VLSI Design, 2002 Vol. 15 (2), pp. 485–489

A Contraction-based Ratio-cut Partitioning Algorithm*

YOUSSEF SAAB†

Computer Engineering and Computer Science Department, University of Missouri–Columbia, Engineering Building West, Columbia, MO 65211, USA

(Received 2 March 2001; Revised 2 April 2001)

Partitioning is a fundamental problem in the design of VLSI circuits. In recent years, ratio-cut partitioning has received attention due to its tendency to partition circuits into their natural clusters. Node contraction has also been shown to enhance the performance of iterative partitioning algorithms. This paper describes a new simple ratio-cut partitioning algorithm using node contraction. This new algorithm combines iterative improvement with progressive cluster formation. Under suitably mild assumptions, the new algorithm runs in linear time. It is also shown that the new algorithm compares favorably with previous approaches.

Keywords: Ratio cut; Clustering; Contraction; Partitioning

MATHEMATICAL DESCRIPTION

A hypergraph $G(V, E)$ consists of a set of nodes V and a set of nets E. Each net $e \in E$ is a subset of 2 or more nodes in V. Throughout this paper, n and m denote the number of node and nets, respectively. For convenience, the nodes are numbered (named) from 1 to n, and the nets are numbered from 1 to m. A node i has an integer size $S(i)$. However, all nets have unit weights, for a net of weight k can be replaced by k nets of unit weight each. An electrical circuit can be modeled as a hypergraph. A graph is a special case of a hypergraph in which each net links exactly two nodes.

A partition of a hypergraph $G(V, E)$ is an unordered pair (V_1, V_2) of subsets of V such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \varnothing$. The size $S(A)$ of a subset of nodes $A \subseteq V$ is the sum of the sizes of its constituent nodes. A partition (V_1, V_2) is a bisection if $|S(V_1) - S(V_2)|$ is as small as possible. A net e is said to be cut by a partition (V_1, V_2) if it links nodes in V_1 and in V_2, i.e. $e \cap V_1 \neq \varnothing$ and $e \cap V_2 \neq \varnothing$. The cut (cost) of a partition (V_1, V_2) is the subset (number) of nets cut and is denoted by $cut(V_1, V_2)$ ($cost(V_1, V_2)$). The ratio-cut partition problem (RCP) seeks a partition (V_1, V_2) which minimizes

$$rcost(V_1, V_2) = \frac{cost(V_1, V_2)}{S(V_1) \times S(V_2)}$$

Like many other partitioning formulations, RCP is NP-Complete [1]. Therefore, in practice only heuristic approach are considered to tackle this problem.

PREVIOUS WORK

Partitioning problems have been studied by many researchers, and many algorithms have also been reported. Some of these algorithms use general techniques for combinatorial optimization such as Simulated Annealing, Mean Field Annealing, Tabu Search, genetic algorithms, and Stochastic Evolution. Other algorithms are specifically designed for the particular problem being solved. Methods developed specifically for partitioning include Group migration, Clustering, and Spectral methods. Recent approaches include a hybrid genetic algorithm for the ratio-cut partitioning [2] and probability-based partitioning [3]. For a review of partitioning methods up to 1995, see Refs. [4,5].

RATIO CUT PARTITIONING FROM LINEAR ORDERING

Some previous approaches for solving RCP [1,6,7] can be considered variant of an approach that computes a partition from a linear ordering of the nodes of the circuit. Let $v_1 < v_2 < \cdots < v_n$ be an ordering of the nodes of

*Based on "A ratio-cut partitioning algorithm using node contraction," by Y. Saab which appeared in *IEEE Midwest Symposium on Circuits and Systems*, August 1999.

†Corresponding author. Tel.: +1-573-882-4559. Fax: +1-573-882-8318. E-mail: saab@cedars.cecs.missouri.edu

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/1065514021000012093

the circuit. For $1 \leq i < n$, let $V_i = \{v_1, v_2, \ldots v_i\}$ be a subset of V consisting of the first i nodes in the given linear order. Let the partition $(V_k, V - V_k)$ be the one that minimizes the ratio cut among all partitions of the form $(V_i, V - V_i)$, $1 \leq i < n$. Then $(V_k, V - V_k)$ can be used as a solution of RCP.

The first approach we tried to solve RCP was based on the above idea using a modification of the linear ordering algorithm CLP [8]. CLP is an iterative improvement algorithm for linear placement that optimizes the wire-length of the final ordering. It starts with an initial linear ordering and iteratively improves it. CLP was modified to record after each iteration the best ratio cut partition obtainable from the current ordering, and to report at termination the best partition encountered in all iterations. Good results were obtained using this approach. However, the computation time was somewhat excessive. Because CLP optimizes wire-length, it consumes more time to update wire-length after iterative movements of nodes. Computation can be much faster if a direct approach is used. This led to the algorithm CRC which is a direct approach to solve RCP like CLP, it is based on the dynamic contraction (DC) methodology [9]. The DC methodology combines iterative improvement and node contraction into an effective solution strategy for combinatorial problems.

NODE CONTRACTION

There have been reports in the literature that iterative partitioning algorithms perform better if the minimum node degree is large, or alternatively, the density of the graph is large [10–12]. In order to increase the smallest degree, subsets of nodes may be clustered (contracted or compacted or coalesced) to form a single node each. When a subset of nodes X is clustered to form a single node x, the nets incident with node x in the new hypergraph are of the form $(e \cap (V - X)) \cup \{x\}$, where e is a net originally incident with some node in X, and with the provision that nets that are reduced to one node are discarded. All other nets and nodes in $V - X$ remain the same. Naturally, the size of a coalesced node is the sum of the sizes of its constituent nodes. Clustering of nodes to improve the performance of iterative heuristics have been described in Refs. [10,11]. In fact many effective partitioning methods use some form of clustering [12–17]. In Ref. [17], we presented an algorithm for the graph bisection problem that enhances iterative improvement with node contractions. This strategy was also used on other problems. See Ref. [9] for more details.

A NEW RATIO CUT ALGORITHM

This sections describes a new algorithm (CRC) for ratio cut partitioning. CRC is also based on the DC methodology [9] in which an iterative improvement algorithm and a node contraction mechanism cooperated to find better solutions.

The iterative improvement step is similar to computing a ratio cut partition from linear ordering. Given an initial partition (V_1, V_2), nodes in V_1 are consecutively moved to V_2 until only one is left in V_1. After each move, if the ratio cut improves, the current partition replaces the best partition. The current partition also replaces the best partition if it has the same ratio cut cost as the best partition but achieved a better size balance. The same process is repeated with nodes in V_2 consecutively moved to V_1. Note that the two orders in which nodes moved from V_1 to V_2 and from V_2 to V_1 can be combined into one linear ordering of the circuit. Therefore, the best ratio cut partition observed during this node shifting phase can be thought of as obtained from the induced linear ordering. The type and number of node contractions are also determined from this induced ordering. The order in which nodes are moved from one side of the partition to the other is determined by the gain of nodes. The gain of a node is the net decrease (may be negative) in the number of nets cuts if the node is moved to the other side of the partition. The node with highest gain is moved first. To facilitate retrieval of the node of highest gain, a bucket structure as described in Ref. [18] is used with the LIFO scheme as described in Ref. [19]. We have previously used the same implementation in Ref. [17]. To get started, the first partition (V_1, V_2) is randomly generated.

During the iterative improvement step, subsets of nodes are selected for clustering. Nodes that are moved are labeled. After moving a node x, all labeled nodes that are connected to x by a critical net are selected for clustering with x and their labels are cleared. If such nodes were found then the label of x is also cleared. A net is critical for node x if it is removed from the cut after moving x to the other side of the partition. Nodes that are selected for clustering with x are those nodes that were moved before x in the same direction and that contributed positively to the gain of node x. The rational for this clustering scheme is simple. Nodes that are heavily connected to each other tend to migrate together during iterative improvement. Therefore, for each moved node x an attempt is made to cluster it with previously moved nodes that contributed positively to its gain. Note that all critical nets of node x are partly made critical by the movement of the same nodes selected for clustering with node x. Other nodes may have contributed to the criticality of nets connected to x. These nodes are not selected for clustering with x for one of two reasons:

- Their labels were cleared after being previously selecting for clustering with some other nodes. It has been observed [15,17] that early formation of large clusters hinders the optimization process. The above clustering scheme ensures that only few nodes are selected for clustering with each other.
- Their labels were cleared because a new best partition was found after they were moved but before x is

moved. The labels of moved nodes are cleared every time a new partition replaces the currently best one. This is done to guarantee that nodes that are clustered together belong to the same side of the best partition seen so far. The best partition found serves as the initial partition for the improvement step on the clustered hypergraph.

Other nodes connected to x via non-critical nets are not selected for clustering with x for two reasons:

- A non-critical net may be very long. Therefore large clusters may be formed if neighboring node via non-critical nets are included.
- For the purpose of efficiency. If all the labeled nodes connected to node x are searched for, many nodes may be examined too many times. For example, consider the situation in which a single net connects all the nodes of the hypergraph. In this situation during the search for labeled nodes connected to x, all the nodes are examined. Since each node is moved once during one iterative improvement pass, the running time of the algorithm becomes quadratic. By restricting the search to labeled nodes connected to x via critical nets this situation is avoided. A net can become critical during the iterative shifting of nodes from one side of the partition to the other if at some points it becomes cut and its last node is eventually moved to the other side. Because iterative shifting of nodes proceeds in two directions, a net cannot be critical more than two times in one iterative improvement pass.

After one iterative pass many subsets of nodes are selected for clustering. These subsets are clustered all at once and a new clustered hypergraph is formed. The best current partition projects to a partition of the clustered hypergraph because nodes that are clustered together are guaranteed to be on the same side of the best partition.

As long as there are improvement in cost or clustering of the circuit, the whole process is repeated. After that, a new best partition for the flat unclustered hypergraph is obtained, and in case of improvement, we repeat again on the flat hypergraph.

The above description of CRC algorithm can be summarized as follows:

1. Let G be the input hypergraph. Compute a copy H of G.
2. Generate a random initial partition (X_1, X_2) for H and save a copy (B_1, B_2) of this partition. Save a copy (P_1, P_2) of (X_1, X_2) as the best partition of G. The initial partition (X_1, X_2) is generated as follows. First all the nodes are put in X_1 and X_2 is empty. Then a random coin is tossed for each node, and if the outcome is head the node is moved to X_2. This is repeated until X_2 contains $\lfloor n/2 \rfloor$ nodes, where n is the total number of nodes of the hypergraph.
3. Let (V_1, V_2) be a copy of (X_1, X_2), clear labels of all nodes, set $i = 1$, and set list $C = \emptyset$.
4. If V_i contains more than one node, let node x be the highest gain node in V_i. find all labeled nodes connected to x via critical nets. If none is found, then label x. Otherwise, Clear labels of all nodes found and place them together with x on the list C of subsets to be clustered. Move x to $V_{(3-i)}$. If $(rcost(V_1, V_2) < rcost(B_1, B_2))$ or $(rcost(V_1, V_2) = rcost(B_1, B_2))$ and $(|S(V_1) - S(V_2)| < |S(B_1) - S(B_2)|)$ then replace (B_1, B_2) with (V_1, V_2).
5. If V_i contains more than one node then go to step 4.
6. If $i = 1$ then set $i = 2$, let (V_1, V_2) be a copy of (X_1, X_2), clear labels of all nodes, set list $C = \emptyset$, and go to step 4.
7. If $C \neq \emptyset$ then cluster all subsets on list C, project (B_1, B_2) to a partition of the clustered hypergraph H, replace (X_1, X_2) with (B_1, B_2), and go to step 3.
8. If the ratio cut cost of the best partition has improved then replace (X_1, X_2) with (B_1, B_2), and go to step 3.
9. Compute the partition (W_1, W_2) of G that corresponds to the best partition (B_1, B_2) of H.
10. If $rcost(W_1, W_2) < rcost(P_1, P_2)$ then replace (P_1, P_2) with (W_1, W_2), replace H with a copy of G, replace both (X_1, X_2) and (B_1, B_2) with (P_1, P_2), and go to step 3.

Avoiding Quadratic Running Time

In the iterative improvement phase, nodes are moved according to highest-gain-first (HGF) scheme, where the gain of a node is defined to be the decrease in the number of nets cuts if the node is moved to the other side of the partition. In the context of ratio cut partitioning, it may seem more appropriate to define the gain of a node as the net decrease in the ratio cut cost of the partition if the node is moved to the other side of the partition. However with this definition, moving one node from one side of the partition to the other affects the gain of all other nodes and not just the neighbors of the moved node. Therefore, it becomes costly to maintain node gains efficiently: The running time becomes quadratic per each iterative improvement pass. In our chosen definition of gain, only linear time is needed per pass using the well known data structures of Ref. [18]. The experimental results will also show that the quality of the solution is not sacrificed by this choice.

After each node move, the algorithm saves the partition if it improves the best ratio cut cost. Therefore, the partition may be saved many times especially during the early stage of the algorithm after starting with a random initial partition. If the status of each node is saved, then $O(n)$ time is needed each time a partition is saved. Since the current partition may be saved $O(n)$ time per iterative improvement pass, the running time of the algorithm may become quadratic. To avoid this, we keep all the moved nodes since the last time a partition is saved in a list, and we only update the status of the nodes on that list when a better cost partition is found. For small hypergraphs,

TABLE I Ratio cut comparisons

Test	# Nodes	RCut1.0	EIG1-IG	CRC	FREQ	Time(s)
bm1	882	12.73	5.53	5.53	100	0.1
19ks	2844	5.88	6.37	4.58	4	0.5
prim1	833	13.5	13.4	13.4	41	0.11
prim2	3014	5.77	5.22	4.57	100	0.55
test02	1663	19.8	14.7	7.96	21	0.3
test03	1607	14.44	9.92	8.78	7	0.25
test04	1515	11.42	5.85	5.67	93	0.27
test05	2595	6.57	3.12	3.06	74	0.49
test06	1752	7.72	8.26	7.38	46	0.33
%improv		37.7	15.88	x		

the effect of this implementation on the running time is insignificant. However, for large hypergraphs the running time is significantly improved.

EXPERIMENTAL RESULTS

The results of our algorithms are compared to those of RCut1.0 [20] and EIG1-IG [6] algorithms. The results of RCut1.0 and EIG1-IG are those reported in Table IV of Ref. [6]. The results are presented in Table I. Our algorithm (CRC) was run 100 times on a DEC 8400 station using random initial partitions. In Table I, we report the best ratio cut cost found in 100 runs of CRC (column CRC), the number of times the best cost was found (column FREQ), and the average running time per run. The last row shows the cumulative percentage improvements over RCut1.0 and EIG1-IG. The results of CRC show 37.7% improvement over those of RCut1.0 and 15.88% improvement over those of EIG1-IG. Note that except for two cases, CRC obtained the best solution in at 21 of the 100 runs. The total running time for all 100 runs for all the inputs in Table I is less than 5 min of CPU time.

We also compared our algorithm to the MLC algorithm in Ref. [15]. MLC is designed to find balanced partitions and therefore it does not necessarily optimize the ratio cut cost. We use the result of MLC simply as reference points to check the effectiveness of CRC. We also wanted to determine the speed of CRC when run on large inputs as those used in Ref. [15]. The results are presented in Table II. Only the cut is reported in Ref. [15] for the hypergraphs in Table II. We computed a ratio cut cost by assuming that the partitions obtained by MLC are perfectly balanced. The results of CRC show 35.3% improvement over those of MLC. Note that except for six cases, CRC obtained the best solution in at least 21 of the 100 runs. The total running time for all 100 runs for all the inputs in Table II is about 2 h of CPU time. Note that several hypergraphs in Table II have at least 20,000 nodes and one has 1,03,048 nodes. Also note that the ratio cut costs obtained by CRC for several circuits were zero. It turned out that those hypergraphs consist of several connected components as we verified by running a connected components algorithm.

TABLE II Ratio cut comparisons

Test	# Nodes	MLC	CRC	FREQ	Time(s)
Balu	801	16.83	16.88	27	0.13
bm1	882	24.2	5.53	100	0.1
prim1	833	27.1	13.4	41	0.11
test04	1515	8.37	5.67	93	0.27
test03	1607	8.67	8.78	7	0.25
test02	1663	12.58	7.96	21	0.3
test06	1752	7.82	7.38	46	0.33
struct	1952	3.46	3.47	2	0.4
test05	2595	4.21	3.06	74	0.49
19ks	2844	5.14	4.58	4	0.5
prim2	3014	6.12	4.57	100	0.55
s9234	5866	0.46	0	100	0.45
biomed	6514	0.78	0.6	22	1.6
s13207	8772	0.28	0	100	0.7
s15850	10470	0.16	0	100	0.9
ind2	12637	0.41	0.05	51	2.8
ind3	15406	0.4	0	100	3.7
s35932	18148	0.049	0.4	31	5.5
s38584	20995	0.042	0	100	3.1
avqsm	21918	0.106	0.05	2	7.2
s38417	23949	0.0341	0.04	1	7.9
avqlrg	25178	0.08	0.036	1	9
golem3	103048	0.05	0	100	34.4
%improv		35.3	X		

CONCLUSION

We have presented an efficient algorithm for ratio cut partitioning. Several runs with random initial starts may be needed to obtain good results. The experimental results show that in most cases about 5 runs (100 divided by 20) are enough to get the best solution.

References

[1] Wei, Y. and Cheng, C. (1991) "Ratio-cut partitioning for hierarchical designs", *IEEE Transactions on Computer-Aided Design* **10**, 911–921.

[2] Bui, T. and Moon, B. (1998) "GRCA: a hybrid genetic algorithm for circuit ratio-cut partitioning", *IEEE Transactions on Computer-Aided Design* **17**(4), 193–204.

[3] Dutt, S. and Deng, W. (1996) "A probability-based approach to VLSI circuit partitioning", *Design Automation Conference*, 100–105.

[4] Alpert, C. and Kahng, A. (1995) "Recent directions in netlist partitioning: a survey", *Integration—The VLSI Journal* **19**(1–2), 1–81.

[5] Johannes, F.M. (1996) "Partitioning of VLSI circuits and systems", *Design Automation Conference*, 83–87.
[6] Hagen, L. and Kahng, A.B. (1992) "New spectral methods for ratio cut partitioning and clustering", *IEEE Transactions on Computer-Aided Design* **11**, 1074–1085.
[7] Chan, P.K., Schlag, M.D.F. and Zien, J.Y. (1994) "Spectral k-way ratio-cut partitioning and clustering", *IEEE Transactions on Computer-Aided Design* **13**, 1088–1096.
[8] Saab, Y. (1996) "An improved linear placement algorithm using node compaction", *IEEE Transactions on Computer-Aided Design* **15**, 952–958.
[9] Saab, Y. (1997) "Combinatorial optimization by dynamic contraction", *Journal of Heuristics* **3**, 207–224.
[10] Bui, T., Heigham, C., Jones, C. and Leighton, T. (1989) "Improving the performance of the Kernighan–Lin and simulated annealing graph bisection algorithms", *Design Automation Conference*, 775–778.
[11] Goldberg, M. and Burstein, M. (1983) "Heuristic improvement technique for bisection of VLSI networks", *Proceedings of International Conference on Computer Design: VLSI in Computers*, 122–125.
[12] Jones, C., "Vertex and edge partitions of graphs" Ph.D. Thesis-Department of Computer Science, Pennsylvania State University (Pennsylvania).
[13] Barnard, S., Simon, H. "Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems." Report RNR-92-033, Nas Systems Division, Applied Research Branch, NASA Ames Research Center, Mail Stop T045-1, Moffet Field, CA 94035, November 1992.
[14] Karypis, G., Aggarwal, R., Kumar, V. and Shekhar, S. (1997) "Multilevel hypergraph partitioning: application in VLSI domain", *Design Automation Conference*, 526–529.
[15] Alpert, C.J., Huang, J.-H. and Kahng, A.B. (1997) "Multilevel circuit partitioning", *Design Automation Conference*, 530–533.
[16] Behrens, D., Harbich, K. and Darke, E. (1996) "Hierarchical partitioning", *International Conference on Computer-Aided Design*, 470–477.
[17] Saab, Y. (1995) "A fast and robust network bisection algorithm", *IEEE Transactions on Computers* **44**, 903–913.
[18] Fiduccia, C. and Mattheyses, R. (1982) "A linear-time heuristics for improving network partitions", *Proceedings of the 19th Design Automation Conference*, 175–181.
[19] Hagen, L.W., Huang, D.J.H. and Kahng, A.B. (1997) "On implementation choices for iterative improvement partitioning algorithms", *IEEE Transactions on Computer-Aided Design* **16**(10), 1199–1205.
[20] Wei, Y. and Cheng, C. (1989) "Towards efficient hierarchical designs by ratio-cut partitioning", *International Conference on Computer-Aided Design*, 298–301.

Youssef G. Saab received the B.S. degree in computer engineering and the M.S. and the Ph.D. degrees in electrical engineering from the university of Illinois at Urbana-Champaign, Urbana, IL, in 1986, 1988, and 1990, respectively. He is currently an assistant professor in the department of computer science at the university of Missouri-Columbia.From January 1986 to July 1990, he was a research assistant at the Coordinated Science Laboratory, Urbana, IL. His research interests include computer-aided design and layout of VLSI circuits, combinatorial optimization, computational geometry, and graph algorithms.Dr Saab is a member of IEEE, ACM, Tau Beta Pi, Eta Kappa Nu, Phi Kappa Phi, and the Golden Key National Honor Society.