top clustering in the hierarchy (with smallest k), then runs FM on the next highest level in the hierarchy, etc. and finishes with a final run on the original flattened netlist. This scheme allows FM to first make big moves, then gradually reduce the size of moves at each level of the hierarchy. The *edge unclustering* method only unclusters the modules incident to cut edges, allowing fine-grained optimization of these modules and coarse-grained optimization of the other modules to occur simultaneously. However, in practice, [94] prefer the iterative unclustering method.

Saab [160] draws an analogy with simulated annealing that clustering (and unclustering) should not proceed too quickly since it might freeze the optimization process in a suboptimal solution. He proposes the opposite of iterative unclustering, namely running FM first on the lowest level of the hierarchy, followed by clustering and then running FM on the next level. An interesting aspect of his approach is that the clusterings themselves are formed based on the order of moves in an FM run. The idea is that when one module is moved from C_1 to C_2, it tries to pull its incident modules into C_2 as well. Hence, a sequence of consecutive moves from C_1 to C_2 may determine a natural cluster. This observation was also made by Hagen et al. [86], who showed that a LIFO FM gain bucked organization encouraged the movement of entire clusters of modules (see Section 3.1.3).

Finally, we note an intriguing approach of Barnard and Simon [17] that applies clustering to spectral bisection. The second eigenvector of the contracted graph is computed and the eigenvector is interpolated to approximate the eigenvector for the original graph, and finally this vector is refined and split into a bipartitioning.

In summary, we have noted many distinct clustering constructions, and many ways in which clusterings can be used within a general partitioning approach. However, we do not seem close to understanding which approaches are better, and why. One reason is that it is generally difficult to compare clustering strategies – many works present a single scheme for use in their own specialized heuristic (e.g., FPGA partitioning [44], or TimberWolf placement [179]) without presenting any results for alternative clustering schemes. Another reason is that while it may be easy to devise clustering heuristics that optimize ratio cut, Absorption, etc., we do not know if these are the appropriate objectives for the specific application. For example, experiments conducted in [4] found that WINDOW clusterings could improve such traditional objectives, but led to only average-quality two-phase FM results. Indeed, it seems that almost any clustering heuristic will improve FM performance via the two-phase strategy, but no heuristic particularly distinguishes itself from the pack.

Since clustering is still a very poorly understood realm of bipartitioning, we believe that seeking answers to the following questions will be important in future work:

- What is the correlation between clustering objectives and the effect of clusterings within two-phase FM? Is there a natural clustering objective for the two-phase FM application?
- What is the best number of clusters? In a hierarchical approach, how many clusters should be at each level and how many levels are needed?
- What is the best way to integrate a clustering or a series of clusterings into an FM-based algorithm (e.g., given a CPU resource in terms of the number of FM passes made, how should this resource be allocated)?
- What cluster size constraints should be imposed? Is it better to have balanced clusters (as advocated by [30]), or to have large clusters mixed with small ones (as in edge unclustering)?

We believe that once the answers to these questions are understood, the best clustering strategies will become evident.

7 Conclusions

The past several years have seen the field of netlist partitioning make numerous advances – in terms of new formulations and objectives for various specific applications, and in terms of new algorithmic approaches. This paper has surveyed these developments, with an emphasis on more recent ideas. Throughout our discussion, we have attempted to note the research directions that merit further investigation, but have also tried to avoid value judgements based on experimental results. In this concluding section, we offer a summary of the benchmarking techniques for graphs and circuits that have appeared in the literature; we end with a brief assessment of the more promising approaches seen in recent works, and possible future directions for the field.

7.1 Benchmarking

Many studies of partitioning are not specific to VLSI applications, or otherwise did not have any set of "real circuits" available for benchmarking. Thus, experiments have been conducted on various classes of random graphs, including:

- **Uniform:** A graph $G(n, p)$ has n vertices, with the edge between each pair of vertices independently present with probability p. Random graphs (and hypergraphs) have been used by e.g., [110] [187] [164]. See [26] for a review of graph-theoretic results established for this model.

- **Geometric:** The graph $U(n, d)$ [110] is generated by picking n random (x, y) coordinate pairs (corresponding to the n vertices) in the unit square, and introducing an edge between two vertices if the Euclidean distance between their corresponding (x, y) locations is no greater than d.

- **"Difficult" d-Regular:** A graph is in the class $G_{Bui}(n, b, d)$ if it has n vertices, is d-regular, and has expected optimum bisection width no greater than b. A construction was first given by Bui et al. [29], and a simpler variant construction was given in [83].

- **Bisection Specific:** A graph in the class of random graphs $G_{2set}(2n, p_1, p_2, bis)$ [30] has a given edge internal to n-vertex clusters C_1 and C_2 with probabilities p_1 and p_2, respectively. Exactly bis edges connect C_1 with C_2, with bis very likely to be the cost of the optimum bisection.

- **k-Way Specific:** The class $G_{Gar}(k, \frac{n}{k}, p_i, p_e)$ [69] represents the multi-way extension of the Bisection Specific construction. A graph in this class has k clusters, each of size $\frac{n}{k}$; the $C(k, 2) \cdot (\frac{n}{k})^2$ intracluster edges are each independently present with probability p_i, and the $k \cdot C(\frac{n}{k}, 2)$ intercluster edges are each independently present with probability p_e. See also such works as [1].

Other, more complex random constructions have also been proposed, e.g., [124] [161].

In recent years, works on netlist partitioning have begun using benchmark suites collected by ACM SIGDA for experimental comparisons.[14] Recently, the CAD Benchmarking Laboratory (CBL) at North Carolina State University has succeeded the Microelectronics Center of North Carolina as the host for maintaining and renewing suites of benchmarks (WWW: http://www.cbl.ncsu.edu/www/CBL_Home.html/ or ftp to ftp.cbl.ncsu.edu or email to benchmarks@cbl.ncsu.edu). The benchmarks that most commonly appear in experiments in the literature belong to the LayoutSynthesis90 and 92, ISCAS85 and 89 [121], and PDWorkshop93 suites; for FPGA partitioning, the Partitioning93 suite is typically used. All of these benchmarks were originally designed for either placement or synthesis, hence each experimenter must translate the benchmark to some usable partitioning format. The differences in interpretations of the benchmarks can lead to discrepancies in the resulting inputs, and researchers may end up comparing the cuts of graphs and hypergraphs that are not the same. For example, the number of modules reported for the Test02-06 benchmarks is higher in [195] than in [82], which is likely due to the interpretation of I/O pads. In other cases, problems may arise from netlist connectivity. For example, when converting the industry3.vpnr benchmark into PROUD format, the authors of GORDIAN observed

[14] Graphs from the Harwell-Boeing sparse matrix collection (anonymous ftp to orion.cerfacs.fr) have also appeared occasionally in the literature (e.g, [160]) and provide an excellent set of test cases for graph partitioning algorithms.

that the netlist was disconnected, and the smaller 27-module component was discarded since it had no external connections and since GORDIAN requires connections to I/O pads in order to place the circuit [155]. Other authors might not remove this smaller component, leading to different bisection and ratio cut results for the same algorithm and the "same" test case.

We believe that the VLSI partitioning field would greatly benefit from a standard set of benchmarks exclusively designed for partitioning. As a first step, TU Munich has made many benchmarks available in PROUD format [183] (WWW: http://www.regent.e-technik.tu-muenchen.de/) and we are currently making benchmarks available in a standard hypergraph format (WWW: http://ballade.cs.ucla.edu:8080/). Our group is currently working with TU Munich to ensure consistency between our formats, making either format suitable for comparisons. Clearly, such entities as the CBL are best equipped to take such efforts a step further, e.g., by constructing and propagating self-consistent translators among the many existing partitioning input formats. Such institutionalized uniformity will help to make future benchmarking results in the literature much more meaningful. At the same time, care must also be taken not to constrain research paths by removing useful information. As examples, (i) logic function information must be retained in order to explore logic-based replication or combined resynthesis and partitioning (cf. [22]), and (ii) directional information must be present in inputs to address acyclic, delay-minimization, etc. formulations.

By way of facilitating future comparisons, Table 1 lists various partitioning formulations, along with works that have provided experimental results for some of the standard benchmarks. These are of historical note, since formulations can become popular simply because of the existence of comparison data. For example, Riess et al. [156] were the first to present 19 benchmark results for the 45/55-constrained bipartitioning objective (i.e., $L = 0.45n$ and $U = 0.55n$). The objective is of interest since ratio cut solutions tend to be highly unbalanced and hence impractical, yet exact bisections are too restrictive. This work has very quickly become a standard object of comparisons (cf. [6] [94] [190]).

Many of the formulations listed in the Table are quite similar, e.g., the first five formulations are all bipartitioning variants. While it is unclear which formulation is "correct", all of the associated bipartitioning algorithms are certainly comparable in some sense. It remains for the field to establish a reasoned basis by which heuristic variants, and associated experimental results, can be evaluated. For instance, Hauck and Borriello [94] recently completed an excellent study of FM-based bipartitioning algorithms, yet their results do not use unit area (rather, they set $w(v) = deg(v) - 2$ for each $v \in V$). Although this unorthodox weighting scheme by no means invalidates their conclusions, it does complicate future extensions of their study vis-a-vis other works. We believe that whether to use area information, which balance constraints apply, etc. are issues that can be addressed by identifying benchmarks (or benchmark variants) that are particularly suited for specific formulations.

Objective	Papers
Ratio Cut (unit area)	[187] [80] [188] [48] [51] [156]
Ratio Cut (module area)	[51] [31]
45/55 Bipartitioning	[156] [190] [94] [6]
Bisection (unit area)	[94]
Bisection (module area)	[188] [82] [51] [176] [195]
FPGA Devices	[125] [126] [44] [100] [38] [157]
Scaled Cost	[37] [3] [5] [4] [6]
Cluster Ratio	[3] [193]
Multi-Way + slack (module area)	[49] [194]
Multi-way + slack (unit area)	[79] [8]
Multi-way balanced	[11]
Min-Delay Clustering	[140] [45] [151] [191]
Absorption	[179] [4]
DS	[82] [4]
Density	[101]
Replication	[122] [104]
MCM/Quadrisection	[178] [159] [158]

Table 1: Various partitioning formulations and some works that provide corresponding experimental results.

Another weakness in the present "comparison-based literature" stems from having non-standard implementations of traditional algorithms, e.g., many researchers (including ourselves) have implemented their own FM-based 2-way or multi-way partitioners. On the other hand, Section 3 observed that subtle differences in implementation can have very large effects on solution quality. Widespread availability of canonical, well-tested implementations of standard algorithms (e.g., through CBL) could alleviate this problem. Also, researchers could perhaps use a shared site to make their source codes and/or executables available, to facilitate future comparisons as more benchmarks become available.[15] Authors of future work might also provide detailed descriptions of their experimental protocols and benchmark instances, since so many discrepancies are possible.

Finally, there are many deeper questions associated with the use of benchmarks in the VLSI partitioning literature (cf. panel discussions at many leading conferences in recent years). The beginnings of a list might include:

- Is it correct, as has been the case in recent years, to equate "progress" with "beating" previous results in terms of both solution quality and run-time? Arguably, this heavily biases against

[15]There are instructive precedents in the fields of combinatorial optimization and operations research. For example, WWW http://netlib.att.com provides a repository for mathematical software and related items, including eigenvector computations, mathematical programming and network flow packages, etc. Similarly, the OR-Library at the Imperial College Management School (WWW http://mscmga.ms.ic.ac.uk/info.html) gives a collection of test data sets for a variety of combinatorial formulations including LP, IP, QAP and matching (but not yet partitioning). Another type of impetus comes from regular "challenges" within the field (cf. the ACM Physical Design Workshop placement challenges from 1987-1991, the recent DIMACS Challenges in combinatorial optimization, etc.).

such novel approaches as evolutionary computing, parallel search, parallel annealing, constraint satisfaction, etc. Experimental setup, data collection and reporting methodology, etc. will also affect the perception of superiority. Furthermore, the present trend may well be responsible for the recent proliferation of "new formulations" for which there are no previous results that must be "beaten".

- Is it wise for the field to focus, as it has in recent years, on results for "real-world", as opposed to "artificial", instances? Note that (i) the provenance and functionality of benchmarks are often "sanitized" away so that it is unclear whether the benchmarks are representative; (ii) optimal solutions are never known for real-world instances (see [87] for a partial workaround); and (iii) any benchmark suite (as opposed to a generator for a class of (random) instances) is limited in size, which can in turn limit the significance of comparisons.

- Should industry users of CAD algorithms be more forthcoming with real-world test cases? Note that the literature for, e.g., the Multi-Way FPGA Device formulation [44] [100] includes test cases that presently can be obtained only under a signed nondisclosure agreement.

7.2 Perspectives

We conclude our work by listing what we consider to be promising directions for future research.

First, we believe that move-based approaches such as iterative improvement, stochastic hill-climbing, or evolutionary optimization are increasingly attractive. Certainly, FM and its derivatives have comprised the partitioning state-of-the-art throughout all areas of VLSI CAD during the past decade. However, small differences in implementations can lead to large differences in solution quality.[16] It seems worthwhile to continue to explore various implementations and tie-breaking strategies, especially given the large payoffs that can result from only minor code changes.

Second, we have noted that in practice, move-based approaches are often run many times with different *random* starting solutions. However, adapting the starting point based on knowledge from prior iterations can more quickly lead to stable solution quality; such an "adaptive multi-start" approach shares characteristics with hybrid genetic-local search techniques in evolutionary optimization. The paradigm of evolutionary optimization, with its unique style of search in the solution space, has claimed successes in the arenas of operations research and combinatorial optimization, which bodes well for future application to VLSI partitioning.

Third, spectral approaches that utilize multiple eigenvectors appear promising. Previous spectral

[16] Such studies can call into question the conclusions drawn in oft-cited works, in this case those of [123] [164] [189] regarding "lookahead" in iterative improvement strategies.

approaches that associate a single eigenvector with a cluster [18] or use only one eigenvector [80] may have inherent limitations. Works such as [68] [6] achieve solutions by associating a partitioning instance with the vector space comprised by multiple eigenvectors, yet heuristics for this representation are largely undeveloped. In addition, older spectral ideas may be worth revisiting, e.g., Donath and Hoffman [59] suggested varying the degree matrix to obtain better eigenvectors.

Fourth, finding good 1-dimensional circuit representations, (i.e., linear orderings of the modules) also seems promising. Dynamic programming can optimally split such an ordering into multiple clusters with respect to many standard objectives. Also, analytic techniques such as conjugate gradient method methods and successive over relaxation are naturally 1-dimensional. Finally, different methods for producing a partitioning from a linear ordering (such as the PARABOLI approach of gradually pulling apart an ordering while reoptimizing it) merit further investigation.

Fifth, we note that three trends – (i) the need to address highly constrained and complex formulations (e.g., timing, I/O, clock period, etc. constraints), (ii) the increased transfer of algorithm techniques from other well-established disciplines, and (iii) the availability of more computing power for CAD optimizations – combine to make combinatorial techniques increasingly viable. Analogous to the rediscovery of spectral and flow computations in recent years, we believe that approaches based on mathematical programming – including implicit enumeration methods such as branch-and-bound solution of ILP formulations – will be a "growth area" in VLSI partitioning.

Sixth, clustering now seems a required extension to many existing algorithms in light of increasing problem complexity. New criteria for clustering, as well as definitive studies correlating performance of the two-phase FM mechanism and the underlying clustering strategy, are still needed. Alternatives to the two-phase methodology are also well worth exploring. Works such as [94] provide important first steps in this direction.

Finally, we have noted that the recent literature contains many heuristic forays and new problem variations, which is not unexpected when many new works are essentially evaluated by the numbers they achieve for a limited suite of (outdated) test cases. In the previous subsection, we sketched a wish list for the future which includes: (i) canonical algorithm implementations (e.g., of the KL and FM heuristics) for comparison, and more widespread conformity with standard benchmark interchange formats, (ii) larger, public benchmarks with more functional and library information, (iii) public availability of partitioning codes, test instances, and actual partitioning solutions used in reported results, (iv) elimination of biases against nascent "future algorithm technologies" which may not yet be competitive in terms of both solution quality and efficiency, (v) more creative criteria for algorithms (cf. the "self-scaling" concept in [87]), and (vi) more complete descriptions of experimental protocols, along with more statistically meaningful data. These may be the most important future directions for researchers in VLSI partitioning, as we strive to achieve increased relevance to current design practice, and a more reasoned, long-term, and scientific approach to this intractable problem.

8 Acknowledgements

We thank Dennis Jen-Hsin Huang, Lars W. Hagen, Kenneth D. Boese and Bernhard M. Riess for their contributions to this review, and Albert Chung-Wen Tsao and Sudhakar Muddu for drawing figures. We also thank Martin D. F. Wong, Chung-Kuan Cheng, Pak K. Chan, Anthony Vannelli, Youssef Saab, and many others who made early versions of their works available to us. Finally, we thank James Hwang, Jon Frankle, Patrick Ciarlet and the anonymous reviewers for their helpful comments.

References

[1] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer, 1987.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[3] C. J. Alpert and A. B. Kahng. Geometric embeddings for faster and better multi-way netlist partitioning. In *Proc. ACM/IEEE Design Automation Conf.*, pages 743–748, 1993.

[4] C. J. Alpert and A. B. Kahng. A general framework for vertex orderings, with applications to netlist clustering. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 63–67, 1994.

[5] C. J. Alpert and A. B. Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. In *Proc. ACM/IEEE Design Automation Conf.*, pages 652–657, 1994.

[6] C. J. Alpert and S. Z. Yao. Spectral partitioning: the more eigenvectors, the better. In *Proc. ACM/IEEE Design Automation Conf. (to appear)*, 1995.

[7] A. A. Andreatta and C. C Ribeiro. A graph partitioning heuristic for the parallel pseudo-exhaustive logical test of vlsi combinational circuits. *Annals of Operations Research*, 50:1–36, 1994.

[8] S. Areibi and A. Vannelli. Advanced search techniques for circuit partitioning. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 77–98, 1993.

[9] S. Areibi and A. Vannelli. Circuit partitioning using a tabu search approach. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 1643–1646, 1993.

[10] S. Areibi and A. Vannelli. A combined eigenvector tabu search approach for circuit partitioning. In *Custom Integrated Circuits Conf.*, pages 9.7.1–9.7.4, 1993.

[11] S. Areibi and A. Vannelli. An efficient solution to circuit partitioning using tabu search and genetic algorithms. In *6th Int Conference of Micro Electronics*, pages 70–74, Istanbul, 1994.

[12] K. S. Arun and V. B. Rao. Two-way graph partitioning by principal components. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 2877–2880, 1990.

[13] K. S. Arun and V. B. Rao. New heuristics and lower bounds for graph partitioning. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 1172–1175, 1991.

[14] B. Awerbuch and T. Leighton. Multicommodity flows: A survey of recent research. In *4th Intl. Symp. Algorithms and Computation*, pages 297–302, 1993.

[15] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley Pub. Co., 1990.

[16] C. F. Ball, P. V. Kraus, and D. A. Mlynski. Fuzzy partitioning applied to vlsi-floorplanning and placement. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 177–180, 1994.

[17] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. Technical Report RNR-92-033, NASA Ames, November 1992.

[18] E. R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal of Algebraic and Discrete Methods*, 3(4):541–550, 1982.

[19] E. R. Barnes, A. Vannelli, and J. Q. Walker. A new heuristic for partitioning the nodes of a graph. *SIAM J. Disc. Math.*, 1(3):299–305, 1988.

[20] J. J. Bartholdi and L. K. Platzman. Heuristics based on spacefilling curves for combinatorial problems in euclidean space. *Management Sciences*, 34(3):291–305, 1988.

[21] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. In D. Touretzky, editor, *Proc. Neural Information Processing Systems*, November 1988.

[22] M. Beardslee and A. Sangiovanni-Vincentelli. An algorithm for improving partitions of pin-limited multi-chip systems. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 378–385, 1993.

[23] C. Berge. *Graphs and Hypergraphs*. American Elsevier, New York, 1976.

[24] J. Blanks. Near-optimal placement using a quadratic objective function. In *Proc. ACM/IEEE Design Automation Conf.*, pages 609–615, 1985.

[25] K. Boese, A. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16:101–113, 1994.

[26] B. Bollobas. *Random Graphs*. Academic Press, 1985.

[27] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 280–285, 1987.

[28] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer, 1985.

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

[30] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the performance of the kernighan-lin and simulated annealing graph bisection algorithms. In *Proc. ACM/IEEE Design Automation Conf.*, pages 775–778, 1989.

[31] T. N. Bui and B. R. Moon. A fast and stable hybrid genetic algorithm for the ratio-cut partitioning problem on hypergraphs. In *Proc. ACM/IEEE Design Automation Conf.*, pages 664–669, 1994.

[32] T. N. Bui and B. R. Moon. A genetic algorithm for a special class of the quadratic assignment problem. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 16, pages 99–116, 1994.

[33] T. Bultan and C. Aykanat. Circuit partitioning using parallel mean field annealing algorithms. In *IEEE Symp. Parallel and Distributed Processing*, pages 534–541, 1991.

[34] R. E. Burkard and T. Bonniger. A heuristic for quadratic boolean programs with applications to quadratic assignment. *European Journal of Operational Research*, 13:372–386, 1983.

[35] R. L. Cannon, J. V. Dave, and J. C. Bezdek. Efficient implementation of the fuzzy c-means clustering algorithms. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(2), March 1986.

[36] R. C. Carden and C.-K. Cheng. A global router using an efficient approximate multicommodity multi-terminal flow algorithm. In *Proc. ACM/IEEE Design Automation Conf.*, pages 316–321, 1991.

[37] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Trans. Computer-Aided Design*, 13(8):1088–1096, 1994.

[38] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. Spectral-based multi-way fpga partitioning. In *Proc. ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, pages 133–139, 1995.

[39] R. Chandrasekharam, S. Subhramanian, and S. Chaudhury. Genetic algorithm for node partitioning problem and applications in vlsi design. *IEE Proceedings E (Computers and Digital Techniques)*, 140(5):255–60, September 1993.

[40] H. R. Charney and D. L. Plato. Efficient partitioning of components. In *Proc. Design Automation Workshop*, pages 16–0 – 16–21, 1968.

[41] A. C. Chatterjee and R. Hartley. A new simultaneous circuit partitioning and chip placement approach based on simulated annealing. In *Proc. ACM/IEEE Design Automation Conf.*, pages 36–39, 1990.

[42] Y.-P. Chen, T.-C. Wang, and D. F. Wong. A graph partitioning problem for multiple-chip design. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 1778–1781, 1993.

[43] C. K. Cheng. The optimal partitioning of networks. *Networks*, 22:297–315, 1992.

[44] N.-C. Chou, L.-T. Liu, C.-K. Cheng, W.-J. Dai, and R. Lindelof. Circuit partitioning for huge logic emulation systems. In *Proc. ACM/IEEE Design Automation Conf.*, pages 244–249, 1994.

[45] J. Cong and Y. Ding. An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 48–53, 1992.

[46] J. Cong and Y. Ding. On area/depth trade-off in lut-based fpga technology mapping. In *Proc. ACM/IEEE Design Automation Conf.*, pages 213–218, 1993.

[47] J. Cong, L. Hagen, and A. B. Kahng. Random walks for circuit clustering. In *Proc. IEEE Intl. ASIC Conf.*, pages 14.2.1–14.2.4, 1991.

[48] J. Cong, L. Hagen, and A. B. Kahng. Net partitions yield better module partitions. In *Proc. ACM/IEEE Design Automation Conf.*, pages 47–52, 1992.

[49] J. Cong, W. Labio, and N. Shivakumar. Multi-way vlsi circuit partitioning based on dual net representation. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 56–62, 1994.

[50] J. Cong, Z. Li, and R. Bagrodia. Acyclic multi-way partitioning of boolean networks. In *Proc. ACM/IEEE Design Automation Conf.*, pages 670–675, 1994.

[51] J. Cong and M'L. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *Proc. ACM/IEEE Design Automation Conf.*, pages 755–760, 1993.

[52] A. Dasdan and C. Aykanat. Improved multiple-way circuit partitioning algorithms. In *Proc. ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, 1994.

[53] H. L. Davidson and E. Kelly, July 1993. Personal communication.

[54] D. E. Van den Bout and T. K. Miller III. Graph partitioning using annealed neural networks. *IEEE Transaction on Neural Networks*, 1(2):192–203, 1990.

[55] S. Dey, F. Brglez, and G. Kedem. Corolla based circuit partitioning and resynthesis. In *Proc. ACM/IEEE Design Automation Conf.*, pages 607–612, 1990.

[56] S. Dey, F. Brglez, and G. Kedem. Partitioning sequential circuits for logic optimization. In *Proc. IEEE Intl. Conf. Computer Design*, pages 70–76, 1990.

[57] W. E. Donath. Placement and average interconnection lengths of computer logic. *IEEE Trans. Circuits and Systems*, CAS-26(4):272–277, 1979.

[58] W. E. Donath. Logic partitioning. In B. Preas and M. Lorenzetti, editors, *Physical Design Automation of VLSI Systems*, pages 65–86. Benjamin/Cummings, 1988.

[59] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Develop.*, 17(5):420–425, 1973.

[60] A. E. Dunlop and B. W. Kernighan. A procedure for layout of standard-cell vlsi circuits. *IEEE Trans. Computer-Aided Design*, 4(1):92–98, 1985.

[61] S. Dutt. New faster kernighan-lin-type graph-partitioning algorithms. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 370–377, 1993.

[62] J. Fan, B. Catanzaro, C. K. Cheng, and S. H. Lee. Partitioning of opto-electronic multichip modules. In *Proc. IEEE Multi-Chip Module Conf.*, pages 138–143, 1994.

[63] T. A. Feo and M. Khellaf. A class of bounded approximation algorithms for graph partitioning. *Networks*, 20:181–195, 1990.

[64] M. Feuer. Connectivity of random logic. *IEEE Trans. Computers*, C-31(1):29–33, January 1982.

[65] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. ACM/IEEE Design Automation Conf.*, pages 175–181, 1982.

[66] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[67] J. Frankle. *Circuit Placement Methods Using Multiple Eigenvectors and Linear Probe Techniques.* PhD thesis, UC Berkeley, 1987.

[68] J. Frankle and R. M. Karp. Circuit placement and cost bounds by eigenvector decomposition. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 414–417, 1986.

[69] J. Garbers, H. J. Promel, and A. Steger. Finding clusters in vlsi circuits. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 520–523, 1990.

[70] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[71] C. H. Gebotys and M. I. Elmasry. *Optimal VLSI Architectural Synthesis: Area, Performance, and Testability.* Kluwer, 1992.

[72] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[73] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

[74] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1:190–206, 1989.

[75] A. V. Goldberg, É. Tardos, and R. E. Tarjan. Network flow algorithms. Technical Report STAN-CS-89-1252, Stanford University CS Dept., 1989.

[76] M. K. Goldberg and M. Burstein. Heuristic improvement technique for bisection of vlsi networks. In *Proc. IEEE Intl. Conf. Computer Design*, pages 122–125, 1983.

[77] R. Gomory and T. C. Hu. Multi-terminal network flows. *J. SIAM*, 9:551–570, 1961.

[78] J. W. Greene and K. J. Supowit. Simulated annealing without rejected moves. In *Proc. IEEE Intl. Conf. Computer Design*, pages 658–663, 1984.

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

[80] L. Hagen and A. B. Kahng. Fast spectral methods for ratio cut partitioning and clustering. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 10–13, 1991.

[81] L. Hagen and A. B. Kahng. Improving the quadratic objective function in module placement. In *Proc. IEEE Intl. ASIC Conf.*, pages 42–45, 1992.

[82] L. Hagen and A. B. Kahng. A new approach to effective circuit clustering. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 422–427, 1992.

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

[84] L. Hagen and A. B. Kahng. Combining problem reduction and adaptive multi-start: A new technique for superior iterative partitioning. *IEEE Trans. Computer-Aided Design (to appear)*, 1993.

[85] L. Hagen, A. B. Kahng, F. J. Kurdahi, and C. Ramachandran. On the intrinsic rent parameter and spectra-based partitioning methodologies. *IEEE Trans. Computer-Aided Design*, 13(1):27–37, 1994.

[86] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Trans. Computer-Aided Design (submitted)*, 1995.

[87] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng. Quantified suboptimality of vlsi layout heuristics. *IEEE Trans. Computer-Aided Design (to appear)*, 1995.

[88] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[89] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:219–229, 1970.

[90] T. Hamada, C.-K. Cheng, and P. M. Chau. A wire length estimation technique utilizing neighborhood density equations. In *Proc. ACM/IEEE Design Automation Conf.*, pages 57–61, 1992.

[91] M. Hanan, P. K. Wolff, and B. J. Agule. A study of placement techniques. *J. Design Automation and Fault-Tolerant Computing*, 2:28–61, 1978.

[92] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proc. ACM/SIAM Symp. Discrete Algorithms*, pages 165–174, Orlando, January 1992.

[93] N. Hasan and C. L. Liu. Minimum fault coverage in reconfigurable arrays. In *IEEE Intl. Symp. on Fault-Tolerant Computing Systems*, pages 248–353, 1988.

[94] S. Hauck and G. Borriello. An evaluation of bipartitioning techniques. In *Proc. Chapel Hill Conf. on Adv. Research in VLSI (to appear)*, 1995.

[95] S. Hauck and G. Borriello. Logic partition orderings for multi-fpga systems. In *Proc. ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, pages 32–38, 1995.

[96] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. Technical report, Sandia National Laboratories, September 1992.

[97] A. G. Hoffman. The dynamic locking heuristic – a new graph partitioning algorithm. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 173–176, 1994.

[98] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[99] T. C. Hu and K. Moerder. Multiterminal flows in a hypergraph. In T. C. Hu and E. Kuh, editors, *VLSI Circuit Layout: Theory and Design*, pages 87–93. IEEE Press, 1985.

[100] D. J.-H. Huang and A. B. Kahng. Multi-way system partitioning into single or multiple type fpgas. In *Proc. ACM/SIGDA International Workshop on Field-Programmable Gate Arrays*, pages 140–145, 1995.

[101] D. J.-H. Huang and A. B. Kahng. When clusters meet partitions: New density-based methods for circuit decomposition. In *Proc. European Design and Test Conf. (to appear)*, March 1995.

[102] M. Hulin. Circuit partitioning with genetic algorithms using a coding scheme to preserve the structure of the circuit. In *Parallel Problem Solving from Nature*, pages 75–79. Springer-Verlag, 1990.

[103] J. Hwang and A. El Gamal. Optimal replication for min-cut partitioning. In *ICCAD*, pages 432–435, 1992.

[104] J. Hwang and A. El Gamal. Optimal replication for min-cut partitioning. *IEEE Trans. Computer-Aided Design*, 14(1):96–106, 1995.

[105] T.-T. Hwang, R. M. Owens, and M. J. Irwin. Exploiting communication complexity for multilevel logic synthesis. *IEEE Trans. Computer-Aided Design*, 9(10):1017–1027, October 1990.

[106] E. Ihler, D. Wagner, and F. Wagner. Modeling hypergraphs by graphs with the same mincut properties. *Information Processing Letters*, 45(4):171–175, March 1993.

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

[108] H. Inayoshi and B. Manderick. The weighted graph bi-partitioning problem: A look at ga performance. In *Parallel Problem Solving from Nature*, pages 617–625. Springer-Verlag, 1992.

[109] D. S. Johnson. Local optimization and the traveling salesman problem. In *Proc. of the 17th Intl. Colloquium on Automata, Languages and Programming*, pages 446–460, July 1990.

[110] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation part i, graph partitioning. *Operations Research*, 37:865–892, 1989.

[111] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[112] A. B. Kahng. Fast hypergraph partition. In *Proc. ACM/IEEE Design Automation Conf.*, pages 762–766, 1989.

[113] Y. Kamidoi, S. Wakabayashi, J. Miyao, and N. Yoshida. A fast heuristic for hypergraph bisection. In *Proc. IEEE Intl. Symp. Circuits and Systems*, volume 2, pages 1160–1163, 1991.

[114] D. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proc. ACM/SIAM Symp. Discrete Algorithms*, pages 21–30, 1993.

[115] H. Karloff. *Linear programming.* Birkhauser, 1991.

[116] S. Kauffman and S. Levin. Toward a general theory of adaptive walks on rugged landscapes. *J. Theoretical Biology*, 128:11–45, 1987.

[117] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2):291–307, 1970.

[118] S. Khan and V. Madisetti. Yield-based system partitioning strategies for mcm and sem design. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 144–149, 1994.

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

[120] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. Computer-Aided Design*, 10(3):356–365, 1991.

[121] K. Kozmiński. Benchmarks of layout synthesis – evolution and current status. In *Proc. ACM/IEEE Design Automation Conf.*, pages 265–270, 1991.

[122] C. Kring and A. R. Newton. A cell-replicating approach to mincut-based circuit partitioning. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 2–5, 1991.

[123] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Trans. Computers*, 33(5):438–446, 1984.

[124] B. Krishnamurthy. Constructing test cases for partitioning heuristics. *IEEE Trans. Computers*, C-36(9):1112–1114, September 1987.

[125] R. Kužnar, F. Brglez, and K. Kozminski. Cost minimization of partitions into multiple devices. In *Proc. ACM/IEEE Design Automation Conf.*, pages 315–320, 1993.

[126] R. Kužnar, F. Brglez, and B. Zajc. Multi-way netlist partitioning into heterogeneous fpgas and minimization of total device cost and interconnect. In *Proc. ACM/IEEE Design Automation Conf.*, pages 238–243, 1994.

[127] P. J. M. Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications.* D.Reidel, Boston, 1987.

[128] B. Landman and R. Russo. On a pin versus block relationship for partition of logic graphs. *IEEE Trans. Computers*, C-20:1469, 1971.

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

[130] E. L. Lawler, K. N. Levitt, and J. Turner. Module clustering to minimize delay in digital networks. *IEEE Trans. Computers*, 18:47–57, 1969.

[131] T. Leighton, F. Makedon, and S. Tragoudas. Approximation algorithms for VLSI partition problems. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 2865–2868, 1990.

[132] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 422–431, 1988.

[133] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley-Teubner, 1990.

[134] A. Lim and Y.-M. Chee. Graph partitioning using tabu search. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 1164–7, 1991.

[135] L. T. Liu, M. T. Kuo, C. K. Cheng, and T. C. Hu. A replication cut for two-way partitioning. *IEEE Trans. Computer-Aided Design (to appear)*, 1995.

[136] L. T. Liu, M. Shih, and C.-K. Cheng. Data flow partitioning for clock period and latency minimization. In *Proc. ACM/IEEE Design Automation Conf.*, pages 658–663, 1994.

[137] L. T. Liu, M. Shih, N.-C. Chou, C.-K. Cheng, and W. Ku. Performance driven partitioning using retiming and replication. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 296–299, 1993.

[138] D. W. Matula and F. Shahrokhi. The maximum concurrent flow problem and sparsest cuts. Technical report, Southern Methodist Univ., March 1986.

[139] B. Mohar. The laplacian spectrum of graphs. In Y. Alavi and et al., editors, *Graph Theory, Combinatorics, and Applications*, pages 871–898. Wiley, 1991.

[140] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. On clustering for minimum delay/area. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 6–9, 1991.

[141] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *Siam Journal of Discrete Mathematics*, 5(1):54–66, 1992.

[142] A. R. Newton, April 1991. Personal communication.

[143] T.-K. Ng, J. Oldfield, and V. Pitchumani. Improvements of a mincut partition algorithm. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 479–473, 1987.

[144] C.-I. Park and Y.-B. Park. An efficient algorithm for vlsi network partitioning problem using a cost function with balancing factor. *IEEE Trans. Computer-Aided Design*, 12(11):1686–1694, 1993.

[145] C. Peterson and J. R. Anderson. Neural networks and np-complete optimization problems; a performance study on the graph bisection problem. *Complex Systems*, 2(1):59–89, February 1988.

[146] L. T. Pillage and R. A. Rohrer. A quadratic metric with a simple solution scheme for initial placement. In *Proc. ACM/IEEE Design Automation Conf.*, pages 324–329, 1988.

[147] S. Pissanetsky. *Sparse Matrix Technology.* Academic Press, 1984.

[148] A. Pothen, H. D. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Analysis and its Applications*, 11:430–452, 1990.

[149] B. T. Preas and M. J. Lorenzetti, editors. *Physical Design Automation of VLSI Systems.* Benjamin/Cummings, 1988.

[150] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[151] R. Rajaraman and D. F. Wong. Optimal clustering for delay minimization. In *Proc. ACM/IEEE Design Automation Conf.*, pages 309–314, 1993.

[152] C. R. Rao. The use and interpretation of principal component analysis in applied research. *Sankhya Series A*, 26:329–358, 1964.

[153] M. Razaz. A fuzzy c-means clustering placement algorithm. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 2051–2054, 1993.

[154] F. Rendl and H. Wolkowicz. A projection technique for partitioning the nodes of a graph. Technical report, University of Waterloo, May 1994.

[155] B. M. Riess, February 1995. Personal communication.

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

[157] B. M. Riess, H. A. Giselbrecht, and B. Wurth. A new k-way partitioning approach for multiple types of fpgas. Technical Report TUM-LRE-95-2, Technical University of Munich, 1995.

[158] B. M. Riess and A. A. Schoene. Architecture driven k-way partitioning for multichip modules. In *Proc. European Design and Test Conf. (to appear)*, 1995.

[159] K. Roy and C. Sechen. A timing-driven n-way chip and multi-chip partitioner. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 240–247, 1993.

[160] Y. Saab. A fast and robust network bisection algorithm. *IEEE Trans. Computers (to appear)*, 1995.

[161] Y. Saab. New methods for construction of test cases for partitioning heuristics. *Progress in VLSI Design (to appear)*, 1996.

[162] Y. Saab and V. Rao. Fast effective heuristics for the graph bisectioning problem. *IEEE Trans. Computer-Aided Design*, 9(1):91–98, January 1990.

[163] Y. Saab and V. Rao. On the graph bisection problem. *IEEE Trans. Circuits and Systems*, 39(9):760–762, September 1992.

[164] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. Computers*, 38(1):62–81, January 1989.

[165] L. A. Sanchis. Multiple-way network partitioning with different cost functions. *IEEE Trans. Computers*, 42(22):1500–1504, 1993.

[166] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 743–751, 1991.

[167] G. Saucier, D. Brasen, and J. P. Hiol. Partitioning with cone structures. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 236–239, 1993.

[168] D. M. Schuler and E. G. Ulrich. Clustering and linear placement. In *Proc. ACM/IEEE Design Automation Conf.*, pages 50–56, 1972.

[169] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proc. ACM/IEEE Design Automation Conf.*, pages 57–62, 1972.

[170] C. Sechen and D. Chen. An improved objective function for mincut circuit partitioning. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 502–505, 1988.

[171] N. A. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, Boston, 1993.

[172] M. Shih, 1993. Personal communication.

[173] M. Shih and E. Kuh. Quadratic boolean programming for performance-driven system partitioning. In *Proc. ACM/IEEE Design Automation Conf.*, pages 761–765, 1993.

[174] M. Shih, E. Kuh, and R.-S. Tsay. Performance-driven system partitioning on multi-chip modules. In *Proc. ACM/IEEE Design Automation Conf.*, pages 53–56, 1992.

[175] M. Shih, E. Kuh, and R.-S. Tsay. Timing-driven system partitioning by constraints decoupling method. In *Proc. IEEE Multi-Chip Module Conf.*, pages 164–169, 1993.

[176] H. Shin and C. Kim. A simple yet effective technique for partitioning. *IEEE Trans. VLSI Systems*, 1(3), September 1993.

[177] G. Sigl, K. Doll, and F. M. Johannes. Analytical placement: A linear or a quadratic objective function? In *Proc. ACM/IEEE Design Automation Conf.*, pages 427–432, June 1991.

[178] P. R. Suaris and G. Kedem. An algorithm for quadrisection and its application to standard cell placement. *IEEE Trans. Circuits and Systems*, 35(3):294–303, 1988.

[179] W. Sun and C. Sechen. Efficient and effective placements for very large circuits. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 170–177, 1993.

[180] L. Tao, Y. C. Zhao, K. Thulasiraman, and M. N. S. Swamy. An efficient tabu search algorithm for graph bisectioning. In *Proc. Great Lakes Symp. VLSI*, pages 92–95, 1991.

[181] M. Toyonaga, S.-T. Yang, T. Akino, and I. Shirakawa. A new approach of fractal-dimension based module clustering for vlsi layout. In *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 185–188, 1994.

[182] S. Tragoudas. An improved algorithm for the generalized min-cut partitioning problem. In *Proc. Great Lakes Symp. VLSI*, pages 242–247, 1994.

[183] R.-S. Tsay and E. S. Kuh. A unified approach to partitioning and placement. *IEEE Trans. Circuits and Systems*, 38(5):521–533, 1991.

[184] A. Vannelli and S. W. Hadley. A gomory-hu cut tree representation of a netlist partitioning problem. *IEEE Trans. Circuits and Systems*, 37(9):1133–1139, 1990.

[185] A. Vannelli and G. S. Rowan. A constrained clustering approach for partitioning netlists. In *Proc. 28th Midwest Symposium on Circuits and Systems*, pages 211–215, 1985.

[186] G. Vijayan. Generalization of min-cut partitioning to tree structures and its applications. *IEEE Trans. Computers*, 40(3):307–314, 1991.

[187] Y.-C. Wei and C.-K. Cheng. Towards efficient hierarchical designs by ratio cut partitioning. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 298–301, 1989.

[188] Y.-C. A. Wei and C.-K. Cheng. An improved two-way partitioning algorithm with stable performance. *IEEE Trans. Computer-Aided Design*, 10(12):1502–1511, 1991.

[189] N.-S. Woo and J. Kim. An efficient method of partitioning circuits for multiple-fpga implementation. In *Proc. ACM/IEEE Design Automation Conf.*, pages 202–207, 1993.

[190] H. Yang and D. F. Wong. Efficient network flow based min-cut balanced partitioning. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 50–55, 1994.

[191] H. Yang and D. F. Wong. Circuit clustering for delay minimization under area and pin constraints. In *Proc. European Design and Test Conf. (to appear)*, 1995.

[192] H. Yang and D. F. Wong. Optimal wiring minimization for partitioned circuits with least replication. Manuscript, 1995.

[193] C.-W. Yeh, C.-K. Cheng, and T.-T. Y. Lin. A probabilistic multi-commodity flow solution to circuit clustering problems. In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pages 428–431, 1992. Also see *IEEE Trans. on CAD*, **14**:(2), pp. 154-162, 1995, for extended version.

[194] C.-W. Yeh, C.-K. Cheng, and T.-T. Y. Lin. A general purpose, multiple-way partitioning algorithm. *IEEE Trans. Computer-Aided Design*, 13(12):1480–1487, 1994.

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

[196] H. J. Zimmerman. *Fuzzy Set Theory and its Applications*. Kluwer Nlijhoff, 2nd edition, 1991.