# Edge Separability Based Circuit Clustering with Application to Circuit Partitioning*

Jason Cong and Sung Kyu Lim
UCLA Department of Computer Science, Los Angeles, CA 90095
{cong,limsk}@cs.ucla.edu

**Abstract— In this paper, we introduce a new efficient $O(n \log n)$ graph search based bottom-up clustering algorithm named ESC (Edge Separability based Clustering). Unlike existing bottom-up algorithms that are based on local connectivity information of the netlist, ESC exploits more global connectivity information "edge separability" to guide clustering process while carefully monitoring cluster area balance. Computing the edge separability for a given edge $e = (x, y)$ in an edge weighted undirected graph $G(V, E, s, w)$ is equivalent to finding the $x$-$y$ mincut. Then, we show that a simple and efficient algorithm CAPFOREST [14] can be used to provide a good estimation of edge separability for all edges in $G$ without using any network flow computation. Related experiments based on large scale ISPD98 [1] benchmark circuits confirm that exploiting edge separability yields better quality partitioning solution compared to various bottom-up clustering algorithms proposed in the literature including Absorption [18], Density [9, 11], Rent Parameter [15], Ratio Cut [19], Closeness [17], and Connectivity [16] method. In addition, our ESC based multiway partitioning algorithm LR/ESC-PM provides comparable results to state-of-the-art hMetis [12] and hMetis-Kway [13].**

## I. Introduction

Due to the recent advances in VLSI technology, hierarchical design paradigm becomes essential in coping with today's high design complexity. Top-down partitioning is a widely adopted method to identify natural circuit hierarchy and thus enable divide-and-conquer design methodology. Since partitioning deals with the original complexity of the given circuit directly, many heuristics have been devised to solve the problem more effectively and efficiently. Among them, clustering has been an attractive method to reduce the problem size efficiently at the same time guide partitioning process successfully to obtain satisfactory results.

Clustering heuristics can be classified into bottom-up and top-down approaches. In bottom-up approaches, each cell initially belongs to its own cluster, and clusters are gradually grown into larger clusters from merging with others. Bottom-up approaches can be further divided into connectivity based [16, 15, 9, 17, 18, 11] and signal flow based methods [4, 6]. In spite of the efficiency from its simple nature, bottom-up approach inherently suffers from the limitation that the clustering process is based mostly on local information such as the local connectivity to neighboring nodes. This locality in clustering decision can lead to unnatural decomposition of the circuit. In top-down approaches [19], the given circuit is recursively partitioned into clusters. This method can exploit more global information from the circuit but usually at the cost of large computation time. A comprehensive survey of various clustering algorithms can be found in [3].

In this paper, we introduce a new efficient $O(n \log n)$ graph search based bottom-up clustering algorithm named `ESC` (Edge Separability based Clustering). Unlike existing bottom-up algorithms that are based on local connectivity information of the netlist, `ESC` exploits more global connectivity information *edge separability* to guide clustering process while carefully monitoring cluster area balance. Computing the edge separability for a given edge $e = (x, y)$ in an edge weighted undirected graph $G(V, E, s, w)$ is equivalent to finding the $x$-$y$ mincut. Then, we show that a simple and efficient algorithm `CAPFOREST` [14] can be used to provide a good estimation of edge separability for all edges in $G$ without using any network flow computation. Related experiments based on large scale ISPD98 [1] benchmark circuits confirm that exploiting edge separability yields better quality partitioning solution compared to various bottom-up clustering algorithms proposed in the literature including Absorption [18], Density [9, 11], Rent Parameter [15], Ratio Cut [19], Closeness [17], and Connectivity [16] method. In addition, our `ESC` based multiway partitioning algorithm `LR/ESC-PM` provides comparable results to state-of-the-art `hMetis` [12] and `hMetis-Kway` [13].

The remainder of the paper is organized as follows. Section II presents theoretical background, Section III presents Edge Separability based Clustering. Section IV provides experimental results. Section V concludes the paper with our ongoing research.

*This research is partially supported by the MARCO/DARPA Gigascale Silicon Research Center (GSRC) and Fujitsu Laboratories of America under the California MICRO Program.

## II. Preliminaries

In this section, we provide theoretical background on the general concept of edge separability, edge contractibility, and Maximum Adjacency (MA) vertex ordering for an edge weighted undirected graph.

### A. Edge Separability and Contractibility

The input netlist is transformed into edge weighted undirected graph $G(V, E, s, w)$ with vertex set $V$, edge set $E$, a positive real vertex size $s(x)$ for each $x \in V$, and a positive real edge weight $w(e)$ for each $e \in E$. Edge weights are computed by the standard clique method that assigns $1/(|k|-1)$ of weight to each edge in clique of size $|k|$, where $|k|$ is the number of cell net $k$ connects. Let $n = |V|$ and $m = |E|$. A non-empty subset $X \subset V$ defines a *cut*, and the *cutsize* of cut $X$, denoted by $c(X)$, is defined as the sum of weights of its outgoing edges. If cut $X$ consists of single vertex $x$, $c(x)$ is used instead of $c(\{x\})$ and alternatively call it *degree* of $x$. A cut $X$ is said to separate vertices $x$ and $y$ if $x \in X$ and $y \in V - X$, or $x \in V - X$ and $y \in X$. A cut $U$ that minimizes $c(U)$ in $G$ is called *minimum cut*, and cutsize of minimum cut $U$ is called *minimum cutsize* and denoted by $\lambda(G)$. The minimum degree of $G$, denoted by $\delta(G)$, is defined as $\delta(G) = \min\{c(x) \mid x \in V\}$.

For given edge $e = (x, y) \in E$, *edge separability* of $e$, denoted by $\lambda(e)$, is defined as the minimum cutsize among the cuts separating $x$ and $y$ in $G$. Then,

$$w(e) \leq \lambda(e) \leq \min\{c(x), c(y)\} \quad (1)$$

The *contraction* of edge $e$ is defined by merging $x$ with $y$, removing $e$ from $G$, replacing each edge of the form $(y, z)$ with $(x, z)$, and updating size of the resulting merged node still called $x$ with $s(x) = s(x) + s(y)$. If this process may create parallel edges, they are merged into a single edge whose weight is equal to the sum of weights of the parallel edges. The graph obtained by contracting all edges in subset $F \subseteq E$ is denoted $G' = G/F$. Then, an edge $e = (x, y) \in E$ is called *contractible* if and only if $\lambda(e) \geq \lambda(G)$. The intuition behind edge contractibility is that the two end vertices $x$ and $y$ of a contractible edge $e = (x, y)$ are guaranteed to be on the same side of some minimum cut $U$ so that $\lambda(G)$ is preserved in $G/e$ after the contraction of $e$.

### B. Tighter Lower Bound of Edge Separability

Computing the edge separability for a given edge $e = (x, y)$ in an edge weighted undirected graph $G = (V, E, s, w)$ is equivalent to finding the maximum flow between $x$ and $y$. Since the currently best known time bounds for solving the maximum flow problem is $O(mn \log(n^2/m))$ due to Goldberg and Tarjan [10], the computation of $\lambda(e)$ for all edges in $G$ requires $O(m^2 n \log(n^2/m))$ time. Thus, direct computation of edge separability for all edges in $G$ is extremely time-consuming, even for moderate size graphs with a few thousand vertices. Equation (1) indicates that $w(e)$ serves as a lower bound of $\lambda(e)$, but we found out that (i) there exists a better approximation of $\lambda(e)$, (ii) it requires only $O(m + n \log n)$ to compute the approximation of $\lambda(e)$ for *all* edges in $E$.

```
CAPFOREST(G)
Input: edge weighted undirected graph G = (V, E)
Output: contractible edge set Z(G) ⊂ E
1.  label all v ∈ V unvisited;
2.  label all e ∈ E unscanned;
3.  r(v) = 0 for all v ∈ V;
4.  while (there exists unvisited vertex)
5.      choose an unvisited vertex x with largest r(x);
6.      for (each y adjcnt to x by unscanned e)
7.          r(y) = r(y) + w(e);
8.          q(e) = r(y);
9.          mark e scanned;
10.     mark x visited;
11. endwhile
12. Z(G) = {e | e ∈ E, q(e) ≥ λ̄(G)}
```

Fig. 1. CAPFOREST algorithm [14] for computing contractible edge set $Z(G)$ in $O(m + n \log n)$ time. The vertices are visited in MA (Maximum Adjacency) ordering.

Recently, Nagamochi and Ibaraki presented a breakthrough algorithm MINCUT [14] to compute $\lambda(G)$ without using any network flow computation in $O(mn + n^2 \log n)$. In MINCUT, an $O(m + n \log n)$ algorithm CAPFOREST is repeatedly used to identify contractible edges. CAPFOREST computes some value $q(e)$ for each edge $e \in E$ to determine contractibility of $e$, and they prove that $q(e) \leq \lambda(e)$. CAPFOREST is based on traversing vertices of $G$ according to the *Maximum Adjacency (MA) ordering*. MA ordering chooses a vertex that is *most tightly connected* to the vertices that are already in the ordering. If set $A$ denotes the vertices that are already been ordered and $v$ is a candidate vertex, the degree of connection is measured by $c(A, v)$, i.e., the sum of weights of edges between $A$ and $v$. Then, CAPFOREST traverses vertices of $V$ in MA ordering while computing $q(e)$ for every edge in $G$.[1]

Initially, all edges are unscanned and all vertices are unvisited. CAPFOREST maintains variables $r(v)$ for each vertex $v$ and $q(e)$ for each edge $e$, where $r(v)$ is $c(A, v)$, i.e., the sum of the weights of the edges between $v$ and vertices already visited in MA ordering. $q(e)$ for $e = (x, y)$ is the value of $r(y)$ when $e$ is scanned from $x$. Then, CAPFOREST always chooses the unvisited vertex $v$ with maximum $r(v)$ and scans all its unscanned outgoing edges. The description of CAPFOREST is shown in Figure 1. From line 7 and

[1] Note that Alpert and Kahng [2] used MA ordering in their formulation of restricted partitioning, where a dynamic programming technique is used to construct a partitioning solution from the MA ordering.

8, we observed that $w(e) \leq q(e)$ since $r(v) \geq 0$ for all $v \in V$ for any positive edge weights. Then,

$$w(e) \leq q(e) \leq \lambda(e) \tag{2}$$

From Equation (1) and (2) we concluded that $q(e)$ serves as a better approximation to edge separability $\lambda(e)$ than edge weight $w(e)$.

The contractible edge set $Z(G)$ is computed by comparing $q(e)$ to $\bar{\lambda}(G)$ as shown in line 12, where $\bar{\lambda}$ denotes the minimum cutsize discovered so far. `CAPFOREST` does not require precomputation of $\lambda(G)$ in order to calculate contractible edge set $Z(G)$. Instead, the minimum degree $\delta(G)$ is used as a starting point to perform gradual update on $\lambda(G)$. `CAPFOREST` updates $\bar{\lambda} = \min\{\bar{\lambda}, c'(x)\}$ at the contraction of $e = (x, y)$, where $c'(x)$ denotes updated degree of vertex $x$.

## III. Edge Separability based Clustering

The `ESC` (Edge Separability based Clustering) algorithm is a bottom-up clustering algorithm, where clusters grow from the contraction of contractible edges. Since the clustering process is guided by $q(e)$, a better estimation of edge separability $\lambda(e)$ than edge weight $w(e)$ as shown in Equation (2), `ESC` exploits more global connectivity information. `ESC` algorithm builds two-level cluster hierarchy, where the bottome level corresponds to the original netlist. Then, partitioning is performed at the top and bottom level for global and local changes in the partitioning solution.

### A. Overview

`ESC` grow clusters from contraction of edges in the contractible edge set $Z(G) \subseteq E$ in a bottom-up fashion, which is equivalent to merging two clusters that have direct connection via a contractible edge. Each vertex belongs to its own cluster initially, and clusters grow from greedy merging based on $q(e)$, the estimated $\lambda(e)$. The following provides overall flow of `ESC` for given netlist $N$;

1. Transform $N$ into $G = (V, E, s, w)$ and let $\bar{\lambda} = \delta(G)$.
2. Run `CAPFOREST`$(G)$ and obtain $Z(G)$
3. Edges in $Z(G)$ are sorted into heap $H$ based on their *rank* (to be defined).
4. Remove edge $e$ from the top of $H$ and see if contraction of $e$ violates the cluster size constraint. If it satisfies the constraint, obtain $G/e$ and update $\bar{\lambda}$; otherwise, discard $e$ and examine next top element from $H$. Repeat while $H$ is not empty.
5. Go to step 2 if stopping criteria is not met (to be discussed).

A *pass* covers steps from 2 to 4, and `ESC` might require multiple passes before termination depending on the stopping criteria.

### B. Size Constrained Edge Contraction

After the computation of contractible edge set $Z(G)$, we heapify edges in $Z(G)$ into edge heap $H$. The *rank* of edge $e = (x, y)$ in $H$, denoted by $r(e)$, is determined as follows;

$$r(e) = \frac{q(e)}{\min\{c(x), c(y)\}} \tag{3}$$

All edges in $H$ are ordered in descending order of their rank. The rank $r(e)$ is computed in such a way that it gives higher priority to edges with larger $q(e)$ values and edges whose contraction results in smaller increase of degree. We use $\min\{c(x), c(y)\}$ instead of $c(x) + c(y)$ to encourage merging of dangling vertices (e.g., vertices with connection to only one other cluster). Ties are randomly broken.

Our `CONTRACT` algorithm performs size constrained edge contraction for given contractible edge set $Z(G)$ and cluster size limit $L$. `CONTRACT` builds and manages edge heap $H$ based on edge rank $r(e)$ and updates the given graph $G$ upon each edge contraction to obtain correct $G/Z(G)$ upon its termination. One important thing to note is that if some parallel edges are merged into single edge $e'$ from edge contraction, the maximum $q$ value among the parallel edges is selected for $q(e')$ to maintain lower bound on edge separability. This in turn causes some edges not included in $H$ initially to be inserted into $H$ due to the update of $q$. If $t_1$ denotes $|Z(G)|$ and $t_2$ denotes the maximum degree among vertices in $G$, the complexity of `CONTRACT` is $O(t_1 t_2 \log t_1)$ due to the $O(\log t_1)$ time update of $H$ for $O(t_2)$ number of edges, which repeats for $t_1$ times in total. In most cases, large nets ($> 100$) such as clock nets are ignored during clustering and partitioning in VLSI circuit design. Then, we may assume that the maximum degree is bounded by a constant $k$, which also implies that $|E| \leq |V| \cdot k/2$. Under such an assumption, the time complexity of `CONTRACT` becomes $O(t_1 t_2 \log t_1) = O(n \cdot k/2 \cdot k \log(n \cdot k/2)) = O(n \log n)$.

### C. Edge Separability based Clustering

Let $G^{i+1}$ denote the reduced graph obtained by applying `CONTRACT` on $G^i$ for $0 \leq i \leq P$, where $G^0$ denotes the original netlist graph. Note that we can compute a new set of contractible edges from $G^i$ for all $0 \leq i \leq P$. Intuitively, this is due to the updated edge weight function $w'$ as well as updated $\bar{\lambda}$ during `CONTRACT`. If no size constraint is imposed, we can repeatedly apply `CAPFOREST+CONTRACT` until the graph has only two vertices. However, `CAPFOREST+CONTRACT` will eventually terminate if cluster size limit $L$ is imposed.

We stop the clustering process if the number of contractible edges computed from the current graph $G^i$ is too small ($|Z(G^i)| < \alpha \cdot |E|$), or the actual number of edges contracted during `CONTRACT` on $G^i$ is too small ($< \beta \cdot |Z(G^i)|$). We observe from related experiment that a typical number of passes, i.e. num-

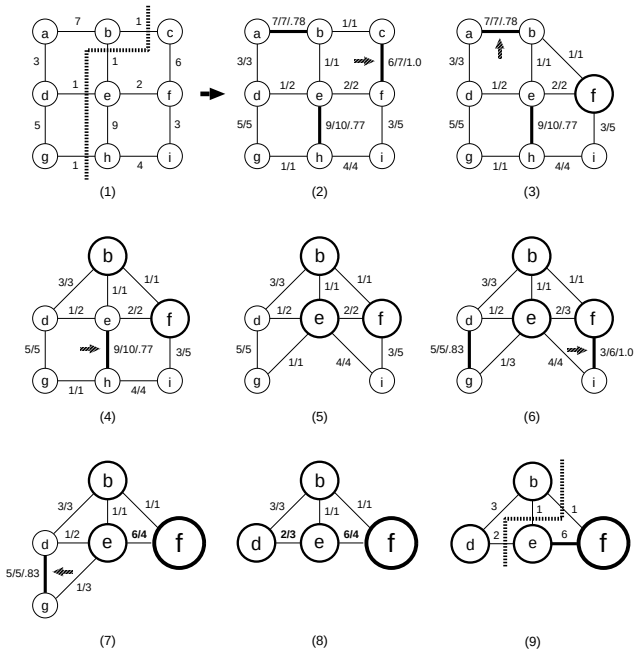

Fig. 2. Illustration of ESC algorithm. Contractible edges are shown in thick lines, and arrows point to the edges contracted under the size constraint $L = 3$. The minimum cut $U$ is shown in thick dotted line. The edge label means $w(e)/q(e)/r(e)$, and bold edge label denotes the updated $w(e)/q(e)$ from contraction. (1) $G^0$, (2) $Z(G^0)$ based on $\bar{\lambda} = \delta(G^0) = c(g) = 6$, (4) $\bar{\lambda} = c(b) = 5$, (5) $G^1$, (6) $Z(G^1)$ based on $\bar{\lambda} = 5$, (8) $G^2$, (9) $U$ preserved in $G^2$.

ber of CAPFOREST+CONTRACT calls until the termination of ESC algorithm, ranges from 1 to 3, depending on the circuit size. CAPFOREST takes $O(m + n \log n)$ and CONTRACT takes $O(n \log n)$ time as discussed in Section III-B. Therefore, the overall complexity of ESC algorithm is $O(k \cdot (m + n \log n + n \log n)) = O(n \log n)$. An illustration of ESC algorithm is shown in Figure 2.

### D. Bottom-Up Clustering Framework

We establish the following framework based on the netlist graph $G(V, E, s, w)$ in order to capture many proposed approaches in the literature;

1. *edge contraction based bottom-up clustering* : initially, each vertex in $G$ belongs to its own cluster. Then, we choose an edge $e = (C_1, C_2)$ with the best *contraction cost*. If $s(C_1) + s(C_2)$ is within the specified size constraint $L$, we perform the contraction of $e$ to obtain larger cluster $C$ and updated $G/e$. Then, the next maximum cost edge will be considered. This greedy selection continues until no more edge can be contracted under the size constraint $L$.

2. *two-phase top-down partitioning* : as a way to integrate clustering into partitioning, we perform partitioning such as FM first on the clustered circuit for global optimization and then on the declustered circuit for local refinement.

For the given edge $e = (C_1, C_2)$ whose contraction generates new larger cluster $C$, the following contraction cost functions are proposed in the literature;

1. *Absorption* [18] (maximize): the absorption of $C$ is defined as $abs(C) = \sum_e w(e)$ for all $e = (x, y)$, $x, y \in C$. It measures the sum of weights of edges "absorbed" into $C$.

2. *Density* [9, 11] (maximize): the density of $C$ is computed as $abs(C)/s(C)$. It measures the "density" of $C$ by taking the ratio of the edges "absorbed" in $C$ to the size of $C$.

3. *Rent Parameter* [15] (minimize): the Rent parameter of $C$ is computed as follows;
$$\frac{\ln(c(C)) - \ln(d(C))}{\ln s(C)}$$
where $d(C) = \frac{1}{|C|} \sum_{x \in C} c(x)$ is the average degree of vertices in $C$, and $|C|$ is the number of vertices in $C$. A "better" placement can be obtained from the smaller Rent parameter associated with each cluster according to the Rent's rule.

4. *Ratio Cut* [19] (minimize): the ratio cut of $C$ is computed as $c(C)/s(C)$. It is a ratio of the sum of weights of outgoing edges to the size of $C$. It tries to identify "natural" clusters by finding cuts that minimize $c(C)/s(C)$.

5. *Closeness* [17] (maximize): the closeness between $C_1$ and $C_2$ is defined as follows;
$$\frac{w(e)}{\min\{c(C_1), c(C_2)\}} - \gamma \frac{s(C_1) + s(C_2)}{ave\_size}$$
where $ave\_size$ denotes the average cluster size, and $\gamma$ is a user-specified parameter to determine the magnitude of penalty term for generating large clusters. It measures the "attraction" between two clusters based on local connectivity information.

6. *Connectivity* [16] (maximize): the connectivity between $C_1$ and $C_2$ is defined as follows;
$$f(s(C_1)) \frac{w(e)}{c(C_1) - w(e)} + f(s(C_2)) \frac{w(e)}{c(C_2) - w(e)}$$
where $e = (C_1, C_2)$, and $f(s(C)) = 1/s(C)$. It is another local connectivity based method that focuses on (i) minimizing number of edges cut after contraction, (ii) preventing early formation of large clusters.

We observe that local connectivity information $w(e)$ plays a major role in determining the sequence of edge contraction for the six schemes shown above. ESC uses $r(e)$ defined in Section III-B as the contraction cost function. The maximum top-level cluster size limit $L$ is set to $\log_2 n$, where $n = |V|$.

## IV. Experimental Result

We implemented our algorithms in C++/STL and tested on SUN ULTRA SPARC60 at 360Mhz. We use area skew of [.45, .55] for 2-way, and $[0.49^n, 0.51^n]$ for 8-way ($n = 3$), 16-way ($n = 4$), and 32-way ($n = 5$) partitioning. We use Cost-1 metric for 2-way cutsize, and SOED (Sum Of External Degrees) metric for all others. The cell area is uniform, and all pads are included to be partitioned. All partitioning results are based on the minimum cutsize of 20 runs, and runtime is measured in seconds. We use ISPD98 benchmark Suite [1].

Table I shows the comparison of various bottom-up clustering algorithms introduced in Section III-D in terms of bipartitioning results. Algorithms in comparison include Absorption [18], Density [9, 11], Rent Parameter [15], Ratio Cut [19], Closeness [17], Connectivity [16], and Edge Separability based method. As discussed in Section III-D, all algorithms are based on (i) edge contraction based bottom-up clustering, and (ii) `FM` based two-phase top-down partitioning framework. Under this framework, `ESC` outperforms all other algorithms that rely on local connectivity information. Runtime results are comparable in all cases.

We developed a multiway partitioning algorithm named `LR/ESC-PM` that uses `ESC` clustering algorithm. It performs `LR` bipartitioning algorithm [6] on top of two-level `ESC` cluster hierarchy. This is then used as the bipartitioning engine for pairwise movement based multiway partitioning framework `PM` [7]. As shown in Table II, `LR/ESC-PM` obtains comparable results to state-of-the-art `hMetis` [12] and `hMetis-Kway` [13] in terms of 2-way, 8-way, 16-way, and 32-way partitioning results. `hMetis` constructs multiple level cluster hierarchy and performs complex declustering and reclustering process called V-cycle. We emphasize that even with simple two-level `ESC` cluster hierarchy along with two-phase partitioning scheme, we can identify natural clusters and obtain high quality partitioning result.

## V. Conclusion and Ongoing Work

We introduced a new efficient bottom-up clustering algorithm named `ESC` (Edge Separability based Clustering). `ESC` exploits more global connectivity information called edge separability to guide clustering process. In addition to cutsize minimization, we are currently developing efficient partitioning techniques to optimize the impact of partitioning on circuit performance with consideration of local and global interconnects. We are exploring techniques such as performance-driven clustering with retiming [5] and/or relaxed acyclic partitioning [8].

## References


[1] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. Int. Symp. on Physical Design*, pages 80–85, 1998.

[2] C. J. Alpert and A. B. Kahng. A general framework for vertex ordering with applications to netlist clustering. In *Proc. Int. Conf. on Computer-Aided Design*, pages 652–657, 1994.

[3] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: a survey. *Integration, the VLSI Journal*, pages 1–81, 1995.

[4] J. Cong and Y. Ding. On area/depth trade-off in LUT-based FPGA technology mapping. In *Proc. Design Automation Conf.*, pages 213–218, 1993.

[5] J. Cong, H. Li, and C. Wu. Simultaneous circuit partitioning/clustering with retiming for performance optimization. In *Proc. Design Automation Conf.*, 1999.

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

[7] J. Cong and S. K. Lim. Multiway partitioning with pairwise movement. In *Proc. Int. Conf. on Computer-Aided Design*, pages 512–516, 1998.

[8] J. Cong and S. K. Lim. Performance driven multiway partitioning. In *Proc. Asia South Pacific Design Automation Conf.*, 2000.

[9] J. Cong and M. Smith. A parellel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design. In *Proc. Design Automation Conf.*, pages 755–760, 1993.

[10] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of ACM*, pages 921–940, 1988.

[11] D. J. Huang and A. B. Kahng. When clusters meet partitions: new density-based methods for circuit decomposition. In *Proc. European Design and Test Conf.*, pages 60–64, 1995.

[12] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. Design Automation Conf.*, pages 526–529, 1997.

[13] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proc. Design Automation Conf.*, 1999.

[14] H. Nagamochi and T. Ibaraki. Computing edge connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Math.*, pages 54–66, 1992.

[15] T. K. Ng, J. Oldfield, and V. Pitchumani. Improvements of a mincut partition algorithm. In *Proc. Int. Conf. on Computer-Aided Design*, pages 470–473, 1987.

[16] D. M. Schuler and E. G. Ulrich. Clustering and linear placement. In *Proc. Design Automation Conf.*, 1972.

[17] H. Shin and C. Kim. A simple yet effective technique for partitioning. *IEEE Trans. on VLSI Systems*, pages 380–386, 1993.

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

[19] Y. C. Wei and C. K. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Trans. on Computer-Aided Design*, pages 911–921, 1992.

TABLE I

COMPARISON OF VARIOUS BOTTOM-UP CLUSTERING ALGORITHMS IN TERMS OF BIPARTITIONING RESULTS. ALL ALGORITHMS USE TWO-PHASE BOTTOM-UP CLUSTERING AND TOP-DOWN PARTITIONING FRAMEWORK EXPLAINED IN SECTION III-D. ALGORITHMS IN COMPARISON INCLUDE ABSORPTION (ABS) [18], DENSITY (DEN) [9, 11], RENT PARAMETER (REP) [15], RATIO CUT (RTC) [19], CLOSENESS (CLO) [17], CONNECTIVITY (CON) [16], AND EDGE SEPARABILITY (ESC) BASED METHOD. TIME INCLUDES TOTAL CLUSTERING AND PARTITIONING TIME.

| ckt | size | ABS | DEN | REP | RTC | CLO | CON | ESC |
|---|---|---|---|---|---|---|---|---|
| ibm01 | 12752 | 252 | 251 | 262 | 186 | 187 | 307 | 205 |
| ibm02 | 19601 | 279 | 281 | 275 | 300 | 296 | 275 | 265 |
| ibm03 | 23136 | 1038 | 1196 | 1174 | 1074 | 1081 | 1194 | 1046 |
| ibm04 | 27507 | 598 | 571 | 618 | 596 | 570 | 627 | 567 |
| ibm05 | 29347 | 1841 | 1767 | 1773 | 1751 | 1744 | 1743 | 1707 |
| ibm06 | 32498 | 971 | 942 | 967 | 1035 | 994 | 1047 | 1002 |
| ibm07 | 45926 | 956 | 902 | 952 | 973 | 1124 | 1322 | 894 |
| ibm08 | 51309 | 1191 | 1183 | 1393 | 1384 | 1475 | 1490 | 1179 |
| ibm09 | 53395 | 1145 | 979 | 1362 | 747 | 865 | 1572 | 690 |
| ibm10 | 69429 | 1859 | 1451 | 1598 | 1477 | 1472 | 2010 | 1399 |
| ibm11 | 70558 | 1534 | 1234 | 1291 | 1169 | 1490 | 1956 | 1120 |
| ibm12 | 71076 | 2424 | 2134 | 2634 | 2323 | 2218 | 2810 | 2192 |
| ibm13 | 84199 | 1031 | 1136 | 1245 | 1145 | 1326 | 2034 | 1152 |
| ibm14 | 147605 | 2196 | 2294 | 3540 | 2287 | 2570 | 3342 | 2078 |
| ibm15 | 161570 | 4414 | 4395 | 4754 | 4512 | 4427 | 4386 | 3801 |
| ibm16 | 183484 | 2429 | 2247 | 2855 | 3574 | 2497 | 2216 | 2005 |
| ibm17 | 185495 | 2869 | 3237 | 4134 | 2743 | 2835 | 3127 | 2545 |
| ibm18 | 210613 | 2347 | 2324 | 3423 | 3723 | 2875 | 2634 | 2205 |
| TOTAL | - | 29374 | 28525 | 34250 | 30999 | 30046 | 34092 | 26052 |
| RATIO | - | 1.13 | 1.09 | 1.31 | 1.19 | 1.15 | 1.31 | 1.00 |
| TIME | - | 8942 | 9875 | 12442 | 9435 | 9645 | 9343 | 9973 |

TABLE II

COMPARISON OF hMetis-Kway *vs* LR/ESC-PM. 2-WAY PARTITIONING RESULTS ARE BASED ON COST-1 METRIC, WHILE OTHERS ARE BASED ON SOED (SUM OF EXTERNAL DEGREES) METRIC. hMetis 2-WAY RESULTS ARE BASED ON 100 RUNS, WHILE ALL OTHERS USE 20 RUNS. hMetis 2-WAY RESULTS ARE FROM http://vlsicad.cs.ucla.edu/~cheese/errata.html, WHICH USE PENTIUM PRO 4-WAY SMP AT 200MHZ. OTHER hMetis-Kway RESULTS ARE FROM [13] THAT USE PENTIUM II AT 300HHZ.

| | | hMetis-Kway | | | | LR/ESC-PM | | | |
|---|---|---|---|---|---|---|---|---|---|
| ckt | size | 2-way | 8-way | 16-way | 32-way | 2-way | 8-way | 16-way | 32-way |
| ibm01 | 12752 | 180 | 1750 | 2883 | 4149 | 180 | 1777 | 2829 | 4147 |
| ibm02 | 19601 | 262 | 3850 | 7556 | 11821 | 297 | 4490 | 8145 | 11787 |
| ibm03 | 23136 | 956 | 5820 | 8205 | 11077 | 972 | 5737 | 8297 | 11002 |
| ibm04 | 27507 | 542 | 6214 | 8992 | 12495 | 526 | 6170 | 9123 | 12341 |
| ibm05 | 29347 | 1715 | 10749 | 15206 | 20020 | 1709 | 10410 | 14372 | 18898 |
| ibm06 | 32498 | 888 | 5784 | 8661 | 12779 | 902 | 6468 | 9173 | 13088 |
| ibm07 | 45926 | 853 | 7586 | 11040 | 15559 | 848 | 7566 | 11109 | 15224 |
| ibm08 | 51309 | 1142 | 7979 | 10976 | 15327 | 1164 | 8214 | 11281 | 15646 |
| ibm09 | 53395 | 624 | 5822 | 8634 | 12460 | 624 | 5918 | 8481 | 12339 |
| ibm10 | 69429 | 1256 | 9144 | 13130 | 19941 | 1377 | 8795 | 13908 | 19922 |
| ibm11 | 70558 | 960 | 7874 | 11706 | 17118 | 987 | 8129 | 11723 | 16941 |
| ibm12 | 71076 | 1918 | 12910 | 17848 | 25228 | 2015 | 13035 | 17976 | 24161 |
| ibm13 | 84199 | 840 | 6079 | 11819 | 17350 | 850 | 6866 | 12547 | 17902 |
| ibm14 | 147605 | 1837 | 11258 | 18232 | 29699 | 1885 | 10992 | 17745 | 27760 |
| ibm15 | 161570 | 2625 | 14586 | 20826 | 31874 | 2846 | 15565 | 20453 | 31162 |
| ibm16 | 183484 | 1755 | 14616 | 22924 | 34879 | 1811 | 14877 | 23520 | 34775 |
| ibm17 | 185495 | 2238 | 19930 | 33344 | 45961 | 2214 | 20468 | 33124 | 44403 |
| ibm18 | 210613 | 1541 | 12177 | 19598 | 30558 | 1673 | 12885 | 19311 | 29188 |
| TOTAL | - | 22132 | 164128 | 251580 | 370295 | 22880 | 168362 | 253117 | 360686 |
| TIME | - | - | - | - | - | 10865 | 27035 | 33449 | 38406 |