Faster and Better Spectral Algorithms for Multi-Way Partitioning

Jan-Yang Chang, Yu-Chen Liu, and Ting-Chi Wang
Department of Information and Computer Engineering
Chung Yuan Christian University
Chungli, Taiwan

Abstract

In this paper, two faster and better spectral algorithms are presented for the multi-way circuit partitioning problem with the objective of minimizing the Scaled Cost. As pointed out in [3], the problem can be approximately transformed into the vector partitioning problem by mapping each circuit component to a multi-dimensional vector. The common key idea of our two algorithms for solving the vector partitioning problem is to first treat the set of vectors as a cluster, and then repeatedly select a cluster, which gives the maximum cost improvement among all the current clusters, and partition it into two new clusters. The bipartitioning process is continued until the number of clusters is equal to the required number of partitions. The experimental results indicate that the two algorithms significantly outperform MELO+DP-RP [3] in both the run time and partitioning result.

1. Introduction

Circuit partition has played an important role at different levels of VLSI design [9,10]. An often adopted type of approaches to circuit partitioning employs the *iterative improvement* technique in which the local perturbation of the current solution is done repeatedly until no further improvement can be achieved [5,8,11,12]. This type of approaches is criticized for not being able to fully capture the global information of the circuit.

Another type of approaches called *spectral partitioning* has been recently received a lot of attention [1,2,3,4,6,7]. The basic idea behind this type of approaches is to first transform the circuit partitioning problem into the graph partitioning problem. After the graph is constructed, spectral methods for graph partitioning are carried out in the following two steps. The first step is to geometrically embed the graph onto the one- or multi-dimensional real space. The second step is to partition the graph vertices into the required number of partitions using the geometric embedding information, such that without violating any given constraints, the given cost function is optimized. The experimental results indicate that spectral methods generate better partitioning results than iterative improvement methods [1,2,3,4,6].

One of the most recent and important theorems for spectral graph partitioning states that if in the first step *all* the eigenvectors of the *Laplacian* matrix of the graph are generated and properly *scaled*, then the second step can be reduced to the *vector partitioning* problem [3]. However, since any *efficient* spectral method cannot afford to generate all the eigenvectors due to the expensive run time, the method presented in [3] generates only up to 10 eigenvectors. To solve the vector partitioning problem, the authors in [3] developed a heuristic called MELO to arrange the vectors in a linear ordering, and then applied a dynamic programming algorithm called DP-RP [2] to split the ordering and generate the final restricted partitioning solution. Although MELO also generates the scaled eigenvectors and constructs an instance of the vector partitioning problem, to simplify our presentation, MELO is referred to as only the part that generates a linear ordering. Hence the method in [3] for solving the vector partitioning problem is referred to as MELO+DP-RP in this paper.

In this paper, we present two faster and better spectral algorithms for the multi-way circuit partitioning problem with the objective of minimizing the *Scaled Cost* (which is a generalization of the two-way *ratio* objective [2]). The common key idea of the two algorithms is to solve the vector partitioning problem by first treating the set of vectors as a cluster, and then repeatedly selecting a cluster, which gives the *maximum cost improvement* among all the current clusters, and partitioning it into two new clusters. The bipartitioning process is continued until the number of clusters is equal to the required number of partitions. The experimental results indicate that the two algorithms generate much better partitioning results in much less run time than MELO+DP-RP.

2. Preliminaries and Problem Formulation

The netlist of a given circuit is represented by a hypergraph H which is *approximately* transformed to a weighted graph $G(V, E)$. The set $V = \{v_1, v_2, ..., v_n\}$ is the vertex set consisting of n vertices, and the weighted edge set E is represented by an $n \times n$ adjacency matrix $A = (a_{ij})$, where $a_{ij} > 0$ if the edge

(v_i, v_j) exists, and $a_{ij} = 0$ otherwise. Let $\deg(v_i) = \sum_{j=1}^{n} a_{ij}$ denote the degree of v_i, and let the $n \times n$ degree matrix D be given by $D = (d_{ij})$, where $d_{ii} = \deg(v_i)$, and $d_{ij} = 0$ if $i \neq j$. Let the $n \times n$ *Laplacian* matrix Q of G be defined to be $D - A$. Let $\lambda_1, \lambda_2, \cdots, \lambda_n$ denote the n eigenvalues of Q, and $\vec{\mu}_1, \vec{\mu}_2, ..., \vec{\mu}_n$ denote their corresponding n-dimensional eigenvectors. We assume that $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, and $\vec{\mu}_1, \vec{\mu}_2, ..., \vec{\mu}_n$ are *normalized* such that for each j, where $1 \leq j \leq n$, ${\mu_j}^T \vec{\mu}_j = 1$. For each d, where $1 \leq d \leq n$, let $U_d = (\mu_{ij})$ denote the $n \times d$ eigenvector matrix whose d columns are $\vec{\mu}_1, \vec{\mu}_2, ..., \vec{\mu}_d$, respectively.

Given a positive integer k, a k-way partitioning of G is to divide the set of vertices of G into a collection of k disjoint partitions, denoted $P^k = \{C_1, C_2, ..., C_k\}$, where each partition C_h, $1 \leq h \leq k$, is a subset of V, $V = \cup_{h=1}^{k} C_h$, and each vertex is in exactly one partition. For a given P^k, the following corresponding terms are defined. The $n \times k$ *assignment matrix* is $X = (x_{ih})$, where $x_{ih} = 1$ if $v_i \in C_h$, and $x_{ih} = 0$ otherwise. For each h, where $1 \leq h \leq k$, the *indicator vector* $\dot{X}_h$ (corresponding to partition C_h) is the hth column of X. The $n \times k$ *projection matrix* is $\Gamma = (\alpha_{jh})$, where $\alpha_{jh} = {\vec{\mu}_j}^T \dot{X}_h$.

With the above notations and definitions, the k-way (generalized) *ratio-cut* problem for G is to find a P^k such that the resulting cost $f(P^k)$, defined to be $\sum_{h=1}^{k} (E_h / |C_h|)$, is as small as possible, where $E_h = \sum_{v_i \in C_h} \sum_{v_j \notin C_h} a_{ij}$. If we let $H \geq \lambda_n$ be some constant, then the $n \times d$ *scaled* eigenvector matrix $V_d = (v_{ij})$ is defined from U_d with $v_{ij} = \mu_{ij} \sqrt{H - \lambda_j}$. Let $\dot{y}_i^d$ denote the ith row of V_d, where $1 \leq i \leq n$. It is not hard to verify $kH - f(P^k) = \sum_{h=1}^{k} \sum_{j=1}^{n} \alpha_{jh}^2 ((H - \lambda_j) / |C_h|)$.

With the scaled eigenvector matrix V_d, the corresponding vector partitioning problem is formulated as follows. Given the positive integer k, and the set $Y = \{\dot{y}_1^d, \dot{y}_2^d, ..., \dot{y}_n^d\}$ of n d-dimensional vectors, the k-way vector partitioning problem is to find a partitioning of Y into a set $S^k = \{S_1, S_2, ..., S_k\}$ of k disjoint subsets such that each $\dot{y}_i^d$ $(1 \leq i \leq n)$ is contained in exactly one S_h $(1 \leq h \leq k)$ and $g(S^k) = \sum_{h=1}^{k} (\| \dot{Y}_h \|^2 / |S_h|)$ is as *large* as possible, where $\dot{Y}_h = \sum_{\dot{y} \in S_h} \dot{y}$, and $\| \vec{Y}_h \|$ denotes the length of $\vec{Y}_h$. By considering each graph vertex v_i as the vector $\dot{y}_i^d$, we say that a graph partitioning $P^k = \{C_1, C_2, ..., C_k\}$ corresponds to a vector partitioning $S^k = \{S_1, S_2, ..., S_k\}$ if and only if $v_i \in C_h$ whenever $\dot{y}_i^d \in S_h$. It has been proved that if P^k corresponds to S^k, and $d = n$, then we have $kH - f(P^k) = g(S^k)$ [3]. This statement implies that a k-way ratio-cut *graph* partitioning problem can be transformed to a k-way vector partitioning problem after generating *all* the scaled eigenvectors of the Laplacian matrix of the graph. However, due to the consideration of the time complexity, the authors in [3] generated only up to d eigenvectors corresponding to $\vec{\mu}_2, \vec{\mu}_3, ..., \vec{\mu}_{d+1}$, respectively, where $1 \leq d \leq 10$. (Note that $\vec{\mu}_1$ can be ignored.)

In this paper, we consider the multi-way *circuit* partitioning problem with the *Scaled Cost* as the objective. Given a hypergraph H representing the circuit netlist, and a positive integer k, the problem is to find a k-way partitioning $P^k = \{C_1, C_2, ..., C_k\}$ such that the resulting Scaled Cost [2], defined to be $(1/(n(k-1))) \sum_{h=1}^{k} (e_h / |C_h|)$, is as small as possible, where e_h denotes *the number of external nets crossing the boundary of partition* C_h.

3. Two New Spectral Algorithms

The multi-way circuit partitioning problem considered in this paper will be *approximately* solved using spectral methods. To simplify the problem, we assume that the hypergraph H has been transformed to a graph by using the *clique* model and adding weight $(4/(p(p-1))) \times ((2^p - 2)/2^p)$ to each possible edge in a p-pin net. (This transformation method was also adopted by MELO+DP-RP.) In addition, the set of d scaled n-dimensional eigenvectors (corresponding to $\lambda_2, \lambda_3, \cdots, \lambda_{d+1}$, respectively) is assumed to have been generated and re-arranged as the set $Y = \{\vec{y}_1^d, \vec{y}_2^d, ..., \vec{y}_n^d\}$ of n d-dimensional vectors, where

d is a user-specified constant. Now the problem remained to be solved is transformed into the vector partitioning problem whose objective is to find a *k*-way partitioning $S^k = \{S_1, S_2, ..., S_k\}$ of *Y* such that the resulting cost $(1/(n(k-1)))\sum_{h=1}^{k}(\| \vec{Y}_h \|^2 / |S_h|)$ is as *large* as possible, where $\dot{Y}_h = \sum_{\vec{y} \in S_h} \dot{y}$. The cost function can be further simplified as $\sum_{h=1}^{k}(\| \dot{Y}_h \|^2 / |S_h|)$ since *n* and *k* both are fixed for a given instance of the problem. Once S^k is obtained, the corresponding P^k can be generated, and the corresponding Scaled Cost can be calculated.

Form now on, we will only focus on the *k*-way vector partitioning problem described above, and present two new algorithms, called Algorithms I and II, for the problem. Throughout the rest of this paper, the terms vector and vertex will be used interchangeably, and the notations *n*, *d*, and *k* will be used to denote the number of vectors, the number of dimensions of a vector, and the required number of partitions, respectively.

3.1 Algorithm I

Algorithm I first treats all the vectors as a global cluster, and then repeatedly selects a "best" cluster from all the current clusters, and uses a bipartitioning algorithm called SPLIT to partition the cluster into two new disjoint clusters. The repetition terminates when the number of clusters is equal to the required number of partitions. A cluster is said to be *best* if among all the current clusters, it gives the *maximum cost improvement* after being bipartitioned. Let *S* denote any cluster, and $D(S)$ denote the cost contributed by *S*. The $D(S)$ is defined to be $(1/|S|)\| \sum_{\vec{y} \in S} \dot{y} \|^2$. Suppose that *S* is partitioned into two clusters S_a and S_b each contributing the costs $D(S_a)$ and $D(S_b)$, respectively. Then the cost improvement due to the bipartitioning of *S* into S_a and S_b is defined to be $\Delta D(S) = D(S_a) + D(S_b) - D(S)$.

Next, let us describe how *S* is partitioned into S_a and S_b by the SPLIT algorithm. Let v_a be the vector in *S* which has the *maximum* length, and v_b be the vector in *S* which produces the *minimum* resulting length after adding it to v_a. Initially, v_a is put into S_a, and v_b is put into S_b. Then, each remaining vector in *S* is considered *sequentially* (i.e., one by one) to be put into either S_a or S_b, but not both. Let *y* denote the current vector in consideration, and let $S_1 = S_a \cup \{y\}$ and $S_2 = S_b \cup \{y\}$ denote the two resulting clusters after adding *y* to S_a and S_b, respectively. If $D(S_1) - D(S_a) \geq D(S_2) - D(S_b)$, then *y* is put into S_a; otherwise it is put into S_b.

To speed up Algorithm I, a binary max heap, which stores all the current clusters and uses $\Delta D(S)$ as the key, is employed to efficiently find a best cluster *S* for bipartitioning. For each cluster *S*, its associated $\Delta D(S)$ must be computed before it is inserted into the heap. In other words, the bipartitioning result of each cluster needs to be *pre-determined exactly once* no matter whether it has a chance to become a best one. (Note that the pre-determined bipartitioning result of each cluster will be saved and used for *actual* bipartitioning if that happens later.) Therefore, whenever a cluster *S* is found to be the currently best one and is actually partitioned into two clusters S_a and S_b, Algorithm 1 will generate the bipartitioning results of S_a and S_b, and calculate $\Delta D(S_a)$ and $\Delta D(S_b)$ before inserting S_a and S_b into the heap.

The time complexity of Algorithm I is analyzed as follows. We first analyze the time taken by the SPLIT algorithm for bipartitioning a cluster. Clearly finding the two seeds v_a and v_b can be done in $O(dn)$ time. Suppose that there is *no particular ordering* when considering putting the vectors of a cluster into the two resulting sub-clusters. Then for each vector, deciding which sub-cluster to put into can be done in $O(d)$ time. Totally, bipartitioning a cluster can be done in $O(dn)$ time.

The whole partitioning hierarchy of Algorithm I can be represented by a binary tree in which there are *k* terminal nodes corresponding to the *k* clusters appearing in the final partitioning solution, and there are $k-1$ internal nodes corresponding to the clusters that are actually partitioned. Each terminal or internal node needs to be partitioned once before it is inserted into the heap. Totally, $k + (k-1) = 2k-1$ clusters need to be partitioned. Since partitioning a cluster takes $O(nd)$ time, and there are $2k-1$ clusters required to be partitioned, the total time spent on partitioning the $2k-1$ clusters is $O(dkn)$.

To find and remove a best cluster from the heap for bipartitioning, and to insert a new cluster into the heap, both can be done in $O(\log k)$ time. The numbers of removal and insertion operations executed on the heap are both $O(k)$. Hence, the total time spent on heap

operations is $O(k \log k)$.

As a result, the total time complexity of Algorithm I is $O(k \log k + dkn)$, which can be rewritten as $O(dkn)$ since $k = O(n)$. This complexity is clearly better than the complexity, i.e., $O((d+k)n^2)$ of MELO+DP-RP since both *d* and *k* are much less than *n* in practice.

3.2 Algorithm II

The second algorithm, called Algorithm II, is a variant of Algorithm I. The only difference between them is that whenever a cluster is considered for partitioning by the SPLIT algorithm, Algorithm II applies MELO to generate a linear ordering for the vectors in the cluster first. Then Algorithm II sequentially puts each vector into one of the two sub-clusters according to the ordering.

4 Experimental Results

Algorithms I and II have been implemented in C language on a PC consisting of a K6-200 CPU and 128M bytes of RAM. The operating system is FreeBSD version 2.2.stable. Eleven ACM/SIGDA benchmark circuits were used as test data. All the parameter settings were the same as those given in [3]. The results obtained by both algorithms are given in Tables 1 and 2, respectively. In each table, the rightmost column reports the average rum time improvement (excluding the time spent on generating eigenvectors) over MELO+DP-RP, and the third column from the right reports the average Scaled Cost improvement over MELO+DP-RP. (Note that the Scaled Cost results in the other columns need to be multiplied by 10^{-5}.) Also, in each table, a negative improvement is preceded by a "–" symbol. The Scaled Cost results of MELO+DP-RP are obtained from [3] and used for comparison, but they are not shown here due to space limitation. On the other hand, the run time results of MELO+DP-RP are observed by ourselves using the same machine although they are not shown due to space limitation. In the bottom row of each table, we also list the average Scaled Cost improvement of each *k*-way partitioning ($2 \le k \le 10$) over MELO+DP-RP.

On the average, Algorithm I achieved 34.39% improvement on the Scaled Cost, and 94.31% improvement on the run time. On the other hand, for Algorithm II, the average improvement on the Scaled Cost increases to 35.74% but the average improvement on the run time decreases to 75.77%. Besides, for each *k*, both the two algorithms also achieve significant improvements on Scaled Cost. Clearly, the two algorithms are able to run faster and generate better partitioning results than MELO+DP-RP.

Acknowledgments

We would like to thank Dr. Charles J. Alpert of IBM for providing us with the source code of MELO+DP-RP. This work was partially supported by the National Science Council of Taiwan under grant NSC-86-2215-E-033-001.

References

[1] C. J Alpert and A. B. Kahng, "Geometric Embeddings for Faster and Better Multi-Way Netlist Partitioning," *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 743-748.

[2] C. J. Alpert and A. B. Kahng, "Multi-Way Partitioning Via Spacefilling Curves and Dynamic Programming," *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 652-657.

[3] C. J. Alpert and S.-Z. Yao, "Spectral Partitioning: The More Eigenvectors, The Better," *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 195-200.

[4] P. K. Chan, M. D. F. Schlag and J. Zien, "Spectral K-Way Ratio Cut Partitioning and Clustering," *IEEE Trans. on CAD 13(9)*, 1994, pp. 1088-1096.

[5] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175-181.

[6] L. Hagen and A. B. Kahng, "Fast Spectral Methods for Ratio Cut Partitioning and Clustering," *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1991, pp. 10-13.

[7] K. M. Hall, "An r-Dimensional Quadratic Placement Algorithm," *Manag. Sci.*, 17(1970), pp.219-229.

[8] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning of Electrical Circuits," *Bell Syst. Tech. J.*, Feb. 1970.

[9] B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, 1988.

[10] S. M. Sait and H. Youssef, *VLSI Physical Design Automation*, McGraw-Hill, 1995.

[11] L. A. Sanchis, "Multiple-Way Network Partitioning," *IEEE Trans. on Computers*, vol. 28, pp. 62-81, 1989.

[12] Y. C. Wei and C. K. Cheng, "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning," *Proc. IEEE Int. Conf. on Computer Aided Design*, 1989, pp. 298-301.

Test Case	Number of Partitions									Total Scaled Cost	Average Improvement of Scaled Cost	Total Run Time	Average Improvement of Run Time
	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$				
19ks	0.47	0.47	1.25	2.05	2.27	2.76	3.65	6.40	7.77	27.09	57.14%	148.39	94.56%
bm1	5.80	7.02	9.42	11.64	14.25	19.56	27.61	38.55	36.19	170.04	-22.71%	14.67	93.54%
p1	12.14	18.56	16.53	21.01	25.33	26.77	34.99	34.92	40.31	230.56	17.54%	13.06	93.57%
p2	2.22	3.06	3.37	3.88	4.27	5.04	4.97	5.80	6.66	39.27	55.53%	169.49	94.19%
t2	1.26	2.15	3.68	4.07	4.90	7.87	8.79	9.65	10.38	52.75	61.49%	51.31	94.27%
t3	1.14	1.97	4.66	5.60	7.32	8.59	8.53	10.19	11.50	59.50	54.33%	47.39	94.28%
t4	0.63	1.81	4.27	6.20	7.42	8.06	8.19	8.64	12.05	57.27	34.85%	41.96	94.34%
t5	1.93	1.94	2.66	3.13	3.29	3.21	3.49	3.81	4.11	27.57	45.59%	124.78	94.51%
t6	0.28	1.77	2.59	3.90	4.59	5.44	8.82	8.89	10.08	46.36	65.54%	56.85	94.31%
balu	3.76	9.64	16.49	18.72	20.99	24.72	32.08	36.05	36.87	199.32	42.19%	11.84	93.68%
struct	7.04	8.83	8.62	10.24	10.74	11.99	12.52	12.42	13.62	96.02	-23.17%	68.85	94.09%
Total	36.67	57.22	73.54	90.44	105.37	124.01	153.64	175.32	189.54	1005.75		748.58	
Average	57.02%	47.22%	44.49%	42.54%	40.08%	35.95%	27.16%	22.50%	21.86%		34.39%		94.31%

Table 1: Results of Algorithm I.

Test Case	Number of Partitions									Total Scaled Cost	Average Improvement of Scaled Cost	Total Run Time	Average Improvement of Run Time
	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$				
19ks	0.47	0.47	1.25	1.73	2.33	2.44	3.40	3.89	4.16	20.14	68.14%	671.50	75.38%
bm1	6.09	6.33	8.99	14.56	23.64	22.83	31.72	49.23	45.67	209.06	-50.87%	67.41	70.30%
p1	12.29	12.32	14.73	19.66	22.52	27.10	28.69	32.60	34.03	203.94	27.06%	42.71	78.98%
p2	2.21	3.06	3.36	3.88	4.27	5.08	4.94	5.63	6.18	38.61	56.27%	708.78	75.71%
t2	1.25	2.15	3.53	4.07	4.41	6.23	9.71	11.69	11.86	54.90	59.92%	255.47	71.47%
t3	1.14	1.96	4.26	6.38	6.46	7.43	9.33	9.16	9.90	56.02	57.00%	203.61	75.42%
t4	0.63	1.81	4.41	6.29	6.72	7.20	7.43	8.51	8.57	51.57	41.34%	168.58	77.26%
t5	1.93	1.95	2.50	3.01	3.29	3.23	3.42	3.81	4.03	27.17	46.38%	570.17	74.92%
t6	0.28	1.77	2.66	4.06	4.76	4.92	5.22	7.32	11.21	42.20	68.63%	279.21	72.03%
balu	3.76	9.64	15.57	19.69	21.48	27.98	30.28	31.76	35.63	195.79	43.22%	39.23	79.07%
struct	4.82	6.94	8.72	8.87	9.95	10.40	11.30	11.95	12.60	85.55	-9.74%	182.60	84.33%
Total	34.87	48.40	69.98	92.20	109.83	124.84	145.44	175.55	183.84	984.95		3189.27	
Average	59.13%	55.36%	47.18%	41.42%	37.55%	35.52%	31.05%	22.40%	24.21%		35.74%		75.77%

Table 2: Results of Algorithm II.