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

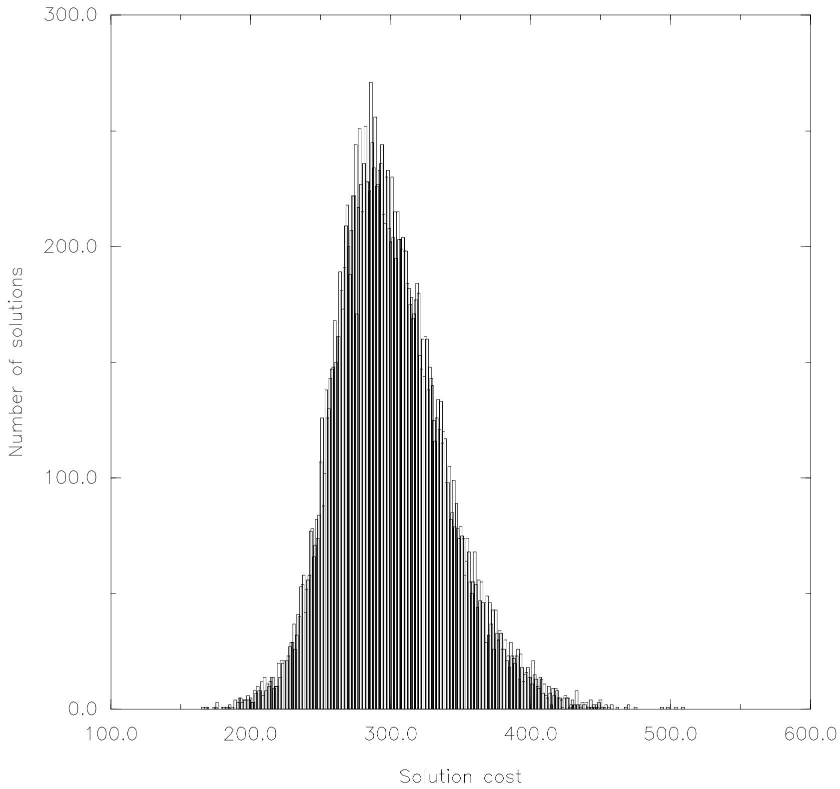

Figure 18: Distribution of 21,203 FM bisection solutions (i.e., local minima) for SIGDA Layout Synthesis benchmark Primary2 (3014 modules). Each solution was generated from a new random starting point.

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: