An Improved Two-Way Partitioning Algorithm with Stable Performance

Chung-Kuan Cheng, *Member, IEEE,* and Yen-Chuen A. Wei, *Member, IEEE*

Abstract—**In this paper, we present a two-way partitioning algorithm that significantly improves on the highly unstable results typically obtained from the traditional Kernighan–Lin based algorithms. The algorithm groups highly connected components into clusters and rearranges the clusters into two final subsets with specified sizes. It is known that the grouping operations reduce the complexity, and thus improve the results, of partitioning very large circuits. However, if the grouping is inappropriate, the partitioning results may degenerate. To prevent degeneration, we use a *ratio cut* approach to do the grouping. By a series of experiments based on the trade-off between cut weight and CPU time, we determine the value which controls the resultant number of groups. Good experimental results have been observed for cut weight and CPU time.**

I. Introduction

With modern fabrication technology, very large scale integrated (VLSI) circuitry has become so dense that a single silicon chip may contain tens of thousands of transistors. These on-chip components communicate with each other through physical connections or wires. The first stage of integration is to physically place them relative to each other. The second stage is to globally route the wires connecting each component. As a consequence, the size of many VLSI design problems has become very large and the procedure has become very complicated. Using the strategy of *divide and conquer*, we divide a complex problem into small subproblems, thus reducing the complexity of the original problem dramatically. An ideal partitioning at this stage places highly connected components together, thus shortening the total interconnection distance between all of the components. This approach significantly improves system performance and reduces layout costs.

Given a circuit (network) consisting of a set of modules (nodes) connected by a set of signals (nets), the objective of a two-way partitioning is to divide the whole circuit into two subsets such that the number of signals crossing these two subsets is minimized.

Manuscript received May 14, 1990; revised September 24, 1990. This work was supported by grants from the National Science Foundation (Project MIP-9009260) and from Amdahl, AT&T, Cadence, DEC, Elite, and Hughes under MICRO. This paper was recommended by Associate Editor M. Marek-Sadowska.

C.-K. Cheng is with the Computer Science and Engineering Department, University of California, San Diego, La Jolla, CA 92093.

Y.-C. A. Wei is with Cadence Design Systems, Inc., San Jose, CA 95134.

IEEE Log Number 9100309.

Ford and Fulkerson [6] proposed the *max-flow-min-cut* algorithm, which finds the optimum solution between two subsets of unconstrained sizes in polynomial time. However, experiments in the applications to VLSI designs have shown that the algorithm usually generates two subsets of very uneven size [20]. In applications, the sizes of the two resultant subsets must be balanced because balanced partitions lead to reduced computational complexity in a divide-and-conquer strategy, and only balanced partitions lead to floor plans with reasonable aspect ratios for the layout of each subcircuit.

Unfortunately, the partitioning problem with specified size constraints falls into the class of NP-complete problems [8]. There are various heuristic approaches [1], [7], [10], [11], [16]–[19]. A classical two-way partitioning technique was proposed by Kernighan and Lin [11]. The algorithm starts with a random partitioning and then applies pairwise swapping and iterations between the two subsets with constraints on the subset sizes. Since then, many improvements have been made to this method. Schweikert and Kernighan [18] proposed the use of a net cut model so that the algorithm could handle multipin net cases. Fiduccia and Mattheyses [7] improved this algorithm by reducing time complexity $O(P)$ with respect to the number of pins, P, in the circuit. Krishnamurthy [10], [16] further improved the algorithm with look-ahead ability. Recently, Sechen and Chen [17] have achieved excellent results with a new multipin net model.

Motivation

The Kernighan–Lin algorithm has been observed to be highly sensitive to the choices of initial partition, and the final results vary significantly as a consequence of different starting points [10]. If we consider a run to be one trial of the Kernighan–Lin algorithm on a given initial partition, then many such runs on randomly generated initial partitions are needed to avoid becoming trapped into local minimal solutions. This process is very time consuming. Furthermore, the probability of finding the optimum solution in a single trial drops exponentially as the size of the circuit increases [11]. In other words, if the circuit size is large, then the heuristic Kernighan–Lin algorithm is unlikely to jump out of local minima, and so the optimum solution will not be found.

Bottom-up clustering techniques are often used to reduce the size of the problem. Krishnamurthy [10] applied

a simple clustering approach before initial partitioning, while Ng *et al.* [14] proposed a clustering based on Rent's rule, and Bui *et al.* [3] used a compaction heuristic. They all reported immediate improvements and more stable final results than those produced by random initial partitioning. However, bottom-up clustering techniques usually lack a global view of the entire network. Fig. 1 shows an example of 12 nodes with weights attached to the edges. The weight is 1 if not specified. Fig. 1(a) shows an optimum partition with cut weight 6.6. If a *maximum matching* [9], [15] criterion is adopted in a bottom-up clustering approach, then nodes with an edge of weight 1.1 between them will be merged. An optimum partition on the merged nodes yields a cut weight of 18 (Fig. 1(b)). In general, a $2n$ node network having a symmetric configuration as in Fig. 1 will have a cut weight of $n^2/2$ if the maximum matching criterion is applied to perform the clustering, while the optimum solution will have a cut weight of $1.1 \times n$. From this extreme case, we can claim the following theorem:

Theorem: There is no constant factor of error bound of the cut weight generated by the maximum matching approach, from the cut weight of an optimum partition.

Proof: As shown in the above example, the factor of error bound is $(n^2/2)/(1.1 \times n) = n/2.2$, which is not a constant. Q.E.D.

Kernighan and Lin [11] also suggested that another run of "orthogonal" partitioning might improve the result generated by the first try. Suppose, for example, that a partitioning result (A, B) is obtained from some random initial partition. We divide A into A_1 and A_2, and divide B into B_1 and B_2. Another run of the Kernighan–Lin algorithm can then be applied to either $(A_1 \cup B_1, A_2 \cup B_2)$ or $(A_1 \cup B_2, A_2 \cup B_1)$. If we find a better partition, we can then repeat the procedure. Krishnamurthy [10] has implemented this approach and confirmed its effectiveness. However, achieving the improvement from the previous final result will take more CPU time with the same complexity of the original problem, since the following runs are still applied to the original network. Furthermore, natural clusters formed by highly connected groups [5] exist in circuit layouts. In Kernighan–Lin based algorithms, a specified subset size must be given before generating the initial partition. This preliminary process works against the clustering structures in the circuit since it is impossible to determine cluster sizes before partitioning [20].

These observations lead us to believe that a good top-down grouping technique is essential for efficient partitioning. In order to reduce the size of large circuits, we recursively use a top-down partitioning technique, and divide the whole circuit into small, highly connected groups. Thus, the groups generated by each partitioning run get progressively smaller, and the size of the problem is reduced at every single run. The final step is to rearrange the small groups into two subsets that match the size constraint. We can afford many trials of the rearrangement because the number of groups is relatively small in comparison with the number of modules in the circuit. Also, the probability of getting a near optimum solution is high since the number of groups being generated is small. In this paper, we use the ratio cut as a basic tool to build up our partitioning algorithm. The ratio cut approach removes the constraint on predefined subset size, and tends to identify natural clusters in the circuit [20], thereby yielding good partitioning results.

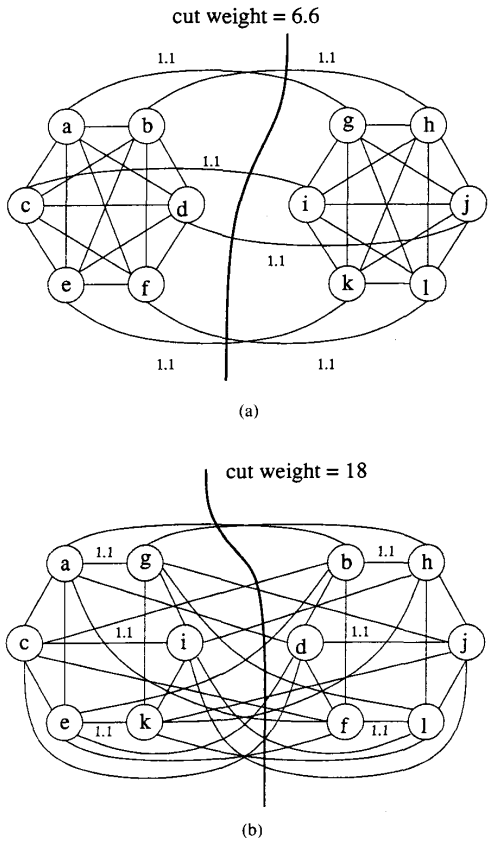


Fig. 1. A 12-node example (default edge weight is 1 if not shown). (a) Result by top-down partitioning. (b) Result by bottom-up clustering.

II. The Ratio Cut Approach and the Algorithm

The ratio cut approach concept, as proposed in [20] and [21], can be described as follows: Given a network $N = (V, E)$, where V is the set of nodes and E is the set of edges, let c_{ij} be the weight of an edge connecting node i and node j. (A, A') denotes a cut that separates a set of nodes A from its complement $A' = V - A$. The *cut weight* is equal to $C_{AA'} = \Sigma_{i \in A} \Sigma_{j \in A'} c_{ij}$. The *ratio* of this cut is defined as $R_{AA'} = C_{AA'}/(|A| \times |A'|)$, where $|A|$ and $|A'|$ denote the cardinalities of subsets A and A' respectively. The *ratio cut* is the cut that generates the minimum ratio among all cuts in the network, i.e. $\min_A C_{AA'}/(|A| \times |A'|)$ $(A \subset V, A \neq \varnothing$, and $A' \neq \varnothing)$.

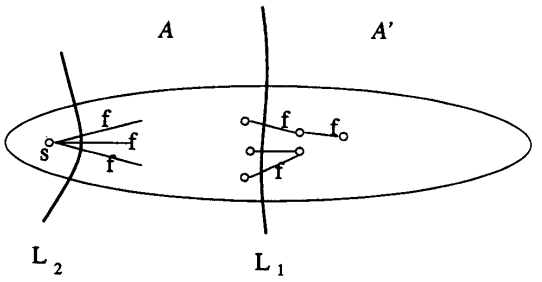


Fig. 2. A random graph with n nodes.

A. The Clustering Property of the Ratio Cut

The clustering property of the ratio cut is best explained by a random graph model [2]. Fig. 2 shows a *uniformly distributed random graph* with n nodes. The probability of having an edge connecting a pair of nodes in this graph is equal to an identical value, f. Consider a cut L_1 in Fig. 2 which partitions the network into two subsets, A and A', with sizes comparable to $\alpha \times n$ nodes and $(1 - \alpha) \times n$ nodes respectively, where $0 < \alpha < 1$. The expected cut weight, $C_{AA'}$, of cut L_1 equals the probability, f, multiplied by the number of possible edges between A and A':

$$E(C_{AA'}) = f \times |A| \times |A'| = f \times \alpha \times (1 - \alpha) \times n^2. \tag{1}$$

On the other hand, if another cut, L_2, in Fig. 2 separates only one node s from the rest of the nodes, the expected number of edges crossing L_2 is

$$E(C'_{\{s\}\{s\}}) = f \times (n - 1). \tag{2}$$

As n approaches infinity, the value of (1) becomes much larger than that of (2), and the probability that $C_{AA'}$ is smaller than $C'_{\{s\}\{s\}}$ approaches 0. If we apply the max-flow-min-cut algorithm to the network, it will most likely generate two subsets of very uneven size, since they naturally give the lowest cut weight value. Therefore, we propose a ratio value, $C_{AA'}/(|A| \times |A'|)$, to balance the sizes on the two subsets. As a consequence, the expected value of this ratio is a constant with respect to different cuts:

$$E(R_{AA'}) = E\left(\frac{C_{AA'}}{|A| \times |A'|}\right) = \frac{f \times |A| \times |A'|}{|A| \times |A'|} = f. \tag{3}$$

Thus, if the edges of the graph are uniformly distributed, then all cuts will have the same ratio value. Therefore, the choice of the cuts does not make a difference in such a uniformly distributed random graph. In a general network, however, different cuts generate different ratios. Cuts that go through weakly connected groups correspond to smaller ratios. The minimum of all cuts according to their corresponding ratios defines the sparsest cut [12], [13], since this cut deviates the most from the cuts of a uniformly distributed graph.

B. The Algorithm to Find a Ratio Cut

Like many other partitioning problems, finding the ratio cut in a general network belongs to the class of NP-complete problems [12], [13]. Therefore, certain heuristic algorithms were proposed in [20] and [21]. We adopt the algorithm in [21] (see the Appendix) as our basic partitioning tool. Because the algorithm runs in linear time, it is fast enough to deal with complicated VLSI circuitry. Also, the algorithm has the full freedom to identify clusters in the circuit without any size constraints.

III. The Stable Two-Way Partitioning Algorithm

A. The Algorithm

First, since the ratio cut algorithm is a good tool for identifying natural clusters in the circuit, we will utilize this algorithm recursively until the circuit is divided into small, highly connected groups. Each small group contains a different number of modules and is of a different size. Next, we will rearrange these small groups into two subsets that match the specified size constraints. The main purpose of this step is to minimize the cut weight between the two subsets. We will apply the Fiduccia–Mattheyses algorithm to the contracted network, with each node corresponding to a group of modules. Because the result from any single run of the Fiduccia–Mattheyses algorithm tends to be erratic, we will apply the algorithm a number of times to obtain a good result. Finally, we will apply a last run of the Fiduccia–Mattheyses algorithm to the original expanded network to fine-tune the final result.

A multipin net network can be defined by using a hypergraph model [4]. Let $N = (V, E)$ represent the network having m modules and n nets, where $V = \{v_1, v_2, \cdots, v_m\}$ and $E = \{e_1, e_2, \cdots, e_n\}$. Each edge, corresponding to a multipin net, is a subset of V, i.e., $e_i \subseteq V$, $|e_i| \geq 2$, $1 \leq i \leq n$. We recursively apply the ratio cut algorithm to N and divide V into the number of small groups, $\tilde{m}$. If we let $\Psi = \{V_i \mid 1 \leq i \leq \tilde{m}\}$ be a set for the collection of the generated groups, the V_i's, then we construct a contracted network $H = (\tilde{V}, \tilde{E})$ as follows. First, we let each element in Ψ be a node in the contracted network, i.e., $\tilde{V} = \{i \mid V_i \in \Psi\}$. Next, for every edge e_j in E, we construct a new edge $\tilde{e}_j$. The edge $\tilde{e}_j$ connects node i in $\tilde{V}$ if e_j connects any modules in V_i, i.e., $\tilde{e}_j = \{i \mid$ there exists $v_k \in V_i$, such that $v_k \in e_j\}$. Finally, we set $\tilde{E} = \{\tilde{e}_j \mid |\tilde{e}_j| \geq 2\}$.

Given a network $N = (V, E)$, an integer g for the number of expected groups, an integer *num_of_reps* for repetition, and *size1*, *size2* for the size constraints of two resultant subsets, the stable partitioning algorithm is as follows:

1) Initialize $\Psi = \{V\}$. Calculate the total circuit size *tsize*.
2) Let V^* be a subset in Ψ such that $|V^*| = \max_{V_i \in \Psi} |V_i|$. While $|V^*| > tsize/g$, repeat (3).
3) Set $\Psi = \Psi - \{V^*\}$. Apply the ratio cut algorithm [21] to V^* to get a cut (A, A') where $V^* = A \cup A'$. Set $\Psi = \Psi \cup \{A, A'\}$.

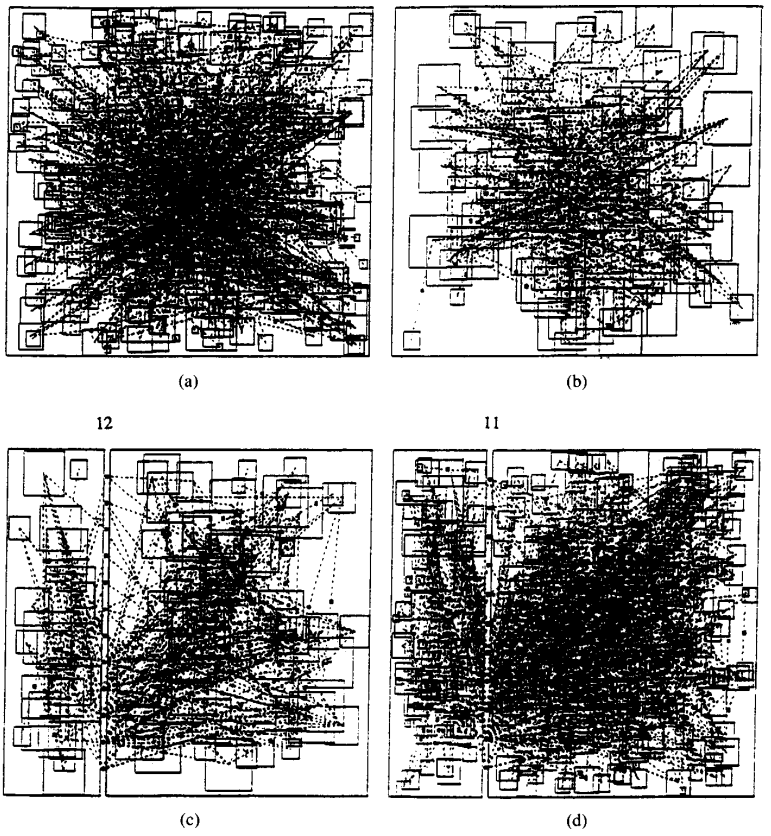


Fig. 3. Different stages in the partitioning process of circuit 8870; g is set to 50 in this case. (a) Before partitioning (#modules = 309, #nets = 349). (b) After recursive partitioning has been done (#modules = 88, #nets = 207). (c) Best result obtained from the contracted circuit. (d) The final result in the expanded circuit.

4) Construct a contracted network $H = (\tilde{V}, \tilde{E}\}$.
5) Apply *num_of_reps* times of the Fiduccia–Mattheyses algorithm to H with the size constraints *size1, size2*.
6) Use the best result from step 5 to the network N. Apply the Fiduccia–Mattheyses algorithm once to N with the size constraints *size1, size2*.

In Fig. 3, we use circuit 8870 from [17] to illustrate the partitioning process. We use squares of varying size to represent modules of varying size in the circuit. Each small dark box represents the center of gravity of a net, connected to all the centers of its modules by dotted lines. The placement of modules is random within the bounding block. Most modules may overlap owing to random placement. The size of the bounding block is exactly equal to the total circuit size. The circuit, as shown in Fig. 3(a), looks quite complicated, containing 309 modules and 349 nets before partitioning. However, after we applied the recursive partitioning in steps 2 and 3, a contracted network was constructed, as shown in Fig. 3(b) ($g = 50$). Since the number of nodes in the contracted network has been reduced to 88, and the number of remaining nets has been reduced to 207, the diagram constructed by step 4 in Fig. 3(b) is much less complex than that in Fig. 3(a). Following the algorithm to step 5, we applied ten runs of the Fiduccia–Mattheyses algorithm to the contracted network. The best result (i.e., cut weight 12) is shown along the separating column in Fig. 3(c). Fig. 3(d) is the result of step 6). The last Fiduccia–Mattheyses algorithm was applied to the original expanded circuit, using Fig. 3(c) as the initial partition. The cut weight is again reduced to 11.

B. Complexity

Given a g value, let $f(g)$ be the number of groups generated after completion of the recursive partition in steps 2 and 3. Since each ratio cut takes $O(p)$ operations [20], where p is the total number of pins, the iterations of steps 2 and 3 requires at most $O(f(g) \times p)$ operations. The complexity of step 4 is $O(p)$, and the complexity of steps 5 and 6 is $O(num\_of\_reps \times p)$. Therefore, the complexity of the whole algorithm is bounded by $O((f(g) +$

num_of_reps) × *p*). In the application, we set *num_of_reps* = 10, and $g = 50$. Our previous experiments have shown that $50 < f(50) < 150$.

The uncertain step in this algorithm was to find a value for g that would force all subsets in Ψ below a certain size before starting the iterations of the Fiduccia–Mattheyses algorithm and that would thereby control the resultant number of groups in the circuit. We performed several experiments to help determine a suitable g value. This is discussed in detail in the next section.

IV. Experimental Results

We tested several algorithms using four circuits: bm1, 19ks, 19kstw, and 26K from the Hughes Aircraft Company; nine benchmarks: PrimGA1, PrimGA2, PrimSC1, PrimSC2, Test02, Test03, Test04, Test05, and Test06 from the Microelectronics Center of North Carolina (MCNC); and two circuits, 8870 and 5655, released in [17]. Table I lists the sizes of all examples. We implemented the Fiduccia–Mattheyses algorithm and the stable two-way partitioning algorithm. In order to compare our results with those obtained from a bottom-up clustering approach, we also implemented the compaction algorithm proposed by Bui *et al.* [3]. The compaction algorithm utilizes the *maximum random matching* to perform the clustering. This clustering approach is one of the most recent methods, and good results have been reported for test cases. Nevertheless, the results obtained from this approach do not imply that the results of the other bottom-up clustering approaches should be similar. All algorithms were written in C and run on a SUN SPARC station 1 machine.

TABLE I
The Sizes of the Tested Examples

Examples	# nets	# modules	# iopads
bm1	904	752	130
19ks	3343	2684	160
19kstw	3731	3079	164
PrimGA1	904	752	81
PrimGA2	3029	2907	107
PrimSC1	904	752	81
PrimSC2	3029	2907	107
Test02	1866	1663	70
Test03	1699	1607	65
Test04	1738	1515	36
Test05	2910	2595	63
Test06	1745	1752	69
8870	349	286	23
5655	843	801	120
26K	29151	25917	360

A. The Number of Groups Controlled by the g Value

The first task was to determine a suitable value for g for all testing cases. We first chose five circuits, 19kstw, Test03, 5655, PrimGA1, and 26K, each of which was selected from a different source and technology. The circuits ranged in size from 800 to 26000. We supplied the circuits with g values ranging from 2 to 800. We made 20 trials for each circuit of a given g value. The maximum size of the final two subsets was set to be three quarters of the total circuit size.

Fig. 4 shows the curves of average cut weights versus g values. The shape of the curves is concave. The reason for this is as follows. When g is small, the average cut weight is high because the partitioning procedure is too coarse. The curves gradually go down and become stable after g equals 25. In our experiments, the best average results occurred while g was in the region from 25 to 100. At values over 100, g becomes so large that the number of subsets in Ψ approaches the total number of modules in the circuit, and the operations tend to behave like the Fiduccia–Mattheyses algorithm. Thus, the curves rise up after g is larger than 150. Notice that the curve for circuit Test03 is relatively flat when compared with the curves of the other circuits. This is a special case because it contains a huge block, one that is almost half of the total circuit size. A partition separating the block from most of the rest of the modules is quite natural. Therefore, almost all g values yield similar results.

We then examined the CPU time in seconds versus g values (excluding circuit 26K), as shown in Fig. 5. Note that all the curves rise up steadily from low CPU time to high CPU time as the value of g increases. Also, notice the curves in the range of g values from 25 to 100 (i.e., the range that gives the best cut weight results in Fig. 4). These experimental results illustrate that we can afford the corresponding CPU time used for different circuits in this range. For example, the best result for the largest circuit, 19kstw, occurs at $g = 50$. We spent 260 s on the average to get this result.

Based on the fact that the stable regions overlapped both between the curves in Fig. 4 and the CPU time consumed curves in Fig. 5, we concluded that 50 is a good value for g for subsequent experiments.

Notice that in the above experiments, *num_of_reps* was set to 10. Fig. 6 shows the resultant cut weights versus *num_of_reps* for circuit PrimGA1. The integer g is set to 50, which always yields a contracted network of around 100 nodes. The deviation in terms of cut weights becomes stable after *num_of_reps* is equal to 6. A *num_of_reps* equal to 10 should be large enough for the five chosen circuits to achieve a good result in the contracted network, using fewer nodes and nets than in the original network.

B. The Stable Algorithm with Size Constraints

After the value for g was determined, we set the maximum size of the final two subsets to three quarters of the total circuit size. Table II compares the results from different algorithms after 20 trials for each case. Under each algorithm, we list the minimum cut weight, the mean value, the standard deviation of the 20 trials, and the average CPU time in seconds used for every example.

In the following, we first investigate the results of the 14 test cases, excluding the 26K circuit, which is much

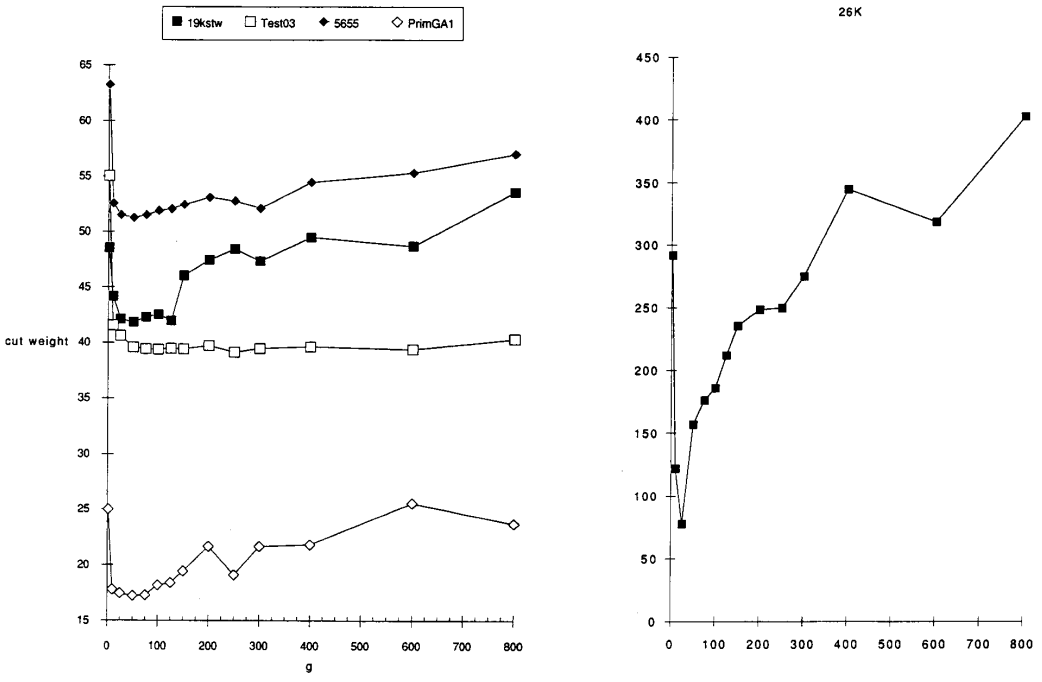


Fig. 4. (a) The curves showing cut weight versus g. (b) The curves showing cut weight versus g for circuit 26K.

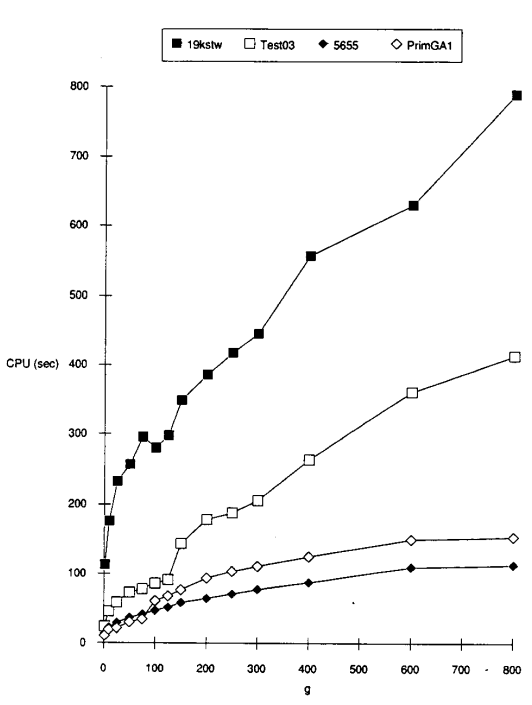


Fig. 5. The curves showing CPU time versus g.

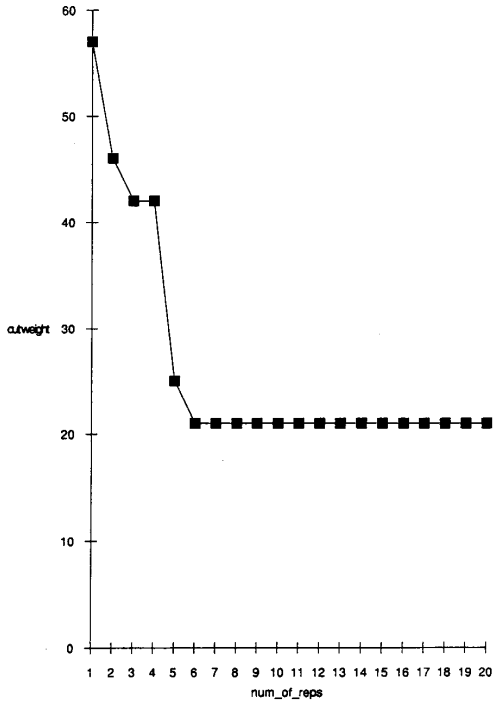


Fig. 6. Cut weight versus num_of_reps.

TABLE II
COMPARISON OF RESULTS OBTAINED FROM THREE ALGORITHMS

	Fiduccia-Mattheyses				Compaction				Stable Algorithm				Improvement (%)	
Examples	min_{fm}	$mean_{fm}$	σ	CPU	min_c	$mean_c$	σ	CPU	min_s	$mean_s$	σ	CPU	$\frac{min_{fm} - min_s}{min_{fm}}$	$\frac{mean_{fm} - mean_s}{mean_{fm}}$
bm1	21	46.40	18.24	5.20	17	48.35	15.06	12.05	17	17.95	1.27	33.85	19	61
19ks	127	173.45	35.85	44.45	93	165.15	26.83	75.95	80	91.00	5.85	228.80	37	48
19kstw	49	90.05	34.51	156.75	53	80.40	22.41	184.95	37	41.80	2.16	260.20	24	54
PrimGA1	29	51.15	16.69	3.10	24	44.00	13.35	7.20	17	17.25	0.91	34.25	41	66
PrimGA2	151	232.85	33.43	23.15	82	124.55	39.36	67.10	77	77.70	2.68	128.30	49	67
PrimSC1	29	54.45	15.59	3.40	17	43.95	18.48	7.30	17	18.20	0.68	29.70	41	67
PrimSC2	113	233.10	49.15	25.27	77	116.40	45.35	69.30	77	77.90	2.84	176.35	32	67
Test02	42	42.80	3.12	20.85	42	42.15	0.49	22.85	42	42.00	0.00	138.70	0	2
Test03	46	88.75	42.29	15.01	39	68.95	32.83	21.85	39	39.55	1.50	66.45	15	55
Test04	44	60.30	17.61	12.15	44	60.70	22.66	24.50	42	43.40	1.05	67.65	5	28
Test05	42	70.70	42.24	33.10	42	95.45	63.26	58.05	42	42.20	0.62	156.50	0	40
Test06	64	91.90	15.99	20.65	58	88.70	25.07	33.30	55	60.85	3.76	142.25	14	34
8870	16	22.85	5.01	2.15	15	21.50	4.22	1.30	11	13.30	1.42	10.25	31	42
5655	57	73.95	12.40	6.95	47	64.75	11.26	7.9	46	51.25	2.86	35.95	19	31
26K	181	410.20	186.55	946.20	229	556.58	180.08	4765.84	65	156.95	64.03	4767.05	64	62

larger than the rest of the test cases. The data in Table II verify the fact that the results of Kernighan-Lin based algorithms are erratic. The average standard deviation for the nets in the first 14 circuits (excluding circuit 26K) derived by Fiduccia-Mattheyses was 24.45 nets. The standard deviation derived by the compaction algorithm was 24.33 nets, about the same range as Fiduccia-Mattheyses. The behavior of the proposed algorithm, by contrast, was very stable. The average standard deviation was only 1.97 nets. Circuit Test02, like circuit Test03, was another special case which contained a huge block with almost half of the total circuit size. Therefore, all of the algorithms we tested yielded similar results.

In most cases, the compaction algorithm performed better than the Fiduccia-Mattheyses algorithm in terms of both the minimum cut weight and the mean value. However, the proposed algorithm generated even better results. For example, in test case PrimGA2, the compaction algorithm improved the minimum cut weight derived by the Fiduccia-Mattheyses algorithm from 151 to 82. The proposed algorithm further reduced it to 77. The mean value also dropped from 232.85 to 124.55 to 77.70. This was a 49% improvement for the minimum cut weight, and a 67% improvement for the mean value, over the Fiduccia-Mattheyses algorithm. The stable partitioning algorithm showed a 25.6% improvement over the best results and a 48.13% improvement over the average results achieved by the Fiduccia-Mattheyses algorithm for all 14 circuits tested. Since we reduced the complexity of the problem at each partitioning run, the stable partitioning algorithm spent six times the CPU time of the Fiduccia-Mattheyses algorithm on average. However, the Fiduccia-Mattheyses algorithm usually takes over 20 runs to produce a good result. Therefore, when the entire procedure is considered, the CPU time savings made possible by the stable partitioning method become apparent.

The last Fiduccia-Mattheyses application in step 6 had a noticeable effect on the final cut weight in the expanded

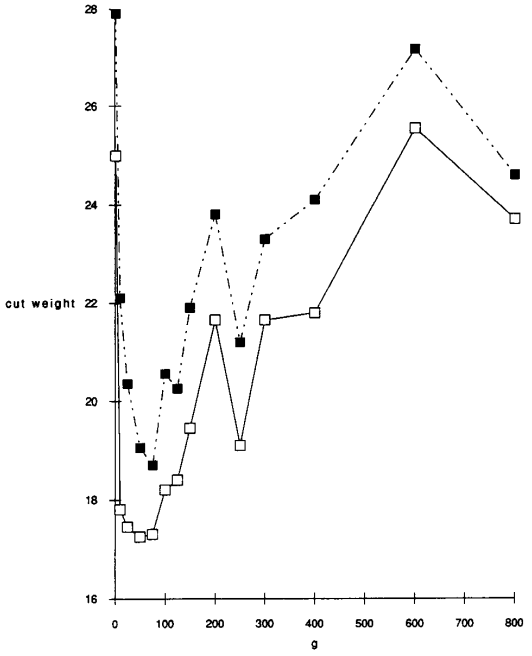


Fig. 7. The curves showing the improvement of the last Fiduccia-Mattheyses algorithm for circuit PrimGA1.

circuit. Fig. 7 shows two curves for circuit PrimGA1 under different g values. The solid curve shows the average cut weight of the 20 tries of the proposed algorithm. The dotted curve shows the average cut weight of the initial partition used by step 6. The average improvement gained from applying the last Fiduccia-Mattheyses algorithm considering all g values for circuit PrimGA1 was 11%.

The 26K test case contains 29151 nets. The best cut weight obtained by the Fiduccia-Mattheyses was 181 nets. The compaction derived a slightly worst result, 229

TABLE III
COMPARISON OF RESULTS OBTAINED FROM TWO ALGORITHMS WITH 1 : 7 SIZE RATIO

Examples	Fiduccia-Mattheyses	Compaction Algorithm	Stable Algorithm
19kstw	33	37	23
PrimGA1	13	11	11
5655	35	31	25

TABLE IV
COMPARISON OF RESULTS OBTAINED FROM TWO ALGORITHMS WITH 1 : 3 SIZE RATIO

Examples	Fiduccia-Mattheyses	Compaction Algorithm	Stable Algorithm
19kstw	49	53	37
PrimGA1	29	24	17
5655	57	47	46

TABLE V
COMPARISON OF RESULTS OBTAINED FROM TWO ALGORITHMS WITH 3 : 5 SIZE RATIO

Examples	Fiduccia-Mattheyses	Compaction Algorithm	Stable Algorithm
19kstw	85	71	60
PrimGA1	35	34	34
5655	48	52	47

TABLE VI
COMPARISON OF RESULTS OBTAINED FROM TWO ALGORITHMS WITH 1 : 1 SIZE RATIO

Examples	Fiduccia-Mattheyses	Compaction Algorithm	Stable Algorithm
19kstw	2101	1462	324
PrimGA1	557	480	198
5655	483	386	180

nets, which may be caused by matching wrong pairs of nodes in this large test case. The proposed algorithm achieved a cut weight of 65 nets, which was 36% of the cut weight obtained by the Fiduccia–Mattheyses. Each try of the algorithm took less than one and a half hours.

C. The Effect of Various Subset Size Ratios

We also investigated the effect of different size ratios for the two final subsets. We tried various ratios from 1 : 7 to 1 : 1. The best results after 20 trials for circuits 19kstw, PrimGA1, and 5655 are listed in Tables III through VI. Note that the results in Table VI are much worse than the results in Tables III–V for all different approaches. The partition results in Table VI tend to be trapped in local optimal solutions because of the strict 1 : 1 size constraint. The stable algorithm still performs much better in all the cases. Fig. 8 shows how the average cut weight obtained by the stable algorithm increases monotonically as the subset sizes become more balanced.

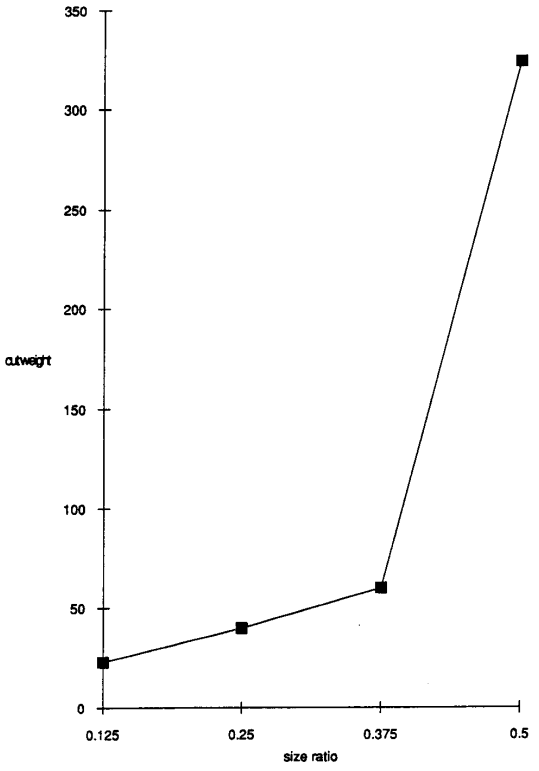


Fig. 8. Cut weight versus different size ratio for circuit 19kstw.

V. CONCLUSION

We have proposed an improved two-way partitioning algorithm whose behavior is extremely stable for all circuits tested. Using the concept of the ratio cut to identify natural clusters in the circuit, the algorithm recursively divides the entire circuit into small, highly connected groups. We conducted a series of experiments in order to determine a suitable number of groups in the circuit. In the final step, we rearranged the groups into two subsets that matched the specified size constraints. We can afford many trials of the rearrangement because the number of groups is relatively small when compared with the number of modules in the circuit.

The stable partitioning algorithm improved the cut weight by an average of 48.13% over the Fiduccia–Mattheyses algorithm for the 15 tested circuits. Furthermore, since the algorithm takes advantage of reducing complexity at each partitioning run, it spends at most six times the CPU time of the Fiduccia–Mattheyses algorithm. Since at least 20 runs of the Fiduccia–Mattheyses algorithm are usually needed to obtain a good result, it is

clear that the proposed algorithm uses much less CPU time when the procedure is considered as a whole.

In the future, we expect to extend our two-way partitioning algorithm to handle multiway partitioning. We will also investigate ways to apply this algorithm to the placement problem. We recommend further experiments with other bottom-up clustering approaches for comparison.

Appendix

The ratio cut algorithm in [21] consists of three major phases: (1) initialization, (2) iterative shifting, and (3) group swapping. These phases are described in detail below.

A. Initialization

Input: a network $N = (V, E)$.
Output: an initial partitioning.

1) Randomly choose a module s in V. Find the other seed module t at the end of a longest path by a breadth-first search starting from s. Let $X = \{s\}$, and $Y = V - \{s, t\}$.
2) Choose a module i in Y whose movement to X will generate the best ratio among all the other competing modules. Move module i from Y to X; update $X = X \cup \{i\}$, and $Y = Y - \{i\}$.
3) Repeat step 2 until $Y = \varnothing$.
4) Repeat steps 2 and 3 with $X = \{t\}$ and $Y = V - \{s, t\}$ until $Y = \varnothing$.
5) The cut giving the minimum ratio found in the procedure forms the initial partitioning.

B. Iterative Shifting

We define a *right (left) shifting operation* as a shifting of the modules with the best resultant ratio value from $s(t)$ toward $t(s)$.

Input: an initial partitioning found in the direction from s to t.

Output: a new partitioning if the ratio of the initial partitioning has been reduced.

1) Repeat right shifting operations until all the modules are exhausted.
2) Choose the minimal ratio value obtained in step 1. If the new ratio value is reduced from step 1, then the cut producing this ratio forms a new starting partition; otherwise, output the previous partition and exit the process.
3) Repeat steps 1 and 2 with left shifting operations.
4) Repeat steps 1–3.

C. Group Swapping

We define the *ratio gain* $r(i)$ of a module i to be the decrement of ratio values if module i (except the two seeds s and t) is moved from its current subset to the other.

Input: a partitioning from the iterative shifting phase.
Output: the final partitioning result.

1) Calculate the ratio gain $r(i)$ for every module i, and set all modules to the "unlocked" state.
2) Select an unlocked module i with the largest ratio gain from two subsets.
3) Move module i to the other side, and lock it.
4) Update the ratio gains of the remaining affected and unlocked modules.
5) Repeat steps 2–4 until all modules are locked.
6) If the largest accumulated ratio gain during this process is positive, swap the group of modules corresponding to the largest gain.
7) Repeat steps 1–6 until the accumulated ratio gain in step 6 is nonpositive.

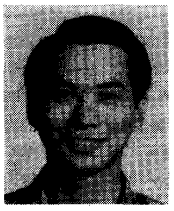
Acknowledgment

The authors would like to thank the reviewers for their helpful suggestions.

References

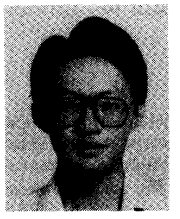
[1] J. Blanks, "Partitioning by probability condensation," in *Proc. ACM/IEEE 26th Design Automat. Conf.*, 1989, pp. 758–761.
[2] B. Bollobas, *Random Graphs*. New York: Academic Press, 1985, pp. 31–53.
[3] T. Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the performance of the Kernighan–Lin and simulated annealing graph bisection algorithms," in *Proc. ACM/IEEE 26th Design Automat. Conf.*, 1989, pp. 775–778.
[4] M. Burstein, "Analysis of a network partitioning technique," in *IEEE Int. Symp. Circuits Syst. Dig.*, vol. 2, 1982, pp. 477–480.
[5] W. E. Donath, "Logic partitioning," in *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, Eds. Menlo Park, CA: Benjamin–Cummings, 1988, pp. 65–86.
[6] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.
[7] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE 19th Design Automat. Conf.*, 1982, pp. 175–181.
[8] M. R. Garey, D. S. Johnson, and L. Stockmeyer, *Some Simplified NP-Complete Graph Problems*, Theoretical Computer Science, 1976, pp. 237–267.
[9] M. Khellaf, "On the partitioning of graphs and hypergraphs," Ph.D. dissertation, Indus. Engineering and Operations Research, Univ. of California, Berkeley, 1987.
[10] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438–446, May 1984.
[11] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.
[12] T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms," in *Dig. IEEE Symp. Foundations of Computer Science*, 1988, pp. 422–431.
[13] D. W. Matula and F. Shahrokhi, "The maximum concurrent flow problem and sparsest cuts," Tech. Rep., Southern Methodist Univ., 1986.
[14] T. Ng, J. Oldfield, and V. Pitchumani, "Improvements of a mincut partition algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 470–473.
[15] C. H. Papadimitriou, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[16] L. A. Sanchis, "Multi-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 62–81, Jan. 1989.
[17] C. Sechen and D. Chen, "An improved objective function for mincut circuit partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1988, pp. 502–505.
[18] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. 9th Design Automat. Workshop*, 1972, pp. 57–62.

[19] Y. Saab and V. Rao, "An evolution-based approach to partitioning ASIC systems," in *Proc. ACM/IEEE 26th Design Automat. Conf.*, 1989, pp. 767–770.
[20] Y.-C. Wei and C.-K. Cheng, "Toward efficient hierarchical designs by ratio cut partitioning," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 298–301.
[21] Y.-C. Wei and C.-K. Cheng, "Ratio cut partitioning for hierarchical designs," *Tech. Rep.* CS90-164, Univ. California, San Diego, Jan. 1990.



Chung-Kuan Cheng (S'82–M'84) received the B.S. and M.S. degrees in electrical engineering from National Taiwan University and the Ph.D. degree in electrical engineering and computer sciences from University of California, Berkeley, in 1984.

From 1984 to 1986 he was a senior CAD engineer at Advanced Micro Devices Inc. Since 1986 he has been an Assistant Professor in the Computer Science and Engineering Department, University of California, San Diego. His research interests include network optimization and design automation on microelectronic circuits.



Yen-Chuen A. Wei (M'91) was born in Taiwan, Republic of China, on September 27, 1959. He received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1982 and the M.S. and Ph.D. degrees in computer science from the University of California, San Diego, in 1987 and 1990, respectively.

He is currently a senior member of technical staff at Cadence Design Systems, Inc., San Jose, CA, where he works on the design and development of VLSI CAD systems. His research interests include hierarchical partitioning, floor planning, and placement and routing algorithms for automatic VLSI design tools.

Dr. Wei is a member of the ACM Special Interest Group in Design Automation.