Multilevel Circuit Partitioning with Maximal-matching Based Clustering and Signal-flow Based Clustering

Wai-Shing Luk, *Member, IEEE*

***Abstract*— Primal-dual algorithm is derived for solving the maximal-matching based clustering (MMC) problem within the multilevel circuit partitioning framework. Compared with the usual greedy approaches, the method generates high-quality clusters using less amount of time. When signal direction information is available from input circuit, a signal-flow clustering method is presented. An algorithm for handling feedback paths and bi-directional pins is proposed. Experiments show that the resulting methods produce a very promising quality of cutsize and run-time performance compared with the recent cutting–edge packages `hMetis` and `UCLA_MLPart`. The tradeoffs between different implementations are also discussed. In particular, the tradeoff between reliability and performance is investigated.**

***Index Terms*— Circuit partitioning, multilevel partitioning, clustering, multiway partitioning, hypergraph partitioning, primal-dual.**

I. Introduction

CIRCUIT partitioning is a fundamental CAD problem and finds application in diverse areas such as top-down placement and circuit simulation. The problem is usually expressed in graph theoretic term. Recall that a hypergraph H is a pair of sets (V, E). The set $V = \{v_1, \ldots, v_n\}$ is a set of the *vertices* and $E = \{e_1, \ldots, e_m\}$ is a set of *hyperedges* (or simply *edge*). Each hyperedge $e_i = \{v_1, \ldots, v_k\}$ is a subset of V. A *weighted hypergraph* is a hypergraph with a function $w : V \mapsto \mathbf{R}$. To connect a circuit (or a *netlist*) with the hypergraph notations, each *module* is represented by a vertex in H. A module can be either a *cell* or a *pad*. The area of each module is represented by the weight function of vertex $w(v_i)$. Each *net* is represented by a hyperedge. Consider the *Minimum Balanced Cut* problem that requires to partition the vertices into K disjoint subsets $V_1, \ldots, V_K$ such that each subset has equal sum of vertices weights. The objective of the problem is to minimize the number of hyperedges crossing the partition, which is called the *cutsize*. The Minimum Balanced Cut problem is proved NP-complete. Nevertheless, because of the importance of the problem, extensive research on finding efficient heuristics have been studied (c.f. the survey paper of [1]). They include methods that are based on local search (e.g. the Kernighan–Lin (KL) method [2] and the Fiduccia–Mattheyses (FM) method [3]), methods are that based on simulated annealing (e.g [4]), methods that based on genetic algorithm (e.g. [5], [6]), methods are that based on tabu search (e.g. [7]), methods are that based on neural network (e.g. [8], [9], [10], [11], methods that are based on mathematical programming (e.g. [12], [13]), methods that are based on multilevel paradigm (see below) and the hybrid of those methods (e.g. [14]).

In circuit partitioning, currently the standard approach is the iterative improvement method. Given an initial feasible solution, in the classical local search approach, modules are successively moved between partitions until there is no more positive *gain* on the current solution. While the iterative improvement method generates new solutions by performing sequences of moves on the current solution per pass. As a result, even though a first few moves may produce negative gain, if the final sequence of moves produces positive gain, the algorithm can still proceed and hence generates smaller cutsize in general. One of the best iterative improvement algorithms is known as the Fiduccia–Mattheyses (FM) algorithm [3]. In the FM algorithm, an array of *gain buckets* is used to maintain the gains of moves. Since the maximum gain for a single move is bounded by $[-p_{max}, p_{max}]$ where p_{max} is the maximum pin count of module, the algorithm runs in nearly linear time per pass if p_{max} is small. It makes the FM algorithm very attractive as a basic algorithm. However, due to the natural of the local search, the iterative improvement method can only find solutions in local minimum.

In the conventional graph partitioning problem, a method of using eigenvectors from the corresponding Laplacian matrix of the graph was shown to be successful to obtain more global solution. One of the advantages of the method is that the smallest eigenvalue gives the lower bound of the cutsize so that the quality of the final cutsize can be estimated. However, to apply the method to the hypergraph partitioning, we first need to approximate the hypergraph by a graph and apply the eigenvector method to the resulting graph. However, obtaining a good approximation is still a big challenge.

In recent years, methods that are based on multilevel paradigm were got attention and they are shown to be outperformed other methods [15], [14], [16], [17], [18], [19], [20], [21], [22]. We state below a two-level cycle of the method. The notations H_ω and H_Ω are used to denote the fine hypergraph and the coarse hypergraph respectively.

- Coarsening. The vertices of H_ω are grouped into clusters, and these clusters form the new vertices of H_Ω.
- Coarse-graph partitioning. A standard partitioning approach such as the FM algorithm is used to obtain the initial partition solutions of H_Ω.
- Uncoarsening and refinement. The solution of H_Ω is projected to H_ω and a refinement algorithm is used on H_ω to reduce the cutsize without violating the balance

constraints.

This two-level cycle can be applied in a recursive way to obtain a multilevel cycle.

In the clustering, one of the objectives is to reduce the number of vertices and the number of hyperedges significantly while the coarsen hypergraph remains good approximation of the original hypergraph. A maximal matching based clustering is commonly used. A set of matching hyperedges is selected and the vertices connected to each hyperedge are grouped in clusters accordingly. In the literature, greedy algorithm or random walk methods are used to obtain the set of matching hyperedges. In this paper, we present a new approach that is based on the transformation of the weighted maximal matching problem to the weighted cover problem. The underlying weighted cover problem is then solved by a generalized Primal-dual algorithm. Compared with the greedy algorithm that in worst case requires $O(n \log n$ run-time for sorting, the proposed method runs in linear time while good quality of the coarsen hypergraphs is maintained.

When signal direction information is available, signal-flow based clustering has been proposed. One problem of this method is that a circuit may contain feedback paths and bi-directional pins. In [23], a method of duplicating modules was suggested. However, the duplicated modules become problematic since they must always appear in the same partition. In this paper, we take another approach that removes feedback paths by temporarily reversing the direction of some pins and re-assigns the direction of a set of bi-directional pins temporarily to either input direction or output direction. In order to make the change as small as possible, a greedy algorithm is proposed in this paper.

The rest of the paper is organized as follows. In Section II, we present our version of multilevel algorithm. Two new clustering algorithms are proposed in Section II-A.1 and Section II-A.2. A discussion of obtaining initial partitioning is given in Section II-B. In particular, tradeoff between different implementations are discussed. Experimental results are given in Section III which shows that our algorithms produce a very promising quality of cutsize and run-time performance compared with the recent cutting–edge packages such as `hMetis` and `UCLA_MLPart`. Finally, in Section IV some conclusions are given and some possible directions for further research are suggested.

II. Overview of Our Implementation

In [24], it was shown that the LIFO bucket organization outperforms the FIFO and random bucket organizations in the FM algorithm. It suggests that modules moves in cluster tend to produce good results in practice. In fact, partitioning algorithms have been shown to be improved significantly by using clustering approach. The multilevel method recursively apply this approach to further improve both the run-time and quality of results. The tradeoff of the storage requirement is obviously expected because every coarse graphs in different levels are needed to be stored. In one of our experiments for 200,000 modules, the standard FM algorithm took about 32MB memory and our multilevel method took about 100MB memory. However, based on the current computer technology that 4GB memory PC or workstation is affordable, the multilevel method is quite attractive.

Our multilevel method starts with successively generating a sequence of coarse hypergraphs. We will describe this process in detail in Section II-A. The process repeats until the number of modules in a coarse graph is less than or equal to the maximum pin count, or no feasible initial partition is obtained in the coarsest graph. Then initial partition algorithm is applied to the coarsest hypergraph. In Section II-B, a more detail of this initial phase will be discussed. The result is then projected to the next level of finer hypergraph and the refinement procedure is applied. The refinement phase will be presented in Section II-C. The refinement phase is repeatedly applied until the bottom level is rearched. In practice, more sophisticated strategies could be used such as *V-cycle* and *W-cycle* (terminologies borrowed from multigrid method in solving differential equations [25]).

A. Coarsening Phase

In the coarsening phase, we first select a set of matching edges by clustering algorithms described in the following sections.

1) Maximal-Matching Based Clustering: In the maximal-matching based clustering (MMC), a maximal-matching set of hyperedges is obtained from H, i.e., a subset $E' \subseteq E$ such that no two edges in E' shares a common vertex and each edge in $E - E'$ shares at least a common vertex with some edge in E'. A vertex that is not adjacent to any edges in E' is called *an isolated* vertex. Usually a following greedy heuristic is used to achieve the goal. The edges are first sorted in ascending order according to edge weight. Each time the edge with the smallest weight, say e_i, is chosen. The algorithm then eliminates e_i and all its neighbors from H and stops when all edges are deleted. Thee algorithm usually take $O(n \log n)$ sorting time.

To solve the maximal matching problem, we propose a method that first solve a weighted minimum edge cover problem, i.e., a subset $E'' \subseteq E$ such that, for each vertex, at least one of the adjacency edges belongs to E''. We employed a primal-dual algorithm for solving this problem that will be described later below. Then based on the solution of edge cover problem, we perform a post-processing such that no two edges in E' share a common vertex. It can be done by traversing all modules once. If a vertex adjacent to more than one edge in E'', the one with minimum weight $c_i(e_i)$ is chosen. Other edges are removed from E''. Since this action may affect some neighbor edges of the removed edges uncovered, some of these uncovered edges are needed to be added to E'' so that each neighbor edge shares at least a common vertex with some edges in E'. Fortunately it can be shown that no further edges will be affected if we carefully choose which uncovered edges to be added. The detailed algorithm is presented in Fig. 1.

For solving the weighted edge cover problem, we extend the primal-dual algorithm described in [26] that originally solves graph weighted vertex cover problem. Each vertex v_i has a non-negative dual variable y_i that is initially zero. Each edge

```
MaximalEgdeMatching $(H, c(e) \geq 0)$
$E' \leftarrow$ PrimalDualEdgeCover$(H, c)$
for each vertex $v \in V$ do
  if more than one net covers $v$ then
    Let $S$ be the set of nets that covers $v$
    Select the net $e'$ with minimum weight in $S$
    $S \leftarrow S - \{e'\}$
    for each edge $e \in S$ do
      $E' \leftarrow E' - \{e\}$
    endfor
    for each edge $e \in S$ do
      for each vertex $v' \in e$ do
        if $v'$ is not covered then
          $S' \leftarrow \emptyset$
          for each edge $e'' \in adj(v')$ do
            if $e''$ doesn't have any covered vertices then
              $S' \leftarrow S' \cup \{e''\}$
            endif
          endfor
          if $S' = \emptyset$ then
            $v'$ is an isolated vertex
          else
            Select the net $e'''$ with min. weight in $S'$
            $E' \leftarrow E' \cup \{e'''\}$
          endif
        endif
      endfor
    endfor
  endif
endfor
return $E'$
```

Fig. 1. Solving maximal matching problem by primal-dual algorithm.

```
PrimalDualEgdeCover $(H, c(e) \geq 0)$
for each dual variable $y_i$ do $y_i \leftarrow 0$ endfor
$E'' \leftarrow \emptyset$
while $E''$ is not an edge cover do
  Let $v_i$ be a vertex not covered by $E''$
  Increase $y_i$ until a constraint become tight for
    one particular adjacency edge $e'$ :
  if $\sum_{v_j \in e'} y_j = c(e')$ then
    $E'' \leftarrow E'' \cup \{e'\}$
  endif
endwhile
return $E''$
```

Fig. 2. Solving minimum edge cover problem by primal-dual algorithm.

has a constraint that the sum of y_i of its connected vertices must be less than or equal to the edge weight. Each time a vertex is visited and its dual variable is increased until one of the constraints of its adjacency edges is tight. That edge will be added to E''. The overall algorithm is presented in Fig. 2. For conventional graph (i.e. $k \leq 2$), it was proved that the solution is guaranteed to be at most twice the optimal solution [26]

Note that the above algorithm can be used to achieve other objectives by reformulating the edge weight with other functions. For example, edges could be weighted by the cluster size, pin count and *separability* [19] and the combination of those. The overall run time of the above algorithm is nearly linear.

2) Signal-Flow Based Clustering: If signal direction information is available, it is reasonable to choose an independent set of nets according to the signal flow. Thus, we implemented the Signal-flow based clustering (SFC). Because a circuit may contain feedback paths and bi-directional pins, therefore, in the first step, we need to make the input circuit "acyclic". It can be done by temporarily reversing the direction of a set of pins. Obviously we want this set of pins as small as possible. A simple technique is to perform a depth first search on the circuit and reverse the pin direction of all the backward nets. However as mentioned in [27], the performance of this algorithm is poor. In our implementation, we extend the greedy algorithm described in [27] originally for graph. The algorithm is relatively simple, and runs in nearly linear time, assuming that the maximum pin count of modules is bounded by a small value. The algorithm starts with obtaining a *vertex sequence* that will be used later for pin direction assignment. Fig. 3 shows the greedy algorithm that obtains the vertex sequence. Vertices are added into one of two lists, S_l and S_r. The greedy algorithm starts with all sources are added to the tail of S_l and all sinks are added to the head of S_r. Denote $outdeg(v_i)$ the out-degree of vertex v_i and $indeg(v_i)$ the in-degree of vertex v_i. After dealing with all sources and sinks, we choose a vertex for which $outdeg(M)$ - $indeg(M)$ is maximum and add it to the tail of S_l. The selection process is maintained by a bucket structure so that the run time is nearly linear. After all vertices are visited, S_l and S_r are then concatenated. The resulting list S is the *vertex sequence* of H.

```
GreedyVertexSequence $(H$, Set of pin direction$)$
$S_l \leftarrow \emptyset$ ; $S_r \leftarrow \emptyset$
$H' \leftarrow H$
while $H' \neq \emptyset$ do
  while $H'$ contains a sink do
    Select a sink $u$
    $H' \leftarrow H' - \{u\}$
    $S_r$.prepend$(u)$
  endwhile
  while $H'$ contains a source do
    Select a sink $v$
    $H' \leftarrow H' - \{v\}$
    $S_l$.append$(v)$
  endwhile
  if $H' \neq \emptyset$ then
    Select a vertex $u$ with maximum outdeg$(u)$ − indeg$(u)$
    $H' \leftarrow H' - \{u\}$
    $S_l$.append$(u)$
  endif
endwhile
$S \leftarrow$ Concatenate$(S_l, S_r)$
return $S$
```

Fig. 3. Hypergraph cycle removal procedure by Greedy algorithm

The next step is to re-assign the pin directions according to the vertex sequence. Suppose that $S = (v_1, v_2, \ldots, v_n)$ of the vertices. The directions of pins are re-assigned such that signal flows only from v_i to v_j with $i < j$. To do the assignment, we traverse the list twice. In the first time, we only assign the *leftmost* pin and *rightmost* pin of each net. The leftmost pins always have output direction and the rightmost pins always have input direction. In the second time, we assign each pin output direction if some leftward pin of the same net has output direction and each pin input direction if some rightward pin of

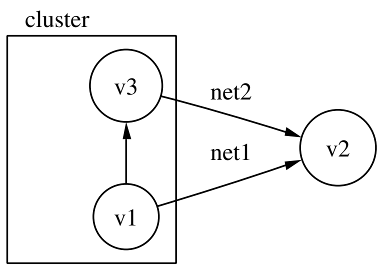

Fig. 4. Parallel edges created during coarsening

the same net has input direction. If a pin direction still cannot be assigned, then output direction will be assigned to that pin. After the cycle removal procedure, we perform a breadth-first traversing from sources and sinks alternatively and construct an independent set of nets accordingly.

After a set of selected nets are found by the above clustering algorithms, the vertices of each selected edges are grouped to form a cluster. In order to significantly reduce the size of hypergraph, we follow the suggestion of [15] that vertices of other edges are then grouped together to form a cluster if there are more than one vertex connected to the edge. Then the remaining isolated modules are simply copies to the next level. Note that this process may create *parallel edges* as shown in Fig 4. The parallel edges should be merged into a single edge whose weight is equal to the sum of weights of the parallel edges. In conventional graph, parallel edges can be easily identified because each edge has at most two vertices. However, in a hypergraph, edges can have more than two vertices. In our implementation, only parallel two-pin nets and parallel three-pin nets are merged. To identify parallel edges with more than three pins is more tedious and whether the overall performance can be improved is unknown. Thus, we leave it as a future work.

B. Initial Partition Phase

This phase is applied to the coarsest hypergraph. Usually initial partitioning is given by randomly assigning vertices in each partition. In [22], a new technique was proposed that initially puts all modules into one partition and then moves a module only if it does not *increase the violation* of balance constraints rather than necessarily result in a legal solution. The method is called VILE ("Very Illegal") in [22]. Observed from our experiments, the VILE scheme gives significantly smaller final cut-size of the benchmark test cases. However, there is a risk that the sequence of moves may not eventually produce a legal solution. Even in the random initialization method, whether the solution is feasible is not guaranteed when actual area of module is used. A conservative way is to minimize the different area between two partitions in this phase. Modules are sorted in non-increasing order according to their areas and are put into each partition by the following rule: always put the next module in the partition of smallest total area. In gate-level netlist, the number of different areas of modules is usually small because many modules are in fact from the same master copy. Thus, we can create a binary search tree for different area type and "bucket" the modules with same area into the same tree node. It makes the sorting method take nearly linear time for the flat-FM algorithm. For the multilevel method, since each module/cluster may have different area, the $O(n \log n)$ run time will be expected.

One problem of our method is that the algorithm is purely deterministic. In multiple runs meta-heuristic, this will limit the seach space of our algorithm. We may add some randomness that the modules with same area are inserted to the binary search tree in random order. However, this only works for the flat-FM algorithm. This raises a question that which implementation is preferable. Should we choose a more conservative method or a more aggressive method? In practice, we may provide two versions of program and let user to choose which version to use. Finally, a warning message should be displayed whenever the solution is illegal in both versions.

C. Refinement Phase

Our refinement algorithm is based on the standard Fiduccia-Mattheyses (FM) heuristic [3]. A more detail description is presented below that follows the reporting method described in [28]:

- If two modules from the two gain buckets have the same highest gain and both of them satisfy the balance constraints, the one which makes better balance condition will be chosen.
- When the delta gain of a move is zero, the gain update will be skipped.
- New element in gain bucket will be attached in LIFO fashion.
- When selecting the best solution encountered during the pass, the tie-breaking decision is that the one that makes better balance condition will be chosen.

Note that the nearly linear run-time per pass is expected in this algorithm if the maximum pin count p_{max} is bounded by a small value. In gate-level netlist, this assumption is usually valid. However, in the multilevel method, we may create clusters that have high pin counts. One way to solve this problem may be to switch the conventional priority queue (which is usually implemented by *heap*) whenever p_{max} is large. In our implementation, we simply use the standard FM algorithm and stop to further generate coarse hypergraphs if p_{max} is detected to be larger than the number of modules.

D. Other Implementation Details

We implemented our algorithms in C++/STL. The source code was compiled with g++ on a Linux platform. We use STL data structure for vector, multi-sets, queue, stack, and doubly–linked lists. However, the doubly–link lists used in the bucket structures were implemented in our own code so that they are fine-tuned for this particular purpose. We use one of the boost packages for the memory management. The GUI interface was implemented with Qt package.

TABLE I

COMPARISON OF OUR ALGORITHM AND VILE. AVERAGE CPU TIME IN SECOND IS GIVEN IN PARENTHESIS. SOLUTIONS ARE CONSTRAINED TO BE WITHIN 2% OF BISECTION.

	Our			VILE		
Circuit	Min	CPU	d(%)	Min	CPU	d(%)
ibm01	357	0.39	0.00	226	0.23	0.64
ibm02	363	1.09	0.26	266	0.93	0.00
ibm03	1612	1.85	0.00	1544	1.15	0.79
ibm04	993	1.99	0.57	1397	1.07	1.88
ibm05	2442	5.14	0.00	1864	2.80	0.00
ibm06	1089	2.74	1.71	1017	2.19	0.00
ibm07	1395	4.39	1.11	1325	3.88	0.63
ibm08	1719	7.84	0.11	1624	7.83	1.10
ibm09	2229	4.59	0.39	3017	3.50	1.51
ibm10	1673	7.23	0.07	1575	4.19	0.56
ibm11	2241	6.83	1.60	3513	13.37	0.00
ibm12	3921	9.12	0.91	2957	7.58	0.48
ibm13	1773	8.59	0.08	1790	6.09	0.73
ibm14	3121	38.16	0.42	3010	34.33	1.53
ibm15	5955	22.35	1.01	5872	15.31	0.00
ibm16	2676	31.96	0.11	3318	38.11	0.00
ibm17	3734	52.30	0.00	4099	48.32	1.74
ibm18	1888	55.62	1.90	2935	69.52	0.00

TABLE II

COMPARISON OF VARIOUS CLUSTERING ALGORITHMS IN TERM OF BI-PARTITIONING RESULTS WITH 20 RUNS. SOLUTIONS ARE CONSTRAINED TO BE WITHIN 2% OF BISECTION.

ckt	size	Greedy–MMC	PD–MMC	SFC
ibm01	12752	296 (0.64)	274 (0.51)	267 (0.70)
ibm02	19601	302 (2.27)	271 (1.56)	319 (1.66)
ibm03	23136	1024 (2.41)	1138 (1.78)	1106 (2.36)
ibm04	27507	595 (2.46)	537 (1.87)	639 (2.81)
ibm05	29347	1786 (3.89)	1792 (3.72)	1765 (4.00)
ibm06	32498	807 (3.91)	825 (2.65)	855 (3.82)
ibm07	45926	863 (4.29)	937 (3.92)	882 (5.97)
ibm08	51309	1288 (7.89)	1320 (7.33)	1391 (7.97)
ibm09	53395	1248 (4.84)	887 (4.30)	809 (6.24)
ibm10	69429	1567 (13.56)	1502 (7.55)	1518 (9.96)
ibm11	70558	1129 (7.02)	1134 (6.65)	1139 (8.68)
ibm12	71076	2363 (12.19)	2361 (10.06)	2521 (14.15)
ibm13	84199	1171 (10.14)	1224 (8.85)	1546 (13.10)
ibm14	147605	1995 (17.28)	1966 (17.10)	2070 (25.84)
ibm15	161570	3309 (26.29)	3266 (20.06)	2859 (33.93)
ibm16	183484	1771 (25.48)	1732 (22.83)	1831 (34.73)
ibm17	185495	2668 (32.23)	2542 (33.54)	2583 (44.24)
ibm18	210613	1852 (37.33)	1691 (33.91)	2425 (46.54)

III. EXPERIMENTAL RESULTS

Our algorithms are tested on a Intel Pentium IV at 1.8GHz PC. The source code of our algorithms is available upon request. We use ISPD-98 benchmark Suite that was downloaded from [29]. The actual area of module is used in all experiments. All pads are included to be partitioned. Run-time is measured in second.

A. Comparisons with VILE Initial Partitioning

Our first experiment compares the performance of our initial partitioning algorithm described in Section II-B with VILE using our flat-FM code. Table I reports the minimum cut and the average CPU time of 100 runs. The balance tolerant tol is set to 2% of the total area, i.e., [0.49, 0.51]. The actual different between two partitions (in %) is given in the third column.

At first sight, the VILE scheme produced significant improvement and all final best solution satisfied the area balance constraint. But actually within the 100 runs, some of them in fact could not produce legal solutions, and whenever this happens, the algorithm stops very quickly and consumes only very small amount of time.

In the following, we present the experimental results by using our initial partitioning algorithm only.

B. Comparison between Various Clustering Approaches

The comparison of various clustering algorithms is shown in Table II and Table III. The partition results are based on the minimum cutsize of 20 runs. Greedy–MMC refers to the maximal–matching based clustering with Greedy algorithm. PD–MMC refers to the maximal–matching based clustering with the Primal-dual algorithm described in Section II-A.1. SFC refers to the signal-flow based clustering described in Section II-A.2. We use the cluster size as the net weight in all three methods. In Table II, solutions are constrained to be with 2% of bisection. In Table III, solutions are constrained to be with 10% of bisection. Average CPU time is given in parenthesis. Under this framework, PD–MMC outperforms the other two algorithms.

C. Comparison with Other Leading–edge Packages

We compared two leading–edge multilevel partitioners `hMetis` (version 1.5.3) and `UCLA MLPart` (version 4.19) which are available at [30] and [31] respectively. First, we use the test case `ibm06` and set the balance tolerant to a ridiculously small value 0.001% such that no feasible solution is possible. The partitioner `UCLA MLPart` reported the best cutsize 4224 as if the solution were legal. This may be explained by the fact that the VILE initialization scheme is used by the package. For `hMetis`, since the package only accept the balance tolerant greater than or equal to 1%, in order to test if the similar suitable happen in the hMetis, we modified the area of one module in the test case to be ridiculously large such that no feasible solution is possible. Surprisingly `hMetis` also failed to report any warning.

`hMetis` provides serveral different coarsening schemes, uncoarsening schemes and V-cycle refinement schemes. We use the `shmetis` in which the default schemes are set. Table IV presents bi-partitioning results for actual area and allowing up to 10% derviation from exact bisection. We use the Primal-dual clustering algorithm for our implementation in this experiment. Cluster size is used as the net weight in

TABLE III
COMPARISON OF VARIOUS CLUSTERING ALGORITHMS IN TERM OF BI-PARTITIONING RESULTS WITH 20 RUNS. SOLUTIONS ARE CONSTRAINED TO BE WITHIN 10% OF BISECTION.

ckt	size	Greedy–MMC	PD–MMC	SFC
ibm01	12752	260 (0.50)	217 (0.54)	219 (0.66)
ibm02	19601	259 (1.28)	257 (1.41)	278 (1.64)
ibm03	23136	788 (1.53)	774 (1.57)	942 (2.11)
ibm04	27507	447 (1.82)	481 (1.73)	445 (2.50)
ibm05	29347	1734 (3.66)	1818 (4.09)	1727 (3.81)
ibm06	32498	752 (2.37)	702 (2.43)	363 (3.42)
ibm07	45926	770 (3.70)	740 (3.55)	761 (4.88)
ibm08	51309	1154 (6.60)	1200 (5.95)	1313 (7.18)
ibm09	53395	559 (3.72)	576 (3.54)	578 (5.24)
ibm10	69429	1054 (7.69)	935 (7.60)	938 (10.07)
ibm11	70558	781 (5.36)	730 (5.04)	757 (7.63)
ibm12	71076	2129 (9.39)	2133 (9.25)	2158 (12.93)
ibm13	84199	874 (7.66)	873 (7.62)	1035 (10.52)
ibm14	147605	1528 (15.80)	1580 (15.98)	1788 (22.43)
ibm15	161570	2090 (20.66)	2178 (19.38)	2288 (26.54)
ibm16	183484	1779 (22.72)	1764 (22.50)	1813 (30.19)

TABLE IV
COMPARISON OF HMETIS1.5.3, UCLA MLPART4.19 AND OUR ALGORITHM. SOLUTIONS ARE CONSTRAINED TO BE WITHIN 10% OF BISECTION. AVERAGE CPU TIME IN SECONDS IS GIVEN IN PARENTHESIS.

ckt	shmetis	UCLA ML	OUR
ibm01	248(4.04)	240(1.19)	217(1.71)
ibm02	284(6.15)	310(2.45)	258(4.57)
ibm03	805(8.87)	722(3.09)	1143(5.67)
ibm04	445(10.61)	477(3.07)	502(5.29)
ibm05	1725(16.41)	1741(6.39)	1805(13.88)
ibm06	368(15.90)	484(4.66)	777(8.28)
ibm07	749(27.78)	775(6.56)	778(12.37)
ibm08	1159(33.42)	1225(8.27)	1215(20.16)
ibm09	522(25.57)	520(8.27)	606(12.25)
ibm10	769(47.06)	871(12.56)	1011(23.03)
ibm11	737(42.43)	745(11.80)	737(16.66)
ibm12	1976(62.48)	2741(14.57)	2152(24.48)
ibm13	876(46.53)	1121(15.69)	912(23.16)
ibm14	1578(153.6)	1875(30.71)	1665(50.70)
ibm15	1939(171.7)	1947(40.31)	2498(58.46)
ibm16	1716(212.7)	1684(45.55)	1750(61.97)
ibm17	2249(280.6)	2291(56.69)	2420(102.72)
ibm18	1539(227.0)	2042(53.39)	1864(82.77)

the Primal-dual algorithm. The initial paritioning solution of the coarsest hypergraph is obtained from the best of three runs of the FM algorithm. The best solution of three runs of our multilevel method is reported and the total CPU time is given in parenthesis. The result shows that our multilevel method produces very promising quality of solution even thought both aggressive strategies and non-trivial fine-tuning techniques have not been used.

IV. CONCLUSION

We have presented two new clustering algorithms, namely the Primal-dual algorithm and the Signal-flow based algorithm. The Primal-dual algorithm outperforms the greedy algorithm which is commonly used in the circuit partitioning literature. The signal-flow based algorithm does not seem to be efficient for this area-constrained net-based problem. However, we believe that the algorithms developed in this paper are useful for the path-based problem and the performance-driven problem.

ACKNOWLEDGMENT

The author would like to thank... [29]

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: a survey," *Integr. VLSI J.*, vol. 19, no. 1–2, pp. 1–81, 1995.
[2] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
[3] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conference19th*, 1982, pp. 175–181.
[4] R.-M. King and P. Banerjee, "Optimization by simulated evolution with applications to standard cell placement," in *Proc. ACM/IEEE Design Automation Conference27th*, Orlando, FL, USA, June 1990, pp. 20–25.
[5] G. C. Sipakoulis, I. Karafyllidis, and A. Thanailakis, "Genetic partitioning and placement for VLSI circuits," in *Proc. 6th IEEE International Conference on Electronics, Circuits and Systems*, vol. 3, Pafos, Cyprus, Sept. 1999, pp. 1647–1650.
[6] B.-R. Moon and C.-K. Kim, "Genetic VLSI circuit partitioning with dynamic embedding," in *Proc. First International Conference on Knowledge-Based Intelligent Electronic Systems*, vol. 2, Adelaide. SA, Australia, May 1997, pp. 461–469.
[7] S. Areibi and A. Vannelli, "Circuit partitioning using a tabu search approach," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 3, Chicago, IL, USA, May 1993, pp. 1643–1646.
[8] T. Hameenanttila and J. D. Carothers, "A Hopfield neural network solution to the TCM partitioning problem," in *Proc. IEEE International Conference on Neural Networks*, vol. 7, Orlando, FL, USA, 1994, pp. 4676–4680.
[9] S. Kumar, K. Forward, and M. Palaniswami, "An experimental evaluation of neural network approach to circuit partitioning," in *Proc. IEEE International Conference on Neural Networks*, vol. 1, Perth, WA, Australia, Dec. 1995, pp. 569–574.
[10] J.-S. Yih and P. Mazumder, "A neural network design for circuit partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 9, no. 12, pp. 1265–1271, Dec. 1990.
[11] C. F. Ball and D. A. Mlynski, "A stochastic neural network approach for circuit partitioning," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 4, Atlanta, GA, USA, May 1996, pp. 687–690.
[12] L.-T. Liu, M.-T. Kuo, S.-C. Huang, and C.-K. Cheng, "A gradient method on the initial partition of Fiduccia–Mattheyses algorithm," in *Proc. IEEE International Conference Computer–Aided Design*, San Jose, CA, USA, Nov. 1995, pp. 229–234.
[13] C. J. Alpert and A. B. Kahng, "Simple eigenvector-based circuit clustering can be effective," in *Proc. IEEE International Conference Computer–Aided Design*, vol. 4, Atlanta, GA, USA, May 1996, pp. 683–686.
[14] C. J. Alpert, L. W. Hagen, and A. B. Kahng, "A hybrid multilevel/genetic approach for circuit partitioning," in *Proc. IEEE Asia Pacific Conference on Circuits and Systems*, Seoul, South Korea, Nov. 1996, pp. 298–301.
[15] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, p. 69, Mar. 1999.
[16] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 17, no. 8, pp. 655–667, Aug. 1998.
[17] S. Wichlund and E. J. Aas, "On multilevel circuit partitioning," in *Proc. IEEE International Conference Computer–Aided Design*, San Jose, CA, USA, Nov. 1998, pp. 505–511.
[18] Y. Cheon and D. F. Wong, "Design hierarchy-guided multilevel circuit

partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 22, no. 4, pp. 420–427, 2003.

[19] J. Cong and K. L. Sung, "Edge separability based circuit clustering with application to circuit partitioning," in *Proc. Asia and South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2000, pp. 429–434.

[20] J. Cong, H. P. Li, K. L. Sung, T. Shibuya, and D. Xu, "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering," in *Proc. IEEE International Conference Computer–Aided Design*, San Jose, CA, USA, Nov. 1997, pp. 441–446.

[21] C. J. Alpert, A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hypergraph partitioning with fixed vertices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 19, no. 2, pp. 267–272, Feb. 2000.

[22] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Improved algorithms for hypergraph bipartitioning," in *Proc. Asia and South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2000, pp. 661–666.

[23] C. J. Alpert, "The ISPD-98 circuit benchmark suite," in *Proc. ACM/IEEE International Symposium on Phyical Design*, Apr. 1998, pp. 80–85, see errata at http://vlsicad.cs.ucla.edu/ cheese/errata.html.

[24] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng, "On implementation choices for iterative improvement partitioning algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 16, no. 10, pp. 1199–1205, Oct. 1997.

[25] P. Wesseling, *An Introduction to Multigrid Methods*. Chichester: John Wiley and Sons, 1992.

[26] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marcetti-Spaccamela, and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer–Verlag, 1999, ch. 2.4.2, pp. 67–69.

[27] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph drawing: algorithms for the visualization of graphs*. Prentice–Hall, 1999, ch. 9.4, pp. 294–300.

[28] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, "Hypergraph partitioning for VLSI CAD: methodology for heuristic development, experimentation and reporting," in *Proc. ACM/IEEE Design Automation Conference36th*, New Orleans, LA, USA, June 1999, pp. 349–354.

[29] C. J. Alpert. The ISPD98 circuit benchmark suite. [Online]. Available: http://vlsicad.cs.ucla.edu/~cheese/ispd98.html

[30] G. Karypis. METIS: Family of multilevel partitioning algorithms. [Online]. Available: http://www-users.cs.umn.edu/~karypis/metis/index.html

[31] A. Caldwell, A. B. Kahng, and I. Markov. Ucla/abkgroup physical design tools. [Online]. Available: http://nexus6.cs.ucla.edu/software/PDtools/