- 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.

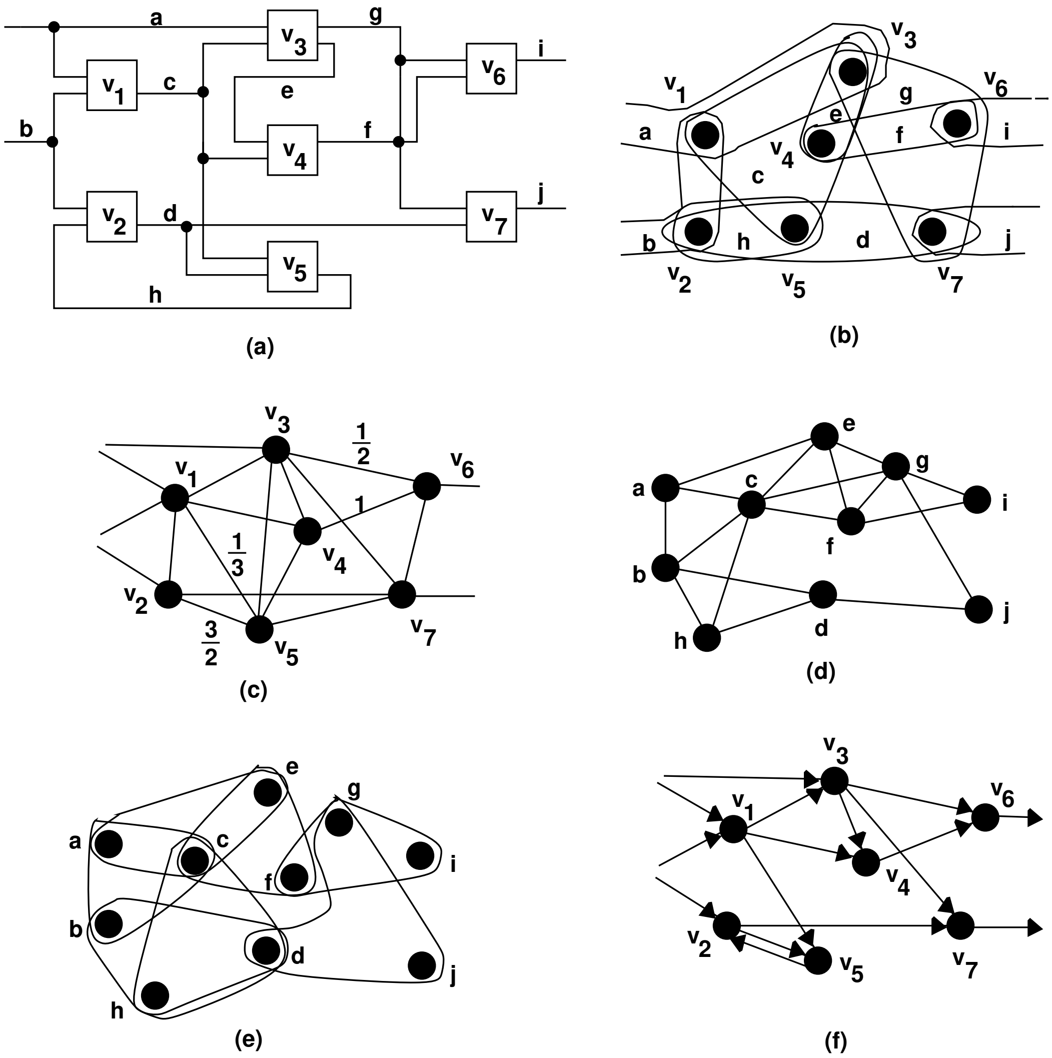

Figure 1: Representations of a circuit with 7 modules and 10 signal nets: (a) circuit diagram with all inputs on the left side of the modules and all outputs on the right side; (b) the hypergraph representation; (c) the weighted graph representation using the standard clique net model with uniform edge 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.

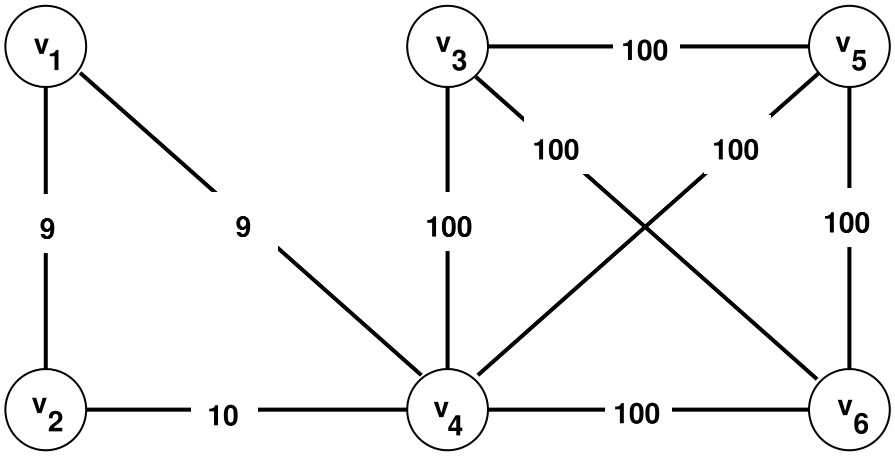

Figure 2: Optimal bipartitionings of an edge-weighted graph with six modules. The min-cut bipartitioning $\{\{v_1\}, \{v_2, v_3, v_4, v_5, v_6\}\}$ will have cutsize 18, but is very unbalanced. The min-cut bisection $\{\{v_1, v_2, v_4\}, \{v_3, v_5, v_6\}\}$ has cutsize 300, much worse than the arguably 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 $w(C) \le U$. 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 $w(C_h) \le U$ 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:

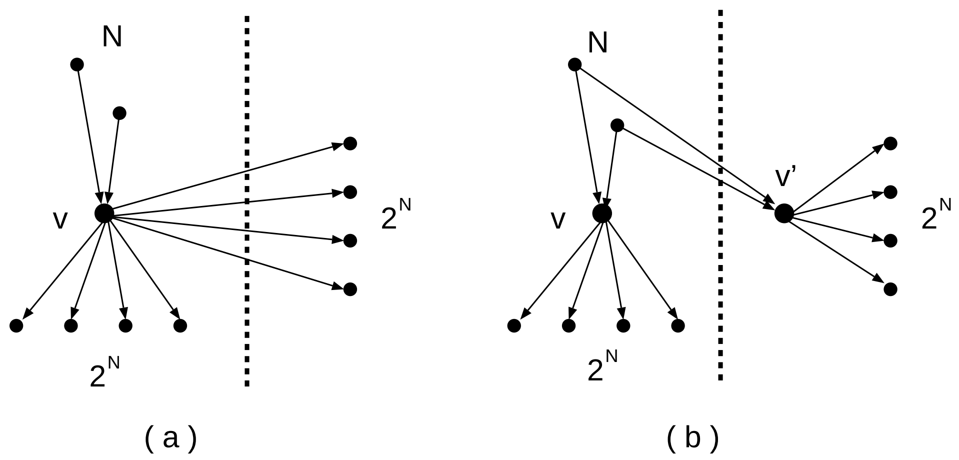

Figure 3: Directed graph corresponding to an N-to-2^N decoder v. Replication of v reduces the cut size from 2^N in (a) to N in (b).

- 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

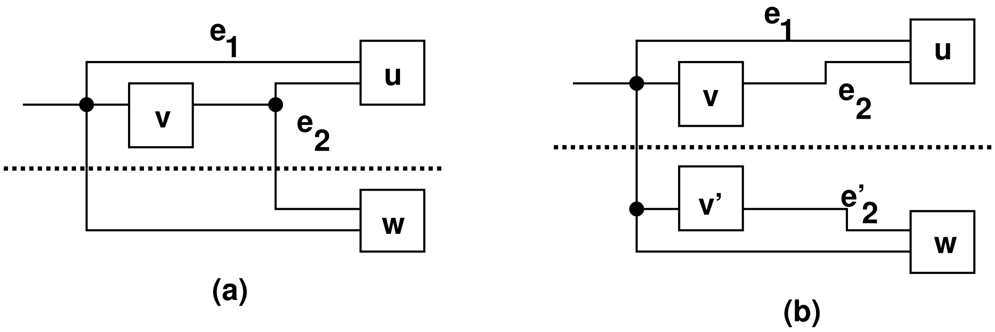

Figure 4: (a) A circuit with two nets crossing the cut, and (b) the same circuit with module v replicated, yielding only one cut net.

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.

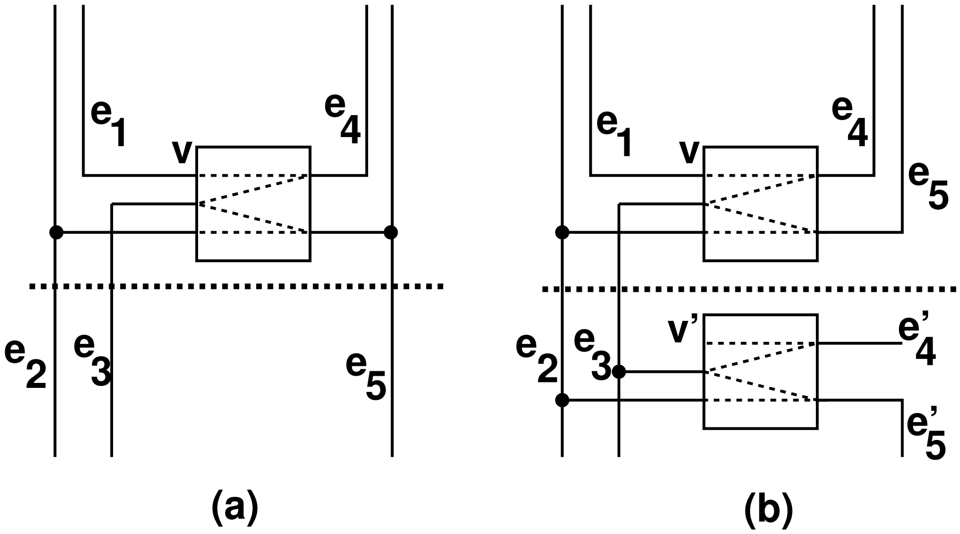

Figure 5: (a) A module v with three nets crossing the cut; (b) replication requires only two nets to cross the cut since the net corresponding to input e_1 is not needed by v'.