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 $w(v_i) + w(p_i(r)) \le U$, 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

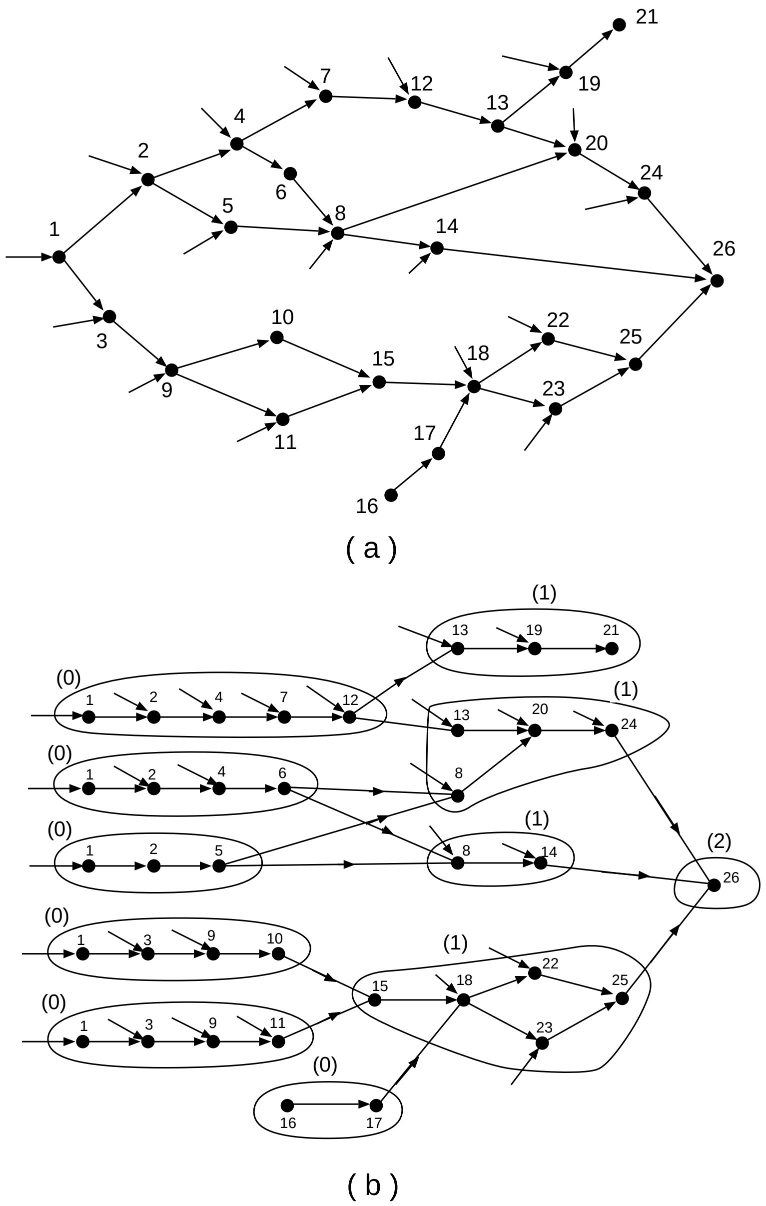

Figure 12: (a) An example DAG from [130] and (b) the clustering solution from Lawler et al.'s [130] labeling algorithm assuming unit module weight and cluster size bound $U = 5$.

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 $w(C_h) \le U, 1 \le h \le k$ and replace them with the pin constraints $|In(C_h)| \le K$ 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 $|In(C_h) \cup Out(C_h)| \le K + 1$ 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 $w(C_h) \le U$, $|E(C_h)| \le K + 1$. 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 $(v, w) \in E$, $f(v, w) \le c(v, w)$; 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 $|E(C)| \leq IO$. 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.

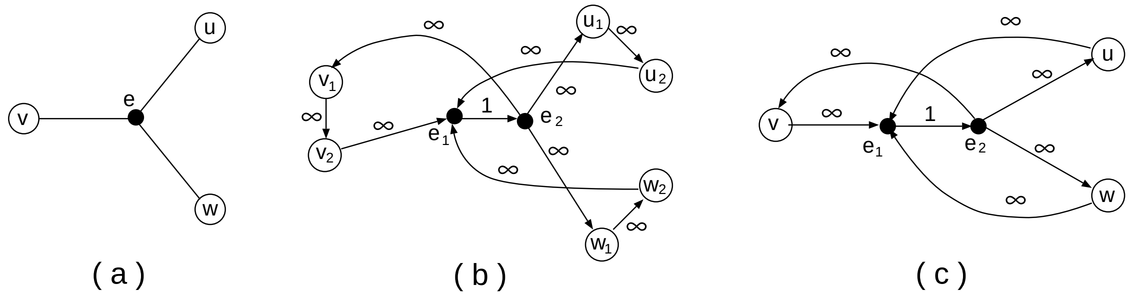

Figure 13: (a) The star representation for a 3-pin net, (b) the application of Lawler's transformation to (a), and (c) a more efficient representation of (b).

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.

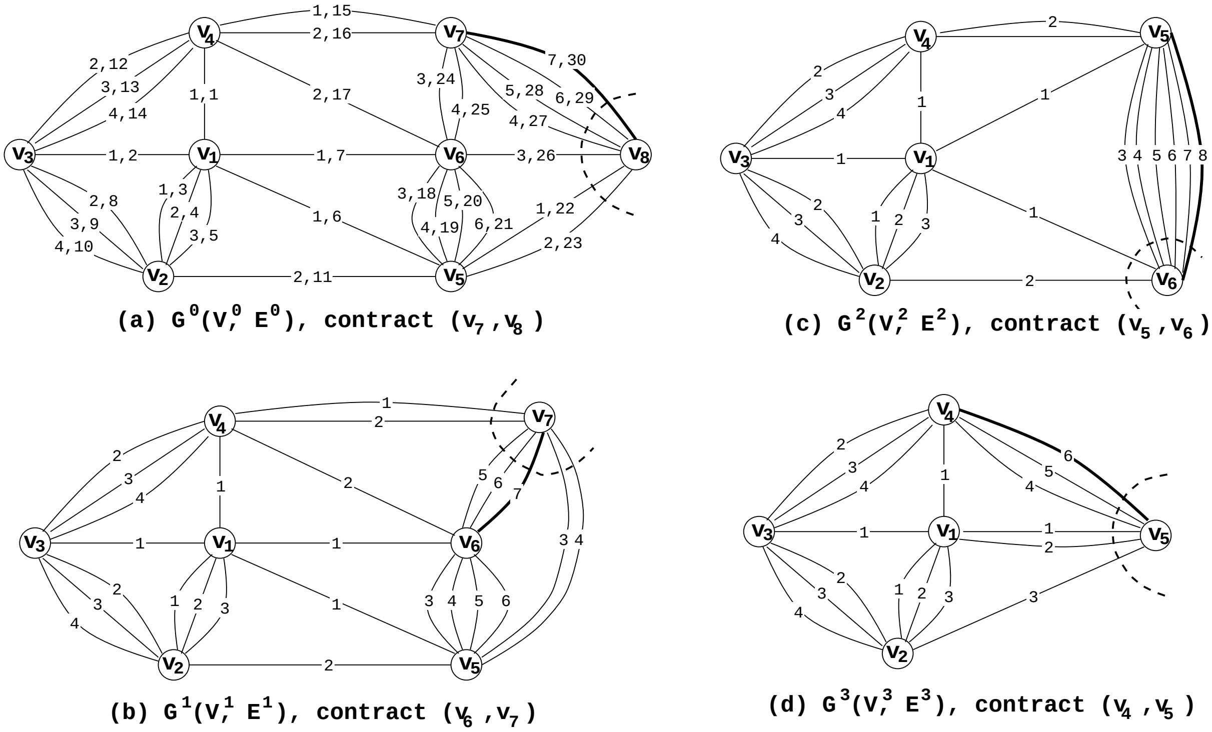

Figure 14: Example of the execution of the min-cut algorithm of [141].

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 $C_1 = \{s\}$).

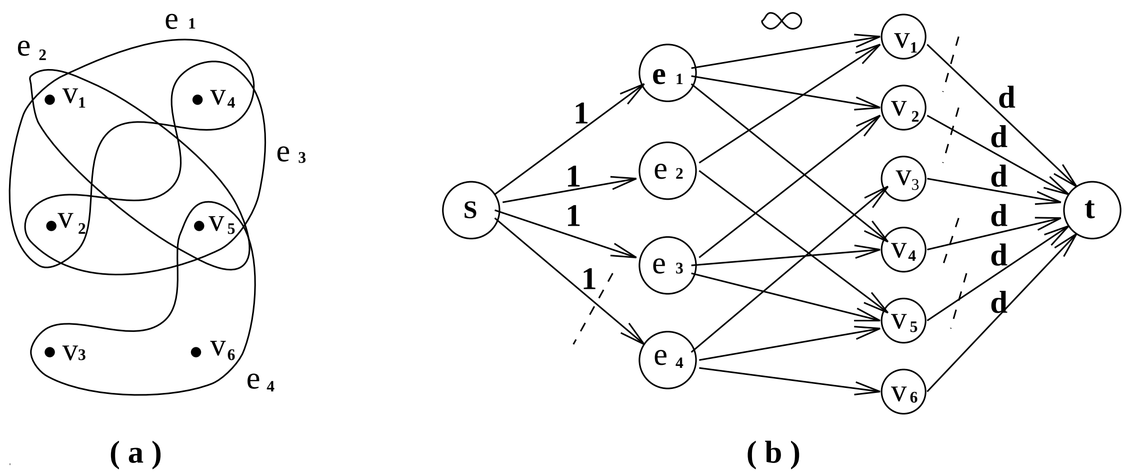

Figure 15: (a) An example of a hypergraph and (b) its density flow network assuming unit module weights.

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.)

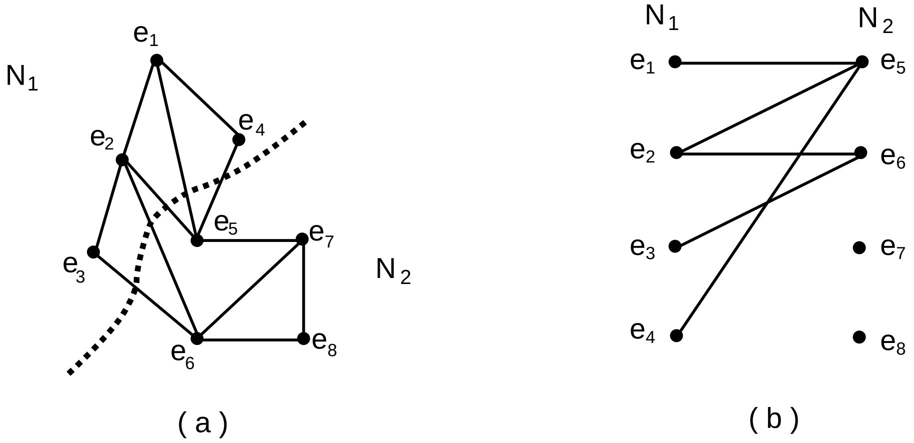

Figure 16: (a) An example of a net bipartitioning of the intersection graph and (b) the bipartite conflict graph it induces.

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,

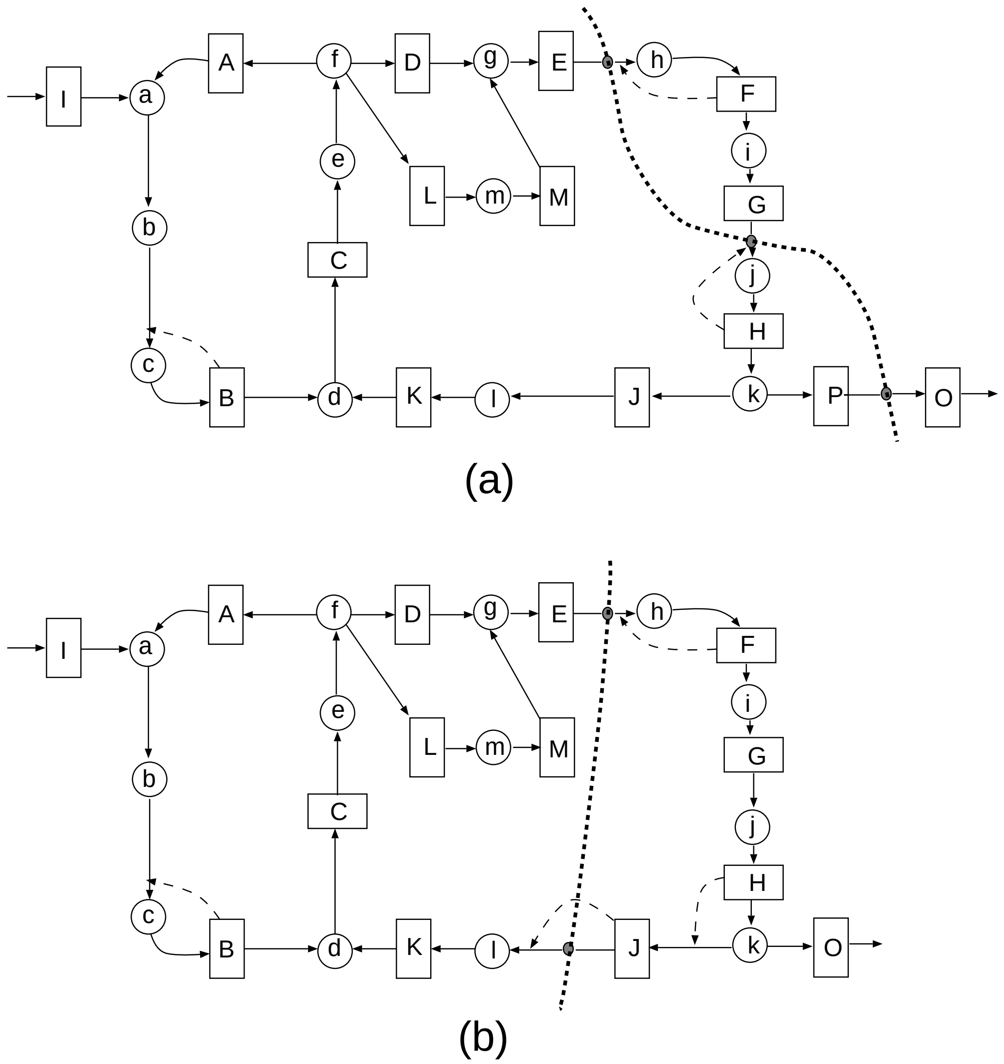

Figure 17: (a) A bipartitioning $T = 3$ before retiming, and $T = 2$ after retiming (with latency 18), and (b) a bipartitioning with $T = 2$ and 16 units latency after retiming.

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.