LETTER

A Hierarchical Circuit Clustering Algorithm with Stable Performance

Seung-June KYOUNG[†], **Kwang-Su SEONG**[††], **In-Cheol PARK**[†], *and* **Chong-Min KYUNG**[†*], ***Nonmembers***

SUMMARY Clustering is almost essential in improving the performance of iterative partitioning algorithms. In this paper, we present a clustering algorithm based on the following observation: if a group of cells is assigned to the same partition in numerous local optimum solutions, it is desirable to merge the group into a cluster. The proposed algorithm finds such a group of cells from randomly generated local optimum solutions and merges it into a cluster. We implemented a multilevel bipartitioning algorithm (MBP) based on the proposed clustering algorithm. For MCNC benchmark netlists, MBP improves the total average cut size by 9% and the total best cut size by 3–4%, compared with the previous state-of-the-art partitioners.

key words: *VLSI, CAD, partitioning, clustering*

1. Introduction

In netlist partitioning which is widely used for such applications as circuit placement, floor-planning and rapid prototyping, the given netlist is partitioned into k sub-netlists such that they have minimum connection among them and satisfy the given size constraint.

When k is 2, the partitioning is called bipartitioning. Generally, the multi-way partitioning is achieved by recursively applying the bipartitioning until each sub-netlist meets the desired size constraint. Thus, the bipartitioning plays an important role in the netlist partitioning. Since finding the optimal solution for the netlist partitioning problem is NP-complete [12], many heuristic algorithms have been proposed. Kernighan and Lin proposed a good heuristic algorithm (KL) for graph bisection in 1970. The KL algorithm iteratively swaps a pair of cells to reduce cut size and has time complexity of $O(n^2 \log n)$ [3]. Based on the KL algorithm, Fiduccia and Mattheyses proposed a linear time heuristic algorithm (FM) which, instead of swapping a pair of cells, moves a cell at a time into the other partition. With ever-growing complexity of the problem, the FM algorithm is more attractive due to its linear time complexity. However, the solution obtained with the FM algorithm tends to be trapped in a local minimum because of the nature of the improvement technique, and is heavily affected by initial solutions. In addition, as the problem size increases, solution space to be explored increases exponentially while search space increases linearly. Therefore, the chance to find a good local minimum significantly decreases. To overcome this weak point, many clustering algorithms were proposed [7]–[10][14].

The netlist of a VLSI circuit can be divided into a number of tightly coupled sub-netlists. Based on this fact, clustering algorithms try to merge such a sub-netlist into a cell (cluster) and construct the contracted netlist. In the previous literatures, it is known that iterative partitioning algorithms produce nearly optimal solutions for small-sized problems. In addition, Goldberg and Burstein [13] show that FM-style algorithms give good results when average cell degree is greater than 5, where the cell degree means the number of nets connected to the cell. Therefore, for large-sized problems, the performance of a FM-style partitioning algorithm can be improved by using clustering methods, since clustering helps reduce the solution space and increases the cell degree.

Previous clustering algorithms can be classified into two categories according to the method used to find clusters: local clustering approach and global clustering approach. Local approach mainly considers the local connectivity of a netlist to form clusters. Bui [14] applied a maximal random matching algorithm to build clusters. Cong and Smith [8] formed clusters by collapsing cliques. And Shin and Kim [7] selected a pair of cells (or clusters) which maximizes the *closeness* function and merged them.

On the other hand, global approach tries to use the global information of a netlist. For example, to capture the global information, Garbers [10] constructed clusters by finding and merging two cells which are connected by more than k edge-disjoint paths whose length is less than l. This approach is promising in finding the natural clusters but has the time complexity of $O(d^{2l-1}n)$, where d is the maximum cell degree. Hagen and Kahng [9] proposed a global clustering algorithm which uses cycles in random walk to construct clusters. And many algorithms based on the eigenvectors of a graph were also proposed [1][2][11].

Although the local approaches have low time complexity, resulting clusters tend to be *unnatural* because

Manuscript received April 27, 1998.

[†]The authors are with Department of Electrical Engineering, Korea Advanced Institute of Science and Technology 373-1 Kusong-dong, Yusong-gu, Taejon 305–701 Korea.

[††]The author is with Electrical Engineering Department, YeungNam University 214-1 Daedong, Kyongsan, Kyongbuk 712-749 Korea.

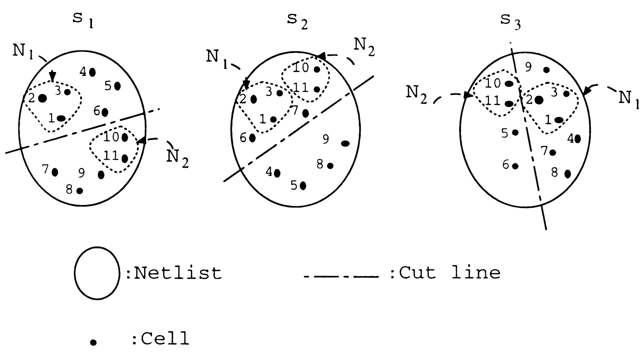

Fig. 1 3 local optimum solutions.

they are generated by using only the local connectivity information, thus limiting the performance. On the other hand, the global approaches are more able to find natural clusters. However, as the problem size increases, the time complexity of global approaches makes it difficult to be used for large-sized problems.

In this paper, we present a clustering algorithm which utilizes the property of local optimum solution space. The proposed clustering algorithm is grounded on the following observation: if a group of cells is assigned to the same partition in numerous local optimum solutions, it is desirable to merge the group into a cluster. Starting with K local optimum solutions selected randomly from the local optimum solution space, the proposed clustering algorithm finds cells that are always assigned to the same partition throughout the K local optimum solutions and merges them into a cluster.

For the case of bipartitioning, let us assume that 3 local optimum solutions (s_1, s_2 and s_3) are randomly selected as shown in Fig. 1. There are two cell groups, N_1 and N_2. Since every cell in N_1 is always assigned to the same partition, the proposed clustering algorithm merges every cell in N_1 into a cluster. This is also true for cells in N_2.

To evaluate the performance of the proposed clustering method, we implemented a multilevel bipartitioning algorithm (MBP) and tested it for MCNC benchmark netlists. Compared with the state-of-the-art partitioners, MBP yields 9% improvement in the total average cut size and 3–4% improvement in the total best cut size for the MCNC benchmark netlists.

The rest of this paper is organized as follows. In Sect. 2, we describe the proposed clustering algorithm in detail. Section 3 analyzes the time complexity of the proposed clustering algorithm, and in Sect. 4, we show the experimental results for MCNC benchmark netlists.

2. The Proposed Algorithm

2.1 Motivations

In the netlist partitioning problem, the size of the local optimum solution space is generally very small compared to that of the random solution space which consists of all solutions satisfying the given size constraint. In addition, there exists a group of cells which move together in a number of local optimum solutions. Since local optimum solutions are resulted from the process of reducing cut size, such cell group tends to be composed of tightly connected cells. This means that for reducing cut size, the cells of such a group should be assigned to the same partition fairly independent of initial solutions. Therefore, the cell group thus obtained, contains some global connectivity information of a netlist and merging it into a clusters is considered to be effective in reducing the solution space. We will now describe how to find the cell groups having the property mentioned above.

A netlist can be modeled by a hypergraph $H = (V, E)$ where V is the set of cells and E is the set of nets. Assuming bipartitioning, let $\mathcal{S}(H)$ be the set of all local optimum solutions for a given size constraints, and N_i be a subset of cells in V. And $\mathcal{S}^K(H)$ denotes a set of K solutions randomly sampled from $\mathcal{S}(H)$. Then, if we randomly sample a local optimum solution from $\mathcal{S}(H)$, the probability that every cell $\in N_i$ is assigned to the same partition becomes

$$p(N_i) = \frac{|\mathcal{S}_{N_i}(H)|}{|\mathcal{S}(H)|} \quad (0 \le p(N_i) \le 1)$$

where $\mathcal{S}_{N_i}(H)$ is the set of all local optimum solutions in which every cell $\in N_i$ is assigned to the same partition.

From the $\mathcal{S}^K(H)$, we make a set of cell groups, $\mathcal{N}^K(H) = \{N_1, N_2, .., N_m\}$, which satisfy the following constraints,

1 Every cell in each N_i is assigned to the same partition in every solution $\in \mathcal{S}^K(H)$
2 $N_i \cap N_j = \emptyset,\ for\ all\ i, j\ (i \ne j)$
3 $|N_i| \ge 2,\ for\ 1 \le i \le m$

Constraint 1 is self-explanatory, and constraint 2 means that no cell can be assigned to more than one N_i. Constraint 3 is added, since one cell is always assigned to the same partition. In Fig. 1 where the 3 local optimum solutions are shown, $\mathcal{N}^K(H)$ becomes $\{\{1, 2, 3\}, \{10, 11\}\}$. Then, we calculate the average of $p(N_i)$ for all $N_i \in \mathcal{N}^K(H)$, denoted $\overline{p(\mathcal{N}^K(H))}$, as follows,

$$\overline{p(\mathcal{N}^K(H))} = \frac{1}{m}\sum_{i=1}^{m} p(N_i)$$

$\overline{p(\mathcal{N}^K(H))}$ implies that if it is close to 1, every cell in each $N_i \in \mathcal{N}^K(H)$ has a tendency to be assigned to the same partition in most local optimum solutions. Therefore, it is desirable to find $\mathcal{N}^K(H)$ whose $\overline{p(\mathcal{N}^K(H))}$ is close to 1 and merge the cells in each $N_i \in \mathcal{N}^K(H)$ into a cluster.

The heart of the proposed clustering algorithm is to find $\mathcal{N}^K(H)$ whose $\overline{p(\mathcal{N}^K(H))}$ is close to 1 using local optimum solutions. There are two aspects to be considered: the first is how to generate local optimum solutions and the second is how to determine K, meaning that how many local optimum solutions should be generated to find $\mathcal{N}^K(H)$ whose $\overline{p(\mathcal{N}^K(H))}$ is close to 1.

Although any iterative bipartitioning algorithm can be used for the local optimum solution generation, the previous bipartitioners tend to generate solutions that are similar to one another. This property is not useful for us as we want to randomly sample local optimum solutions. Generally a greedy iterative bipartitioner is much more dependent on initial solutions than those which have hill-climbing capability. This property of the greedy iterative bipartitioner is useful for our purpose because starting from random initial solutions, we can generate local optimum solutions that are far different from each other. Thus, we used the greedy iterative bipartitioner for generating local optimum solutions.

To determine K, we need to analyze the characteristics of a netlist. However, as the analysis is quite complicated to be applied for real circuits, we chose K based on the following observation. Given K local optimum solutions $(\mathcal{S}^K(H))$, we determine $\mathcal{N}^K(H)$. Although, we can not compute the exact value of $p(N_i)$, we can estimate $\overline{p(\mathcal{N}^K(H))}$ by randomly generating many local optimum solutions and then counting the solutions in which every cell in $N_i \in \mathcal{N}^K(H)$ is assigned to the same partition, i.e.,

$$\begin{aligned}\overline{\hat{p}(\mathcal{N}^K(H))} &= \frac{1}{m}\left(\frac{|\hat{\mathcal{S}}_{N_1}(H)|+..+|\hat{\mathcal{S}}_{N_m}(H)|}{|\hat{\mathcal{S}}(H)|}\right)\\ &\approx \overline{p(\mathcal{N}^K(H))}\ \ (when\ \ |\hat{\mathcal{S}}(H)| \rightarrow \infty)\end{aligned}$$

where $\hat{\mathcal{S}}(H)$ means the set of local optimum solutions actually generated as a substitute for $\mathcal{S}(H)$, and $\hat{\mathcal{S}}_{N_i}(H)$ $(\subseteq \hat{\mathcal{S}}(H))$ is the set of local optimum solutions where every cells in N_i is assigned to the same partition. For example, Fig. 2 shows $\overline{\hat{p}(\mathcal{N}^K(H))}$ according to different K values for a benchmark named "industry2P". The horizontal axis in logarithmic scale represents the number of local optimum solutions generated, and the vertical axis denotes $\overline{\hat{p}(\mathcal{N}^K(H))}$. It is observed in Fig. 2 that for a sufficient number of local optimum solutions generated, $\overline{\hat{p}(\mathcal{N}^K(H))}$ converges to a certain value, and that when K is larger than 20, the resulting value of $\overline{\hat{p}(\mathcal{N}^K(H))}$ is close to 1.

2.2 Algorithm description

In this section, we describe the proposed clustering algorithm as shown in Fig. 3. We assume that K is given here. The $Greedy\_Partition(H, K, SC)$ procedure of step 1 generates K local optimum solutions, $\mathcal{S}^K(H)$,

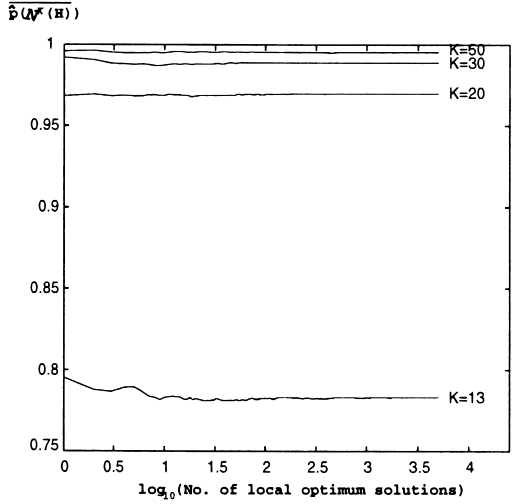

Fig. 2 $\overline{\hat{p}(\mathcal{N}^K(H))}$ for different K values on "industry2P" benchmark.

```
Procedure $Clustering(H, K)$
Input: $H = (V, E)$ : Input netlist.
        $K \equiv$ The number of local optimum solutions.
Constant: $NCELLS \equiv$ Number of cells in $H$.
Variable: $j \equiv$ Local optimum solution index.
          $t \equiv$ Cell index.
          $l \equiv$ Cluster index.
          $\mathcal{S}^K(H) \equiv \{s_1, .., s_K\}$
                    $\equiv$ Local optimum solution set.
          $cell_t[1, 2..K] \equiv$ Block assignment array.
          $BT \equiv$ Binary tree.
Output: $\mathcal{C} = \{C_0, \ldots, C_{l-1}\}$ = clusters of $H$.
1  $\mathcal{S}^K(H) = Greedy\_Partition(H, K, SC)$;
2  for($j = 1; j \leq K; j = j + 1$){
       for($t = 1; t \leq NCELLS; t = t + 1$)
           $cell_t[j]$ = block index of $cell_t$ in $s_j$;
   }
   $BT = \emptyset$;
3  for($t = 1; t \leq NCELLS; t = t + 1$){
       Traverse BT using $cell_t[1..K]$;
       if(the path of $cell_t[1..K]$ does not exist)
           Expand BT to contain this path;
       Append $cell_t$ to the cell list of the leaf node;
   }
   $l = 0$;
4  foreach(all leaf nodes of $BT$){
       $C_l$ = cells in this leaf node;
       l=l+1;
   }
5  return $\mathcal{C} = \{C_0, C_1, \ldots, C_{l-1}\}$;
```

Fig. 3 The proposed clustering algorithm.

where H is the input netlist and SC is the size constraint. The $Greedy\_Partition(H, K, SC)$ procedure is a greedy bipartitioner whose algorithm is similar to the FM algorithm except that the hill-climbing capability is dropped out.

Step 2 makes a block assignment list for each cell of H. The block assignment list of a cell is the index list of partition to which the cell is assigned for all solutions in $\mathcal{S}^K(H)$. Step 3 constructs a binary tree, BT, which

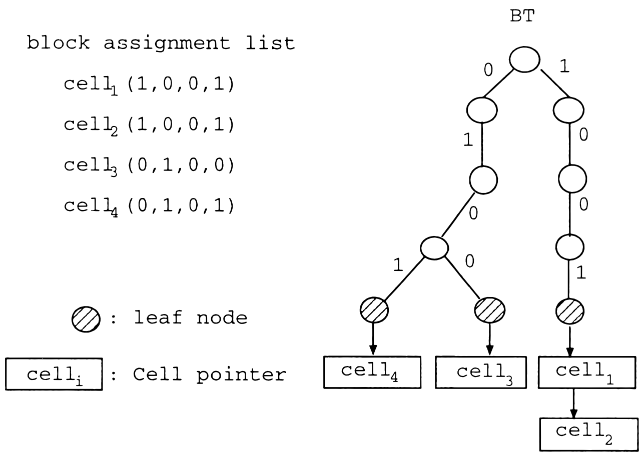

Fig. 4 Binary tree construction.

```
Procedure MBP
Input: $H_0 = (V_0, E_0) \equiv$ Input netlist.
Variable: $L \equiv$ Number of level.
          $\mathcal{C}_i \equiv$ Intermediate clustering.
          $P_i \equiv$ Intermediate bipartitioning.
Constant: $T \equiv$ Minimum cell number.
          $ITER \equiv$ Iteration number.
Output: $P_0 = \{X_0, Y_0\} \equiv$ Final bipartitioning.

    $i = 0$, $|V_{-1}| = \infty$;
1   While $((|V_i| \geq T)$ and $(|V_i| < |V_{i-1}|))$ {
        $\mathcal{C}_i = Clustering(H_i, K)$;
        $H_{i+1}(V_{i+1}, E_{i+1}) = Make\_Netlist(H_i, \mathcal{C}_i)$;
        $i = i + 1$;
    }
    $L = i$;
2   $P_L = FM(H_L, ITER, NULL)$;
    for($i = L - 1; i \geq 0; i = i - 1$){
        $P_i = Mapping(\mathcal{C}_i, P_{i+1})$
        $P_i = FM(H_i, 1, P_i)$
    }
3   return $P_0$
```

Fig. 5 Multilevel bipartitioning algorithm(MBP).

contains all paths of block assignment lists. Cells with the same block assignment list are linked to the cell list of the corresponding leaf node. An example is shown in Fig. 4 which shows the resulting BT and cell lists of leaf nodes for the given block assignment lists. In step 4, we merge all cells linked to a leaf node into a cluster (C_l). Thus, the cluster formed in step 4 consists of cells that are always assigned to the same partition in every solution $\in \mathcal{S}^K(H)$.

The proposed multilevel bipartitioner (MBP) is described in Fig. 5. Step 1 forms the coarsening phase. We hierarchically simplify the given netlist until the number of cells in H_i becomes less than the given cluster number (T) or H_i is not simplified. $Clustering(H_i, K)$ is the proposed clustering procedure, and $Make\_Netlist(H_i, \mathcal{C}_i)$ is the procedure which constructs the simplified netlist H_{i+1} from H_i and $\mathcal{C}_i$ (clustering results of H_i). Step 2 is the uncoarsening phase. We first construct a bipartitioning result for H_L (the most simplified netlist). The $FM(H_i, ITER, INIT)$ procedure is the standard FM bipartitioner, where $ITER$ means the number of iterations and $INIT$ is an initial solution. P_L is the best solution among the results of $ITER$ runs of FM. $Mapping(\mathcal{C}_i, P_{i+1})$ procedure takes a bipartitioning result P_{i+1} of H_{i+1} and a clustering result $\mathcal{C}_i$ of H_i, and then constructs the initial solution for H_i to further improve P_i. This process is repeated until the bipartitioning result of the original netlist is obtained.

3. Time Complexity

In this section, we analyze the time complexity of the proposed clustering algorithm. The time complexity of step 1-2 in Fig. 3, is $O(Kn)$, where K is the number of local optimum solutions to be generated and n is the number of cells in the netlist. The *traverse* and *expand* operation in step 3, can be completed by searching a cell's block assignment list. Therefore, if we account for all cells, the time complexity of step 3 is $O(Kn)$. As K is constant, the proposed clustering algorithm has linear time complexity.

4. Results

Figures 6 and 7 show $\overline{\hat{p}(\mathcal{N}^K(H))}$ for "p2" and "golem3" benchmarks, respectively. In Figs. 6 and 7, the horizontal axis in logarithmic scale represents the number of local optimum solutions generated and the vertical axis denotes $\overline{\hat{p}(\mathcal{N}^K(H))}$. Characteristics of benchmarks are shown in Table 1. As shown in the figures, $\overline{\hat{p}(\mathcal{N}^K(H))}$ increases sharply when K is less than 20, and the value of $\overline{\hat{p}(\mathcal{N}^K(H))}$ becomes close to 1 when K is greater than 20. Therefore, we can say that every cell in each $N_i \in \mathcal{N}^K(H)$, that is found from more than 20 random local optimum solutions, moves together in almost all local optimum solutions. Thus, we can regard $\mathcal{N}^K(H)$ determined from more than 20 local optimum solutions randomly generated as sufficiently good for our purpose.

Figure 8 shows how the average cut size obtained from MBP varies according to different values of K for five benchmark circuits. The benchmarks cover huge circuits as well as small circuits. Results are obtained by bipartitioning each benchmark circuit, where the cut size means the number of crossing nets between two partitions. We performed this experiment with $T = 400$ and $ITER = 50$. The horizontal axis represents K and the vertical axis denotes the ratio of the cut size, i.e., (average cut size from 10 runs)/(average cut size from 10 runs for $K = 2$). As we can expect, the quality of clusters is too low for small K values to yield good partitioning results. As shown in Fig. 8, MBP produces good results for whole test circuits when K is greater than 20.

From the above experimental results, we know that

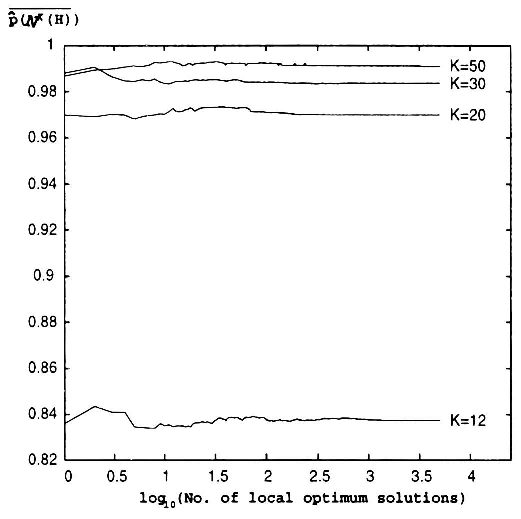

Fig. 6 $\overline{\hat{p}(\mathcal{N}^K(H))}$ for different K values on "p2" benchmark.

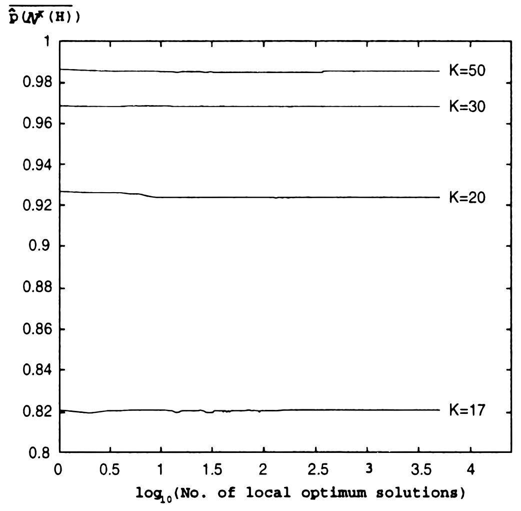

Fig. 7 $\overline{\hat{p}(\mathcal{N}^K(H))}$ for different K values on "golem3" benchmark.

as $\overline{\hat{p}(\mathcal{N}^K(H))}$ becomes close to 1, MBP tends to produce good results. These experimental results coincide with our observation that: if a cell group is assigned to the same partition in numerous of local optimum solutions, it is desirable to merge the cell group into a cluster.

The proposed MBP algorithm was implemented in C language and run on a SPARC20 machine. We ran MBP with $K = 24$, $T = 400$ and $ITER = 50$. Table 2 shows bipartitioning results for MCNC benchmark netlists with the 45-55% size balance constraint. We assume that the size of each cell is uniform. In Table 2, the first column shows the name of each benchmark netlist. The column labeled "cut size" represents the number of crossing nets between two partitions.

ML_c [4] and hMeTiS [6] are the state-of-the-art multilevel partitioners whose results are quoted from [4] and [6]. The column labeled MBP(10) reports the best and average cut size obtained from 10 runs of our algorithm, and hMeTiS(20) reports the best cut size from 20 runs of hMeTiS. The column labeled FM(100) shows the best and average cut size of FM from 100 runs, and the column labeled ML_c shows the best cut size of ML_c from 10 runs and average cut size from 100 runs. For the best cut size, MBP is 40% better than FM, 3% better than ML_c and 4% better than hMeTiS. In addition, MBP is 61% better than FM and 9% better than ML_c in terms of the total average cut size. The total average cut size of MBP is only 3% worse than the total best cut size. This implies that the proposed clustering method is quite effective for capturing the global connectivity information of a netlist.

Table 1 Charateristics of benchmark circuits.

bench	♯cells	♯nets	♯pins
balu	801	735	2697
bm1	882	903	2910
p1	833	902	2908
t4	1515	1658	5975
t3	1607	1618	5807
t2	1663	1720	6134
t6	1752	1541	6638
struct	1952	1920	5471
t5	2595	2750	10076
19ks	2844	3282	10547
p2	3014	3029	11219
s9234	5866	5844	14065
biomed	6514	5742	21040
s13207	8772	8651	20606
s15850	10470	10383	24712
ind2	12637	13419	48404
ind3	15406	21923	65791
s35932	18148	17828	48145
s38584	20995	20717	55203
avq_small	21918	22124	76231
s38417	23849	23843	57613
avq_large	25178	25384	82751
golem3	103048	144949	338419

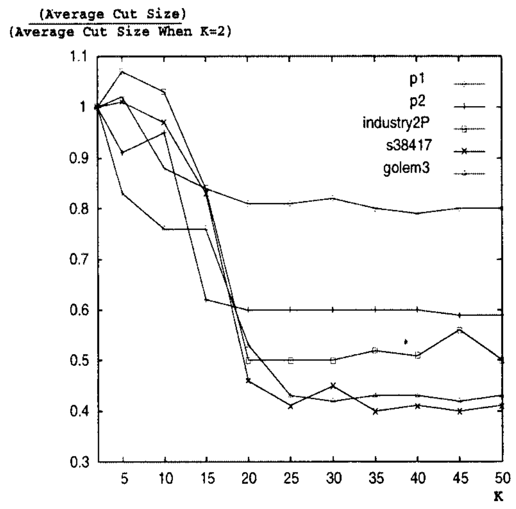

Fig. 8 Average cut size trend for different K values.

Table 2 Bipartitioning results for MCNC benchmark netlists. Each block satisfies 45-55% size constraint. NA means 'Not Available from ref [4]'.

bench	cut size							standard deviation(σ)	
	FM(100)		hMeTiS(20)	ML_c		MBP(10)		FM	MBP
	Min	Avg	Min	Min	Avg	Min	Avg		
balu	27	36	27	27	*NA*	27	27	9.63	0.00
bm1	48	74	51	51	*NA*	47	48	16.44	2.76
p1	48	74	50	52	*NA*	47	49	9.51	2.51
t4	87	135	51	49	*NA*	53	54	29.93	0.63
t3	59	105	58	58	*NA*	58	59	30.06	1.12
t2	111	167	88	92	*NA*	86	88	20.04	1.54
t6	63	88	60	60	*NA*	60	64	7.11	2.59
struct	42	55	33	33	*NA*	33	33	5.12	0.64
t5	106	175	71	72	*NA*	72	72	31.50	1.27
19ks	123	172	106	108	*NA*	104	108	17.94	2.46
p2	141	267	145	145	*NA*	142	146	27.39	4.78
s9234	49	87	40	41	45	40	41	28.03	1.60
biomed	83	127	83	84	91	83	83	54.07	0.49
s13207	80	123	55	55	71	53	57	19.70	4.61
s15850	120	176	42	56	56	42	47	30.82	7.34
ind2	305	631	167	174	196	162	172	70.14	10.38
ind3	248	514	254	243	276	241	272	177.34	30.15
s35932	52	181	42	42	45	42	45	49.56	3.16
s38584	54	225	47	48	52	47	47	110.53	0.80
avq_small	342	601	130	134	159	128	140	121.47	5.97
s38417	182	366	51	50	72	49	52	77.99	5.50
avq_large	320	731	127	131	163	127	139	149.05	5.92
golem3	2468	3269	1445	1374	1462	1338	1349	222.79	9.43
total	5158	8379	3223	3179		3081	3192		
sub-total					2688		2444		
%imprv over	40.2%	61.9%	4.4%	3.0%	9.0%	-	-		

The last two columns of Table 2 show the standard deviation in 10 runs for FM and MBP. As known in the previous literatures, FM yields unstable results, while MBP produces very stable results. The average standard deviation of MBP for the 23 benchmarks is only 4.59 nets. This implies that we can obtain fairly good bipartitioning results from a single run of MBP.

Table 3 shows the CPU time of the FM algorithm, ML_c and MBP. The CPU time in seconds is the total required time for 1 run of each algorithm on a SPARC20 machine. Since the CPU time presented in the literatures is not directly comparable to ours because of the difference of underlying computer systems, we implemented the FM algorithm and ML_c as specified in [5] and [4], respectively, to obtain relative CPU time. In the experiment of ML_c, we used the same parameter specified in [4]. The most time-consuming procedure of MBP is the *Greedy_Partition*() which computes K local optimum solutions. In order to shorten the required CPU time of MBP, it is desirable to parallelize the procedure. The column labeled "1 run" of MBP reports the CPU time in the case that the procedure *Greedy_Partition*() is not parallelized. And the column labeled "1 run (6M)" shows the CPU time of MBP when the procedure *Greedy_Partition*() is parallelized for 6 machines. For the former case, MBP is about 5 times slower than ML_c, while for the latter case, MBP is about 1.5 times slower than ML_c.

Table 3 The total required time of each algorithm for 1 run.

bench	CPU time (seconds)			
	FM	ML_c	MBP	
	1 run	1 run	1 run	1 run(6M)
balu	0.36	2.97	6.49	6.04
bm1	0.35	3.28	6.70	6.14
p1	0.51	2.99	7.96	7.11
t4	0.96	5.59	11.49	9.04
t3	0.59	5.80	14.03	9.84
t2	0.72	6.02	15.57	10.48
t6	0.72	6.53	12.41	9.68
struct	0.63	5.73	10.79	8.98
t5	1.66	9.59	19.17	13.09
19ks	2.25	10.25	25.99	16.93
p2	1.40	11.62	27.13	19.00
s9234	4.71	14.11	34.23	19.75
biomed	5.32	23.64	49.44	26.48
s13207	6.82	21.15	53.40	26.96
s15850	8.06	26.32	71.75	32.97
ind2	10.30	56.32	163.83	68.39
ind3	17.82	91.07	262.70	104.84
s35932	18.02	51.95	174.22	74.58
s38584	21.73	67.85	203.51	81.69
avq_small	31.92	110.42	355.28	134.11
s38417	23.05	64.69	198.80	73.65
avq_large	33.40	122.72	385.97	154.16
golem3	154.78	468.63	2985.52	914.34
total	346.07	1189.27	5096.46	1828.33

5. Conclusions

In this paper, we proposed an efficient clustering algorithm which utilizes the property of the local optimum space. We experimentally showed that about 20 local optimum solutions are enough for capturing the property of the local optimum solution space. The multilevel bipartitioner (MBP) based on the proposed clustering algorithm, is able to produce very stable and satisfying results. Besides the obtained partitioning results are better than those of the previous algorithms, the total average cut size of MBP is only about 3% worse than the total best cut size. This means that for the netlist partitioning problem, using the property of the local optimum solution space is a profitable method.

References

[1] C.J. Alpert and A.B. Kahng, "Fast spectral methods for ratio cut partitioning and clustering," Proc. IEEE Int. Conf. on Computer-Aided Design, 1991.

[2] C.J. Alpert and A.B. Kahng, "Simple eigenvector-based circuit clustering can be effective," Proc. IEEE Int. Symp. on Circuits and Systems, pp.IV/683–686, Atlanta, May 1996.

[3] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell Syst. Tech.J, vol.47, pp.291–307, Feb. 1970.

[4] J.H. Huang, C.J. Alpert, and A.B. Kahng, "Multilevel circuit partitioning," ACM/IEEE Design Automation Conf., pp.530–533, 1997.

[5] C.M. Fiduccia, et al., "A linear time heuristic for improving network partitions," ACM/IEEE Design Automation Conf., pp.175–181, 1982.

[6] V. Kumar, G. Karypis, R. Aggarwal, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," ACM/IEEE Design Automation Conf., pp.526–529, 1997.

[7] H. Shin and C. Kim, "A simple yet effective technique for partitioning," IEEE Trans. VLSI Systems, vol.1, no.3, Sept. 1993.

[8] J. Cong and M. Smith, "A parallel bottom-up clustering with applications to circuit partitioning in VLSI design". ACM/IEEE Design Automation Conf., pp.755–760, 1993.

[9] L. Hagen J. Cong and A.B. Kahng, "Random walks for circuit clustering," IEEE Intl. ASIC Conf., pp.14.2.1–14.2.4, 1991.

[10] H.J. Promel, J. Garbers, and A. Steger, "Finding clusters in VLSI circuits". In IEEE Int. Conf. Computer-Aided Design, pp.520–523, 1990.

[11] L. Hagen and A.B. Hahng, "New spectral methods for ratio cut partitioning and clustering". vol.11, no.9 Sept. 1996.

[12] M. Garey and D.S. Johnson, "Computers and intractability: A guide to the theory of np-completeness," Freeman, 1979.

[13] M.K. Goldberg and M. Burstein. "Heuristic improvement technique for bisection of VLSI networks," Proc IEEE Intl. Conf. Computer Design, pp.122–125, 1983.

[14] C. Jones T. Bui, C. Heigham, and T. Leighton, "Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms". ACM/IEEE Design Automation Conf., pp.775–778, 1989.