A Simple Yet Effective Technique for Partitioning

Hyunchul Shin, *Member, IEEE,* and Chunghee Kim

Abstract— **Partitioning is an important step in the top-down design of large complicated integrated circuits. In this paper, a simple yet effective partitioning technique is described. It is based on the clustering of "closely" connected cells and the gradual enforcement of size-constraints. At the beginning, clusters are formed in the bottom-up fashion to reduce the problem size. Then the clusters are partitioned using several different parameters to find a good starting point. The best result achieved during the cluster partitioning is used as the initial solution for the lower level partitioning. The gradual constraint enforcement technique is used to cope with the local minimum problems. It allows cells or clusters to move with more freedom among the subsets during earlier iterations and thus may effectively find a near optimum solution. Several experimental results show that the new partitioning technique produces favorable results. In particular, our method outperforms the F&M method [1] by more than 60% in the number of crossing nets on the average.**

I. Introduction

As the complexity of VLSI circuits increases, a hierarchical design approach becomes essential to shorten the design period. Partitioning plays an important role in finding the hierarchy of a circuit or a system. Good partitioning can significantly reduce the complexity of a design problem and improve both the performance and the reliability of the system. However, the graph partitioning problem belongs to the class of NP-hard problems [2]. Probably the most well-known heuristic partitioning algorithm is [3], and a more efficient version of it [1] is widely used. We also adopted the iterative improvement technique from [1]. Then we incorporated the clustering technique to efficiently search the large solution space and the gradual constraint enforcement technique to alleviate the solution from being stuck at a local optimum point.

Kerninghan and Lin (K&L) [3] suggested the noted min-cut algorithm, which partitions the nodes of a graph with weights on its edges into two subsets of given sizes so as to minimize the sum of the weights on all edges cut. The K&L algorithm iteratively swaps subsets of selected nodes and takes $O(n^2 \log n)$ for a pass of optimization, where n is the number of nodes. Based on the K&L algorithm, an efficient bisection heuristic was developed by Fiduccia and Mattheyses (F&M) [1] in which a single pass exhibits linear time complexity in the number of pins. Quadrisection [4] is an extension of the above bipartitioning algorithms in which nodes are partitioned into four subsections. Sechen and Chen [5] suggested a modified cost function to minimize the net crossings rather than nets cut and reported 38% average reduction in nets cut when compared with those of the F&M [1] method. However, experimental results show that our method works better with the exact cost function as explained in Section V.

Another method [6] used "level gain" to predict the cost changes when moving each cell and Sanchis [7] adapted this model to multiple-way partitioning. In [8], [9], an evolution-based approach was reported which outperformed the F&M method by 27% and a version of annealing-based algorithm with an efficient annealing schedule [10] by 54% on the average. These methods move or exchange nodes so that the size constraints are always satisfied.

Recently, several authors reported a ratio-cut [11] approach which does not impose hard limits on the subset sizes. In these methods, subset sizes may be significantly different when the cut size can be substantially reduced by doing so. However, ratio-cut may not be used when tight control on the subset sizes is required. In [12], top-down clustering is performed by using the ratio-cut algorithm and then iterative moves are made to enhance the partitioning result. Hagen and Kahng [13] proposed spectral methods in which the second smallest eigenvalue of a matrix derived from the netlist yields a lower bound on the optimal ratio cut partition cost.

To partition a given circuit, we developed a new hierarchical gradual constraint-enforcing partitioning technique. In contrast to other partitioning methods, the size constraints on subsets are not enforced at the beginning to allow more freedom to cells to move among subsets. Then the size constraints are enforced gradually to the subsets and the sum of cell sizes to be moved gets more restricted. The partitioning is performed in two levels of hierarchy. At the cluster-level, clusters are formed and partitioned several times by using several different sets of parameters. The best solution of the cluster level partitioning is used as the initial partitioning at the cell level. This partitioning technique generated over 60% better solutions in cut size for the examples we tried when compared with the average results of the F&M method [1]. Our method produced 4% better results on the average over the primal-dual method [14].

In Section II, the overall partitioning algorithm based on the hierarchical gradual constraint-enforcing method is described. In Section III, the clustering technique is presented. In Section IV, the gradual constraint-enforcing partitioning (GCEP) algorithm is described. The GCEP algorithm is used both for the cluster partitioning and for the cell partitioning. In Section V, several experimental results for benchmark circuits are presented to illustrate the effectiveness of our two-level hierarchical GCEP (HGCEP). Finally, conclusions are summarized in Section VI.

Manuscript received September 15, 1992; revised April 5, 1993. This work was supported in part by Samsung Electronics Co.

The authors are with the Department of Electronics Engineering, Han-Yang University, Korea.

IEEE Log Number 9210701.

II. Overall Algorithm

The hierarchical GCEP (HGCEP) algorithm is based on the iterative improvement technique like many other partitioning methods [1], [3]. The results of the iterative improvement methods are more or less dependent on the initial solution from which the improvements begin. For example, a significant cut-size reduction can be obtained by applying the F&M [1] algorithm to hundreds of randomly generated initial partitions and then by choosing the best solution [14]. However, the CPU time is increased as many times as the number of runs. GCEP appears to be less dependent on the initial partitioning than F&M. However, it's a good idea to use a clustering-based initial partitioning rather than a random one since closely coupled cells should be partitioned into the same subset. To find a "good" initial partition efficiently, we applied the GCEP algorithm to clusters of cells. In the present implementation, clusters are partitoned 50 times from different initial partitions and the best result is partitioned once using GCEP to find the final result. Since cluster partitioning is efficient, a pass of HGCEP takes slightly less than four times as long as a pass of flat level GCEP in CPU time. The clusters are constructed by merging "closely" connected cells as explained in the next section.

The overall algorithm shown in Algorithm 1 is composed of two parts. In the first part, clusters are constructed and partitioned to find a good initial solution. In the second part, flattened cells are partitioned by using the GCEP algorithm. The first part of the algorithm tries to search the solution space as efficiently as possible. The partitioning problem, being an NP-hard problem, does not allow the exhaustive search to find the optimum solution except for very small-sized problems. Hence we incorporated two techniques: the use of hierarchy (clustering) and random initial partitioning for the efficient search for a "good" initial partition.

Algorithm 1 : Overall HGCEP

```
input_data;
cut = infinity;
for( i = 1; i <=K; i++ ) {
   /*num_cl[i] contains the number of clusters
      to be formed at i-th iteration */
   make_cluster(num_cl[i]);
   for( j = 1; j <=M; j++ ) {
      random_initial_partition(j);
      mincut = GCEP();
      if( mincut < cut ) {
            cut = mincut;
            save_result;
      }
   }
}
/* The save_result is used as the initial partitioning
   for cell_level partitioning */
flatten_clusters;
GCEP();
output;
```

TABLE I
Sensitivity of the Algorithm to α

α	0.1/200		0.5/200		1.0/200		2.0/200	
examples	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
PrimSC1	38	45.6	38	38.0	38	38.0	44	45.2
PrimSC2	124	130.8	119	119.0	119	120.6	119	122.6

The random initial partitioning is obtained by randomly partitioning the clusters into the given number of subsets. The desired number of clusters, *num_cl[i]*, is passed as a parameter to the procedure *make_cluster*. Then *make_cluster* constructs clusters by merging pairs of cells or clusters in the decreasing order of "closeness" until the number of clusters is reduced to *num_cl[i]*. The same GCEP algorithm has been used to partition the clusters and the cells. In the current implementation, $K = 5$, $M = 10$, and *num_cl*[1..5] = {*num_cell*/8, *num_cell*/9, *num_cell*/10, *num_cell*/11, *num_cell*/12} are used. Therefore, ten different randomly partitioned solutions are evaluated for each of five different sets of clusters. The *make_cluster* procedure is explained in detail in Section III and the GCEP procedure is described in Section IV.

III. Clustering for Hierarchical Partitioning

The clustering is bottom-up. The closeness of two cells or clusters, C and D, is evaluated by the following formula:

$$closeness(C,D) = num\_cnet(C,D)/ MIN(num\_pin(C), num\_pin(D)) - \alpha * (cl\_size(C,D)/avg\_size)$$

where the variable *num_cnet*(C, D) is the number of common nets between C and D, and *num_pin* (C) is the number of pins in C. The variable *cl_size*(C, D) is the size of the new cluster constructed when C and D are merged. The variable *avg_size* is the average size of clusters. The first term represents the attractive force due to common nets between C and D, and the second term represents the repulsive force to encourage forming uniformly sized clusters. In the current implementation, α = 0.5/200 is used to partition standard-cell and gate-array circuits.

A. Weighting Factor

To find the appropriate range of α and to see the sensitivity of the partitioning algorithm to the parameter, our partitioning algorithm has been run using several values of α and the partitioning results for the primary examples are summarized in Table I. The reason we selected primary1 (PrimSC1) and primary2 (PrimSC2) is because they are popular and they differ significantly in size. To reduce the dependency on the initial solution, we ran our partitioning program five times and listed the best and the average costs in the table.

When α = 0.5/200, our method produced the same cost values for all five random initial partitions for both examples. The cost increases slowly when α increases or decreases from

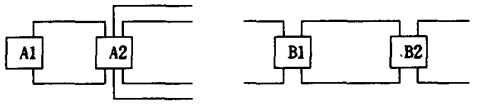


Fig. 1. Closeness of cells (or clusters).

TABLE II
COMPARISONS OF CLOSENESS FUNCTIONS

Z	$f1$ (5 runs)		$f2$ (5 runs)		$f1 + f2$(5 runs)	
examples	best	average	best	average	best	average
PrimSC1	38	38.0	38	41.2	40	41.2
PrimSC2	119	119.0	132	135.8	119	123.0

α = 0.5/200. Fortunately, the sensitivity of HGCEP algorithm to α is relatively low.

B. Attractive Force Due to Common Nets

In this section, we explain that it is reasonable to use the minimum of *num_pin*(C) and *num_pin*(D) in the denominator of the first term of *closeness* than to use *num_pin*(C) + *num_pin*(D). For example, two cases are shown in Fig. 1.

Let's define f_1 and f_2 as follows.

$$f_1(C,D) = \text{num\_cnet}(C,D)/\text{MIN}(\text{num\_pin}(C), \text{num\_pin}(D))$$
$$f_2(C,D) = \text{num\_cnet}(C,D)/(\text{num\_pin}(C) + \text{num\_pin}(D))$$

Then in Fig. 1 it is easy to find out that

$$f_1(A1, A2) = 2/2$$
$$f_1(B1, B2) = 2/4$$
$$f_2(A1, A2) = 2/8$$
$$f_2(B1, B2) = 2/8$$

Hence f_1 prefers merging $A1$ and $A2$ to merging $B1$ and $B2$ while f_2 does not show any preference. A human designer would merge $A1$ and $A2$ first since $A1$ should be merged to $A2$ if it ever merges with another cell.

Closeness functions based on f_1, f_2, and $(f_1 + f_2)$ are compared using the primary circuit examples. As expected, the closeness function based on f_1 generates the best results for both examples, as shown in Table II.

The following algorithm shows the *make_cluster* procedure.

```
Algorithm 2 : make_cluster( num_cl )

/* Each cell is a cluster at the beginning */
for( each cluster C) {
      find_candidate(C);
}
while( number of clusters > num_cl ) {
   find the pair of clusters(C1, C2)
            which have the largest closeness;
   merge_clusters(C1, C2);
   for( each affected cluster C* )
      find_candidate( C* );
}
find_candidate( C ) {
   ov = −∞;
   for( each cluster D having a connection to C ) {
      v = closeness(C, D);
      if( v > ov ) {
         ov = v;
         C.closely_connected = D;
         C.closeness = v;
      }
   }
}
```

IV. GRADUAL CONSTRAINT-ENFORCING PARTITIONING (GCEP)

At the beginning, the size constraints are relaxed and the sizes of partitioned subsets can be severely unbalanced. This allows very flexible movements of nodes among the subsets. After each pass of optimization, the size constraints are updated gradually so that the desired size constraints are enforced at the final pass. At each optimization pass, the movements of cells among subsets are limited only by the size constraints at the pass. Thus the GCEP algorithm has the hill-climbing effect and searches a broader solution space. Also, this method is less dependent on the initial solution when compared with F&M [1]. The GCEP can be used for k-way partitioning. We describe the two-way GCEP first and then show that it can be extended to k-way GCEP.

A. Two-Way GCEP

After the initial partitioning a *cell_gain* is calculated for each cell. *Cell_gain* represents the cost reduction when the cell is moved from the current subset to the other subset. When the cost increases, then the *cell_gain* has a negative value. When a net should not be a crossing net, one may assign a large weight for the net. The cost of a partition is the sum of weights of the crossing nets between two subsets. The cells are moved from a subset to the other in the decreasing order of cell gains. Note that the *cell_gain* is updated after moving each cell. The movement of cells continues until no further movement is possible without violating the size constraints at the iteration. After a pass of cell movements, the size-constraints are tightened and then cells are moved from the other subset to the subset in the decreasing order of cell gains. By repeating these steps, closely coupled cells remain in a subset and near optimum results can be obtained. For cluster partitioning, the clusters are moved just as the cells.

The two-way partitioning algorithm can be summarized as shown in Algorithm 3.

```
Algorithm 3 : GCEP_2_way

input cell list, net list;
evaluate_initial_ gain;
/* Initial_BAL is currently 2*given_BAL */
BAL = Initial_BAL;
while {
```

```
from_group = larger group;
/* min_group_size is the lower bound on the
   subset size */
min_group_size = mid_group_size - mid_group_size
   * BAL;
for( TRUE ) {
   cell_to_be_moved={the cell with the largest gain in
      from_group}
   if(|from_group| - |cell_to_be_moved|
         ≥ min_group_size) {
      move_cell;
   up_date_gain( cell_to_be_moved );
   }
   else
      break
} end for;
/* If BAL is less than given_BAL
   the subset sizes are balanced */
if( BAL ≤ given_BAL ) break;
else
   /* BAL is reduced by the reduction_ratio
   to increase the min_group_size */
   BAL = max( BAL * reduction_ratio, given_BAL );
} end while;
up_date_gain( cell )
{
   for( each net n connected to cell ) {
      for( each cell c1 connected to n )
         evaluate_cell_gain_( c1 );
   } end for
} end up_date_gain
```

Let $|A|$ be the sum of cell sizes in subset A. Then the size constraints are enforced such that $\min(|A_i|, |B_i|) < \min(|A_{i+1}|, |B_{i+1}|)$, for $i = 1, 2, \cdots$. The iteration finishes when the size constraints are satisfied. Formally, the minimum subset size to balance the two subset sizes at iteration i is given by

$$\text{min subset size}(i) = (\text{total size})/2 * (1 - \text{BAL}(i))$$

where the total size is the total sum of cell sizes. The value of $\text{BAL}(i)$ is reduced as i increases and the final value is decided from the desired balance of the two subset sizes.

In the current implementation, the initial value of BAL is twice the value of given_BAL and BAL is reduced to 90% of its previous value after each iteration, i.e., *reduction_ratio* = 0.9. We applied HGCEP to the primary examples for several different values of the *reduction_ratio* and listed the results in Table III. HGCEP was run five times using different initial partitions to reduce dependency to the initial solution. For primary 1, HGCEP produced good results for *reduction_ratio* $\geq$ 0.9. For primary 2, HGCEP produced good results for *reduction_ratio* $\geq$ 0.75.

As long as the relaxed size constraints are satisfied, cells are moved even if the *cell_gain* is negative. This helps the algorithm to avoid local optima. For example, in Fig. 2 (a) and (b), if cell 4 is moved to the right-hand side, cost is increased (*cell_gain* < 0). However, after moving cell 4, cells 3 and 2 can be moved to the right-hand side with a positive *cell_gain*, resulting in a better partition, as shown in Fig. 2 (c).

TABLE III
SENSITIVITY OF HGCEP TO THE *reduction_ratio*

Examples	PrimSC1		PrimSC2	
R_ratio	cost	time/run	cost	time/run
0.95	38,38,38,38,38	46	119,119,119,119,132	631
0.9	38,38,38,44,38	41	119,119,119,119,119	588
0.85	44,44,44,44,44	39	119,119,119,119,119	573
0.8	44,38,44,38,38	38	119,119,119,119,119	568
0.75	44,44,38,44,38	37	119,119,119,119,119	563
0.7	41,42,44,41,44	37	144,137,145,132,137	548
0.6	44,44,46,46,46	36	140,130,132,119,140	545

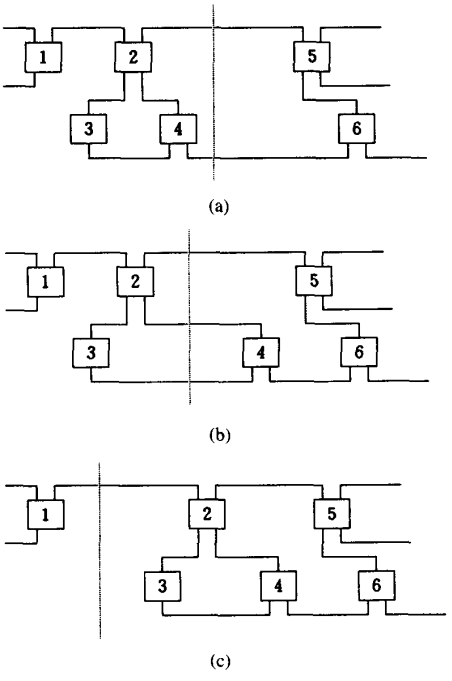


Fig. 2. Hill climbing movement when *cell_gain* < 0.

Even though the standard deviation of costs of GCEP is usually less than that of F&M, still GCEP is dependent on the initial partitioning. Note that HGCEP is not quite dependent on the initial partitioning. In Table IV, the total standard deviation of costs for the primary examples for the F&M, GCEP, and HGCEP methods are shown to be 83.3, 68.0, and 1.3, respectively.

B. K-way GCEP

The GCEP algorithm can easily be extended to solve multiple-way partitioning problems. In this subsection, we describe a new k-way partitioning algorithm using the GCEP technique.

The min-cut bisection algorithm can be used recursively to partition the nodes into k subsets. However, one may want

TABLE IV
DEPENDENCE OF ALGORITHMS ON INITIAL SOLUTIONS

	F&M (20 runs)			GCEP (20 runs)			HGCEP (20 runs)		
Examples	Best	Avg.	Std dev.	Best	Avg.	Std. dev.	Best	Avg.	Std. dev.
PrimSC1	66	79.8	10.7	45	69.9	15.0	38	38.4	1.3
PrimSC2	188	323.6	72.6	181	292.1	53.0	119	119.0	0.0

TABLE V
COMPARISONS OF HIERARCHICAL AND FLAT GCEP

	HGCEP : 1 run		GCEP : 20 runs			
Examples	Cost	Time (s)	Best Cost	Avg. Cost	Max. Cost	Time (s/run)
PrimSC1	38	41	45	69.9	92	12
PrimSC2	119	588	181	292.1	356	152

TABLE VI
EXAMPLES

Example	#IO	#nodes	#nets
Test02	70	1663	1866
Test03	65	1607	1699
Test04	36	1515	1738
Test05	63	2595	2910
Test06	69	1752	1745
PrimGA1	81	752	904
PrimGA2	107	2907	3029
PrimSC1	81	752	904
PrimSC2	107	2907	3029

to partition directly into k subsets. For example, simultaneous four-way partitioning (quadrisection) is more appropriate for two-dimensional layout problems [4]. Naturally, quadrisection can generate superior results than those of recursive min-cut bisections since quadrisection takes care of the two perpendicular (horizontal and vertical) cut lines simultaneously while the three passes of bisections are applied independently.

Modifying the GCEP_2-way to GCEP_K-way is very simple. Since there are k subsets, we need to choose two subsets, *from_subset* and *to_subset* to apply the GCEP. Each subset becomes the *from_subset* in turn. The *to_subset* is determined so that the cell gain is maximized when a cell is moved from *from_subset* to *to_subset*. Other parts of the GCEP_k-way is similar to those of the GCEP_2-way.

C. Complexity of GCEP

The complexity of moving a cell and updating the affected cell gains is $O(P)$ like F&M method [1]. Let L be the maximum number of cells moved during a pass of optimization, then one pass takes $O(L * P)$. One may fix the number of optimization passes by properly choosing BAL(i)'s and the *reduction_ratio*. Hence, the total complexity of the GCEP algorithm can be bounded by $O(L * P)$. The experimental results show that the CPU time increases almost linearly as the number of cells increases.

V. PARTITIONING RESULTS

The HGCEP algorithm is implemented in C and run on SUN4/40 under the UNIX operating system. We describe the effectiveness of clustering in Section V.A. In Section V.B, we compare the results of HGCEP with those of other typical approaches using MCNC benchmark examples. In Section V.C, we discuss the sensitivity of HGCEP to parameters and cost functions.

A. Effectiveness of Clustering

The hierarchical approach based on clustering in HGCEP improves both the performance and the efficiency of partitioning. As can be seen in Table V, one run of HGCEP can generate significantly better results than the best results of 20 runs of GCEP. In CPU time, one run of HGCEP takes less than four times as long as one run of GCEP.

B. Benchmark Results

Table VI shows the sizes of nine benchmark examples from MCNC. Among the examples, PrimGA1 (PrimGA2) has the same netlist as for PrimSC1 (PrimSC2). However, their cell sizes are different. Since partitioning is performed under size-constraints, the partitioned result for PrimGA1 (PrimGA2) may be different from that for PrimSC1 (PrimSC2). For all the examples, the size constraint on the subsets, unless otherwise stated, was set to:

$$\text{average\_subset\_size} * 0.8 \leq \text{subset\_size} \leq \text{average\_subset\_size} * 1.2$$

Table VII shows the two-way partitioning results. In the table, the simulated annealing (SA), the Fiduccia–Mattheyses (F&M), the primal-dual (PD), and the HGCEP algorithms are evaluated using the real circuit examples. The results of SA, F&M, and PD are from [14]. (They were obtained from SUN SPARC station1.) Note that the CPU time listed in the table is only the time for one run of each algorithm. Hence 500 runs of F&M would takes 500 times the CPU time given in the table.

The simulated annealing is implemented with a cooling ratio $r = 0.95$. At each temperature, random moves are generated L times. In [14], $L = 16*$(the number of cells) is used. The cost is defined by the number of crossing nets + size penalty, where size penalty = $\frac{0.05}{avg\_size} * \sum_i f_i$ and

$$f_i = \begin{cases} (|V_i| - Min)^2 & \text{if} |V_i| < Min. \\ & (Min = 0.4 * (\textit{total sizes of cells})) \\ (|V_i| - Max)^2 & \text{if} |V_i| > Max. \\ & (Max = 0.6 * (\textit{total sizes of cells})) \\ 0 \text{ otherwise.} \end{cases}$$

The authors also implemented the simulated annealing algorithm and obtained similar results. We have not used sophisticated annealing schedule such as [10] which may improve the efficiency of the simulated annealing algorithm. When the simulated annealing algorithm can not find a near optimum solution within a "reasonable" amount of CPU time (say 10^4 s/run), one may run a faster version of it several times and then

TABLE VII
PARTITIONING RESULTS

	SA (10 runs)			F&M (500 runs)			PD(1run)		HGCEP(1run)	
Example	Best Mincut	Avg. Mincut	sec/run	Best Mincut	Avg. Mincut	sec/run	Mincut	sec	Mincut	sec
Test02	90	109.0	4081	107	126.7	24.4	92	217	83	408
Test03	59	86.4	1284	81	145.8	20.6	58	530	58	286
Test04	53	69.5	2784	44	46.0	10.4	43	431	44	324
Test05	53	107.8	3176	42	44.9	20.1	42	1563	47	904
Test06	74	85.5	4710	74	180.7	55.3	62	1216	47	347
PrimGA1	36	38.9	2424	37	86.5	17.2	39	207	38	42
PrimGA2	131	168.5	1903	146	397.0	193.4	123	1612	119	594
PrimSC1	41	63.8	4490	39	88.6	17.7	39	195	38	41
PrimSC2	128	156.4	4920	157	402.3	191.3	120	2512	119	588
Total (%)	665 (112%)	885.3 (149%)		727 (123%)	1518.5 (256%)		618 (104%)		593 (100%)	

TABLE VIII
4-WAY PARTITIONING RESULTS

	Quadrisection (5 runs)			HGCEP(1 run)	
Examples	Best	Average	sec/run	Cost	sec
Test2	241	258.4	81	161	393
Test3	179	236.4	78	132	297
Test4	181	208.0	69	137	309
Test5	332	358.2	203	201	870
Test6	209	232.2	89	182	356
PrimGA1	138	149.2	12	96	45
PrimGA2	385	506.0	185	296	590
PrimSC1	109	133.2	10	89	45
Primsc2	431	492.2	182	286	582
Total (%)	2205 (140%)	2573.8 (163%)		1580 (100%)	

choose the best result. The F&M algorithm is implemented according to [1].

The hierarchical GCEP produced 4% better results on the average when compared with those of the primal-dual method and 18% better results than the best results of 500 runs of the F&M method. It outperforms the F&M method by more than 60% in the number of nets cut on the average.

As can be seen from the tables, HGCEP is efficient enough to handle large examples. In practice, the CPU time increases almost linearly as the number of cells increases.

Table VIII shows four-way partitioning results for the Quadrisection [4] and HGCEP. The CPU time in Table VIII is also the time for one run of each algorithm. HGCEP outperformed the quadrisection by 39% on the average at the expense of three to five times longer CPU time.

C. Tuning the Parameters

All the results of HGCEP in Table VII and VIII are obtained using the same set of parameters. The two major parameters of our HGCEP algorithm are α (used during clustering) and *reduction_ratio* (used to enforce size-constraints). The sensitivity of the HGCEP due to α is shown in Table I and $\alpha = 0.5/200$ seems to be a reasonable choice. The sensitivity due to *reduction_ratio* is shown in Table III and it is easy to choose *reduction_ratio* = 0.9 from the table.

Of course, one may obtain better results by tuning the parameters for each circuit or for a small set of circuits. For several examples, HGCEP could reduce the number of nets cut by 2 or 3 when different values and *reduction_ratio* are used for each circuit. However, optimized values for one circuit may make other results worse. Since we assume one run of HGCEP, we used the same set of parameters for all the circuits.

We also examined different cost functions such as the one suggested in [5]. However, contrary to [5], the results get worse. The new cost in [5] is an estimation of the number of net crossings under certain assumptions on the layout style. Note that the number of net crossings in a real layout can be much larger than the number of nets cut. We believe that HGCEP works better with the exact cost function since it is effective in minimizing the given cost function.

VI. CONCLUSIONS

A new hypergraph-partitioning algorithm has been developed. The GCEP algorithm starts with a relaxed size constraints on subsets and then gradually tightens the size constraints so that the final solution satisfies the required constraints. To find a good initial solution, closely connected cells are merged into clusters and then the clusters are partitioned several times using different sets of parameters by the GCEP algorithm. This hierarchical approach improves the partitioning results and allows efficient evaluation of various initial partitions. Tuning the parameters for HGCEP is relatively easy. Experimental results show that the new partitioning algorithm is effective and efficient.

REFERENCES

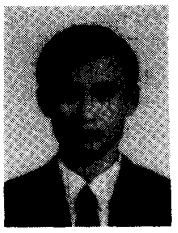
[1] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. 19th Design Automation Conference*, 1982, pp. 175–181.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman, 1979.

[3] B. M. Kerninghan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 297–307, Feb. 1970.
[4] P. R. Suaris and G. Kedem, "Quadrisection: A new approach to standard cell layout," in *Proc. Int. Conf. Computer-Aided Design*, 1987, pp. 474–477.
[5] C. Sechen and D. Chen, "An improved objective function for mincut circuit partitioning," in *Proc. Int. Conf. Computer-Aided Design*, 1988, pp. 502–505.
[6] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438–446, May 1984.
[7] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, Jan. 1989.
[8] Y. Saab and V. Rao, "An evolution-based approach to partitioning ASIC systems," in *Proc. 26th Design Automation Conf.*, 1989, pp. 767–770.
[9] Y. G. Saab and V. B. Rao, "Stochastic evolution: A fast effective heuristic for some generic layout problems," in *Proc. 27th Design Automation Conf.*, 1990, pp. 26–31.
[10] J. Lam and J. Delosme, "Performance of a new annealing schedule," in *Proc. 25th Design Automation Conf.*, 1988, pp. 306–311.
[11] Y. C. Wei and C. K. Cheng, "Towards efficient hierarchical designs by ratio cut partitioning," in *Proc. Int. Conf. Computer-Aided Design*, 1989, pp. 298–301.
[12] C. W. Yeh and C. K. Cheng, "A general purpose multiple way partitioning algorithm," in *Proc. 28th Design Automation Conf.*, 1991, pp. 421–426.
[13] L. Hagen and A. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1074–1085, Sept. 1992.
[14] C. W. Yeh, C. K. Cheng, and T. Y. Lin, "An experimental evaluation of partitioning algorithms," in *Proc. Int. ASIC Conf.*, 1991, p. 14–1.
[15] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, May 13, pp. 671–680, 1983.
[16] S. Mayrhofer and U. Lauther, "Congestion-driven placement using a new multi-partitioning heuristic," in *Proc. Int. Conf. on Computer-Aided Design*, 1990, pp. 332–335.

Hyunchul Shin (S'79–M'80–S'85–M-87) received the B.S. degree in electronics engineering from Seoul National University, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology in 1978 and 1980, respectively, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1987.

From 1980 to 1983, he was with the Department of Electronics Engineering, Kum-Oh Institute of Technology, Korea. In 1983, he received a Fulbright Scholarship. From 1987 to 1989, he was a Member of the Technical Staff at AT&T Bell Laboratories, Murray Hill, NJ. Since August 1989, he has been with the Department of Electronics Engineering, Han-Yang University, Korea. His research interests include design and synthesis of integrated circuits and systems.



Chunghee Kim received the B.S. degree and the M.S. degree in electronics engineering from Han-Yang University, in 1991 and 1993, respectively. He is working toward the Ph.D. degree in electronics engineering at Han-Yang University.

His research interests include automatic layout and synthesis of VLSI's.