Journal of Korea Multimedia Society Vol. 7, No. 8, August 2004(pp. 1183-1194)

A Linear-Time Heuristic Algorithm for *k*-Way Network Partitioning

Tae-Young Choe[†]

ABSTRACT

Network partitioning problem is to partition a network into multiple blocks such that the size of cutset is minimized while keeping the block sizes balanced. Among these, iterative algorithms are regarded as simple and efficient which are based on cell move of Fiduccia and Mattheyses algorithm, Sanchis algorithm, or Kernighan and Lin algorithm. All these algorithms stipulate balanced block size as a constraint that should be satisfied, which makes a cell movement be inefficient. Park and Park introduced a balancing coefficient R by which the block size balance is considered as a part of partitioning cost, not as a constraint. However, Park and Park's algorithm has a square time complexity with respect to the number of cells. In this paper, we proposed *Bucket* algorithm that has a linear time complexity with respect to the number of cells, while taking advantage of the balancing coefficient. Reducing time complexity is made possible by a simple observation that balancing cost does not vary so much when a cell moves. Bucket data structure is used to maintain partitioning cost efficiently. Experimental results for MCNC test sets show that cutset size of proposed algorithm is 63.33%~92.38% of that of Sanchis algorithm while our algorithm satisfies predefined balancing constraints and acceptable execution time.

선형의 시간 복잡도를 가지는 휴리스틱 *k*-방향 네트워크 분할 알고리즘

최 태 영[†]

요 약

네트워크 분할 문제는 주어진 네트워크를 여러 개의 블록으로 분할하되 절단집합의 크기를 최소화하는 동시에 블록들의 크기는 균일하도록 하는 문제이다. 많은 네트워크 분할 알고리즘들 중에서 반복 알고리즘 부류는 간단하면서도 효과적이라는 것이 알려져 있다. 대표적인 반복 알고리즘은 Fiduccia와 Mattheyses가 제안한 셀 단위의 이동을 하는 알고리즘, 이의 개량형인 Sanchis가 제안한 *k*-방향 분할 알고리즘이 있다. 이들 알고리즘은 '블록들의 균일한 크기'를 반드시 만족되어야 하는 조건으로 고정하고 있으며, 이 조건은 비효율적인 셀 이동을 유발하는 원인이 된다. Park과 Park은 블록들의 균일한 크기를 제한 조건으로 두는 대신 '균형 비용'으로 정하고, 균형 비용의 크기를 조절하는 균형 계수 R을 제안했다. 하지만 그 알고리즘은 셀 개수의 제곱에 해당하는 상당히 높은 시간복잡도를 가진다는 문제가 있다. 본 논문에서는 셀 개수의 선형에 해당하는 시간복잡도를 가지면서도 균형 비용의 장점을 이용하는 *bucket* 알고리즘을 제안한다. MCNC 테스트 셋을 통한 실험은 Sanchis가 제안한 알고리즘에 비해서 제안된 알고리즘이 만들어낸 분할에서의 절단 집합의 크기가 63.33%에서 92.38%로 줄어들었으며, 균형 조건을 명시하지 않았음에도 불구하고 결과 분할들은 균형 조건을 만족함을 보여준다.

Key words: Network Partition(네트워크 분할), Heuristic Algorithm(휴리스틱 알고리즘), Parallel System (병렬 시스템), Linear Time Complexity(선형 시간 복잡도)

※ 교신저자(Corresponding Author) : 최태영, 주소 : 경북 구미시 신평동 188(730-701), 전화 : 054)467-4389, FAX : 054)467-4473, E-mail : choety@kumoh.ac.kr

접수일 : 2003년 12월 15일, 완료일 : 2004년 2월 4일

[†] 금오공과대학교 컴퓨터공학부

※ This paper was supported by Research Fund, Kumoh National Institute of Technology.

1. Introduction

Network partitioning is a fundamental problem in the field of design automation. The goal of network partitioning is to divide the cells of a network into several subsets subject to size (or balance) constraints while minimizing the interconnections among those subsets. Since the network partitioning problem is known to be NP-complete, many heuristic algorithms have been proposed[1-5].

Kernighan and Lin proposed K-L algorithm that starts with a random 2-way partition and it tries to reduce cutset size by making small local changes such as successively exchange a pair of cells between two blocks[1]. Fiduccia and Mattheyses proposed 2-way partitioning algorithm called F-M algorithm. F-M algorithm moves cell one by one [6]. *Sanchis* algorithm is proposed by Sanchis[2] and it extends F-M algorithm to allow k-way partition. These algorithms have two drawbacks: First, result partitions are largely influenced by initial partitions. Second, they set allowable unbalance size $r = S^{max}$ to allow all cells can move, where S^{max} is the largest cell size. As the result, block size difference becomes too large to be used in real circuit design. If they restrict $r < S^{max}$ moderate size, some cells cannot move and result partitions with larger cutset are generated[7].

Simulated annealing is probabilistic method to find an optimal solution. Given a randomly generated partition, it transforms the partition either to a new partition of lower cost or to another new partition of higher cost with low probability. Though this method generates comparatively good result, it requires much execution time.

Spectral method is mainly applied to graph partitioning problem[3,8-10]. Spectral method partitions a given graph using eigenvectors of Laplacian matrix, which represents topology of the graph. The result partition by Spectral method can be further improved by iterative methods. It has an advantage that global information of a graph is used. A disadvantage is that a given network must be modified to a graph with some costs in terms of computation time and accuracy of results. Moreover, if the network includes many nodes of high degree, the generated graph will be very dense, which requires large memory and long computation time for calculating eigenvectors.

Balancing factor is a method to overcome the problems of F-M family algorithms. Partitioning algorithms that consider the balancing factor regard an unbalanced state as cost rather than as unpardonable condition. Ratio-cut algorithm[4] defines ratio-cut as the partitioning cost obtained by multiplying two blocks size and dividing it by cutset size. The more balanced two blocks are or the smaller cutset is, the lower ratio-cut is. The goal of ratio-cut algorithm is to get smallest ratio-cut partition.

Park and Park's algorithm[11] (P-P algorithm) is a generalization of ratio-cut algorithm. It can control the relative importance between cutset cost and balancing cost by adding balancing coefficient R to cost function. However, its time complexity is higher than F-M algorithm because whenever a cell is moved, balancing cost of all free cells must be recalculated.

In this paper, we propose an improved algorithm (Bucket algorithm) which works better than F-M family algorithms in terms of cutset size and balancing cost while maintaining reasonable computation time. Main idea of this paper is to use the balancing coefficient for control of balancing and to use bucket structures for execution time reduction. The remainder of the paper is organized as follows; In Chapter 2, we introduce network partitioning problem. In Chapter 3, we present an algorithm and its data structure. In Chapter 4, the experimental results are shown, and lastly we discuss the conclusion and further research in Chapter 5.

2. Definitions

In order to describe the proposed algorithm, it is required to introduce some definitions and their explanations. A network is a hyper-graph $H = (C, N)$, where C is a set of cells and N is a set of nets as shown in Fig. 1. A cell c is an object having its size. A net n is a set of cells which are connected by the net. A net has its weight. Without loss of generality, we assume the weight of every net to be 1. A pin is a connection point between a cell and a net.

Notation 1 For a given network $H = (C, N)$,

1. $|C|$: the number of cells.
2. $|N|$: the number of nets.
3. $S(c)$: the size of a cell c.
4. $n_c = \{n \mid c \in n\}$: the set of nets incident on a cell c.
5. $|n_c|$: the number of nets incident on a cell c.
6. $c_n = \{c \mid c \in n\}$: the set of cells on a net n.
7. $|c_n|$: the number of cells on a net n.
8. $|n_c|^{\max} = \max_{c \in C}|n_c| = p$.
9. $|c_n|^{\max} = \max_{n \in N}|c_n|$.
10. $S^{\max} = \max_{c \in C} S(c)$.
11. $W = \sum_{c \in C} S(C)$.

For a given network H, we define the size of

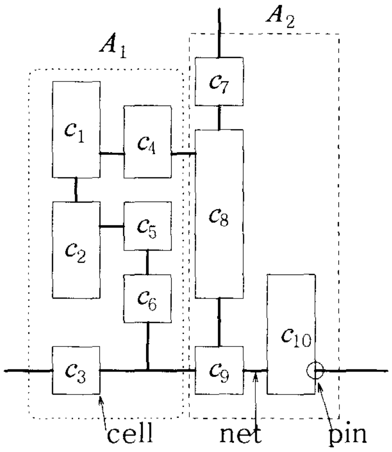

Fig. 1. An example of a 2-way network partition.

the network m as the total number of pins in the network. Thus,

$$m = \sum_{c \in C}|n_c| = \sum_{n \in N}|c_n|.$$

A k-way partition of a network is described by the k blocks $(A_1, A_2, \cdots, A_k)$ where $A_i \cap A_j = \emptyset$ for $i \neq j$ and $\bigcup_i A_i = C$. A_i is called i-th block of the partition. Given a k-way partition, a net is called *uncut* if all cells in the net reside in one block and *cut* otherwise. The *cutset* is a set of cut nets and its size (i.e., the number of cuts in the cutset) is denoted by $\aleph$ and defined as follows:

$$\aleph = |\{n | A_i \cap c_n \neq \emptyset \text{ and } A_j \cap c_n \neq \emptyset \text{ for } i < j\}|. \quad (1)$$

Fig. 1 shows an example of a 2-way partition of a circuit: the partition $P = (A_1, A_2)$ where $A_1 = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ and $A_2 = \{c_7, c_8, c_9, c_{10}\}$. Note that the cutset includes two nets: (c_4, c_8) and (c_6, c_3, c_9), i.e., $\aleph = 2$.

The k-way network partitioning problem is to find a k-way partition $(k \geq 2)$ such that the cutset size is minimized and each block is constrained to have a certain size. For a given network H, assume a k-way partition P. The partitioning cost of P is defined to be a function of two components: a cutset cost and a balancing cost. The cutset cost represents the cutset size $\aleph$ and the balancing cost represents how much P is balanced. If we denote the partitioning cost, cutset cost, and balancing cost by $C_{\mathrm{PAR}}(P)$, $C_{\mathrm{CUT}}(P)$, and $C_{\mathrm{BAL}}(P)$, respectively, then $C_{\mathrm{PAR}}(P) = f(C_{\mathrm{CUT}}(P), C_{\mathrm{BAL}}(P))$. In the form of formal notations, the partitioning problem is to find partition $P_{\min}$ such that

$$C_{PAR}(P_{\min}) = \min_{P \in \wp} C_{PAR}(P),$$

where $\wp$ is the set of all possible k-way partitions of a network.

There are many ways to evaluate how much a partition is balanced[4,8,11]. A partition P is said to be perfectly-balanced if the total sum of the size differences of all disjoint pairs of blocks is minimum. That is, the partition P is perfectly-balanced if

$$\sum_{A_i, A_j \in P, i<j} |S(A_i) - S(A_j)| = \min_{P \in \mathscr{P}} \Big(\sum_{A'_i, A'_j \in P', i<j} |S(A'_i) - S(A'_j)| \Big).$$

3. Proposed Algorithm

For a given network H, assume a k-way partition $P = (A_1, A_2, \cdots, A_k)$. The partitioning cost $C_{PAR}(P)$ of P is defined as [11]:

$$C_{PAR}(P) = C_{CUT}(P) + R \cdot C_{BAL}(P), \qquad (2)$$

where cutset cost $C_{CUT}(P)$ is defined as $\aleph$, and balancing cost $C_{BAL}(P)$ is defined as

$$C_{BAL}(P) = \sum_{1 \le i<j \le k} |S(A_i) - S(A_j)|. \qquad (3)$$

Note that $C_{BAL}(P)$ in Equation (3) is different from that in [11]. The gain of cell c in block A_i to block A_j, denoted as $\gamma_{A_j}(c)$, represents the amount of partitioning cost to be reduced after c is moved from A_i to A_j. Let $P' = (A_1, A_2, \cdots, A'_i, \cdots, A'_j, \cdots, A_k)$ be the partition after c is moved from A_i to A_j. Then,

$$\gamma_{A_j}(c) = C_{PAR}(P) - C_{PAR}(P'). \qquad (4)$$

The gain $\gamma_{A_j}(c)$ consists of two components: cutset gain $\chi_{A_j}(c)$ and balancing gain $\pi_{A_j}(c)$. These components are defined as follows:

$$\chi_{A_j}(c) = C_{CUT}(P) - C_{CUT}(P'). \qquad (5)$$

$$\pi_{A_j}(c) = C_{BAL}(P) - C_{BAL}(P'). \qquad (6)$$

By substituting equation (2) to equation (4), and substituting equation (5) and (6), we have

$$\gamma_{A_j}(c) = \chi_{A_j}(c) + R \cdot \pi_{A_j}(c). \qquad (7)$$

Cutset gain $\chi_{A_j}(c)$ is obtained by setting gain level to 1 in[2]. The maximum cutset gain $\chi^{\max}$ is $|n_c|^{\max}$ or p.

Balancing gain $\pi_{A_j}(c)$ is obtained by

$$\begin{aligned}\pi_{A_j}(c) = & \sum_{l \ne i,j} (|S(A_l) - S(A_i)| - |S(A_l) - S(A_i) + S(c)|) \\ & + \sum_{l \ne i,j} (|S(A_l) - S(A_j)| - |S(A_l) - S(A_j) - S(c)|) \\ & + |S(A_i) - S(A_j)| - |S(A_i) - S(A_j) - 2S(c)|).\end{aligned} \qquad (8)$$

Maximum balancing gain is $S^{\max}(2k-2)$ in case that a cell with size $S^{\max}$ is in A_i, target block is A_j, and $A_j + S^{\max} \le A_l \le A_i - S^{\max}$, for all blocks A_l ($l \ne i,j$). Thus range of partitioning gain covers from $-p - R \cdot S^{\max}(2k-2)$ to $p + R \cdot S^{\max}(2k-2)$. Since the cutset cost is the most explicit quality of result partition, the size of bucket is set as $2 \cdot \lceil p + R \cdot S^{\max} \cdot (2k-2) \rceil + 1$.

Fig. 2 shows the main step of proposed network partitioning algorithm. Procedure *DO_PASS* iteratively improves a given partition. *DO_PASS*

```
Procedure PARTITION_NETWORK(H, k, R)
begin
// input: a network $H = (C, N)$, the number of blocks $b$,
// and balancing coefficient $R$
// output: $k$-way partition $P = (A_1, A_2, \cdots, A_k)$
read network description;
construct an initial partition $P$;
repeat
  DO_PASS(P);
until there is no more improvement in cutset size;
end
                              (a)

Procedure DO_PASS(P)
begin
1 Free all cells;
2 Calculate initial cutset gain of all cells;
3 Calculate initial balancing gain of all cells;
4 Calculate initial partitioning gain of all cells;
5 while there exist free cells do
6   Select a free cell $c$ which has maximum partitioning gain.
7   Move and lock the cell $c$;
8   Adjust cutset gain of adjacet free cells to $c$;
9   Recalculate balancing gain of all free cells;
10  Recalculate partitioning gain of all free cells;
  endwhile
11 return the best partition $P'$ among produced partitions;
end.
                              (b)
```

Fig. 2. Proposed network partitioning algorithm.

first frees all cells and calculates gain of each cells. Then it repeatedly finds free cell with max gain, moves it, locks it, and recalculates gain of all the remaining free cells. If cell c moves from block A_i to block A_j, cutset gain of free cells which are neighbour of c is changed, and balancing gains of every free cells are changed.

In procedure *DO_PASS*, time complexity of line 1, 2, and 4 is $O(|C|)$, $O(km)$, and $O(k|C|)$, respectively. Since the time complexity of equation (8) is $O(k)$, time complexity of line 3 is $O(k^2|C|)$. *While* statement in line 5 runs $|C|$ times because all cells are moved. Time complexity of statements in line 6 and 7 is $O(|C|)$, since max gain cell is already selected in line 4 and 10. Time complexity of line 8 is $O(mk)$, which is same as Sanchis algorithm. Statement in line 9 uses equation (8) for each free cell and change of a block size influence to balancing gains of all free cells. Thus it requires $O(k^2|C|^2)$ step in procedure *DO_PASS*. The number of steps in line 10 is $O(k|C|^2)$. Thus time complexity of procedure *DO_PASS* is $O(k^2|C|^2)$. Calculation of balancing gain is dominant part of proposed algorithm, and it makes proposed algorithm have higher time complexity than F-M or Sanchis algorithms. Proposed algorithm need not data structure like bucket since all cells are scanned each time.

4. Reduction of balancing gain calculation time

The main time complexity of our algorithm comes from the fact that balancing gain is recalculated after each cell is moved. We expect that there is little performance degradation even if a partitioning algorithm computes balancing gain approximately considering two reasons: First, if we use small balancing coefficient R like 0.1, balancing gain is less important than cutset gain, and small change of balancing gain can be negligible. Second, difference of balancing gain between cells in a block is little modified. In 2-way partition $P=(A_i, A_j)$, when a cell c in block A_i moved to A_j, balancing gain of other cell d in block A_i changes to

$$\pi_{A_i}(d) = |S(A_i) - S(A_j)| - |S(A_i) - S(A_j) - 2S(c)|.$$

Using this equation, relation between balancing gain and cell size is drawn in Fig. 3, where $S(A_i) = S(A_j) + 2S$ for some size $S > 0$. Consider

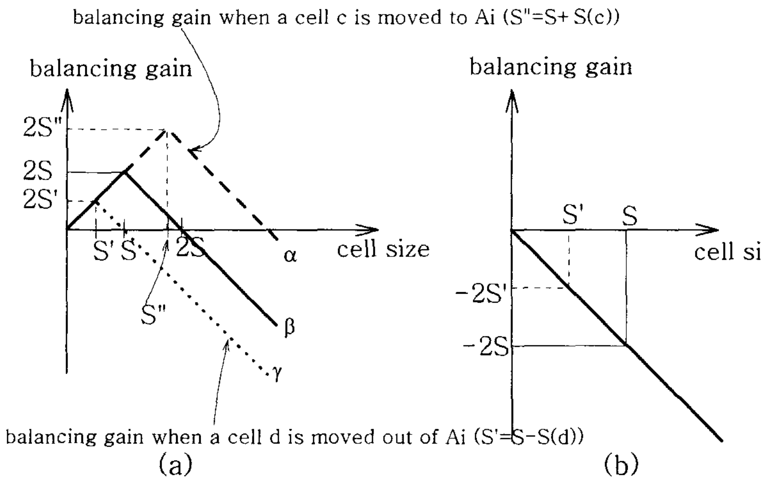

Fig. 3. Relation between cell size and balancing gain. In 2-way partition $P = (A_i, A_j)$, when $S(A_i) = S(A_j) + 2S$, (a) balancing gain π_{A_i} for each cell in block $S(A_i)$, (b) balancing gain π_{A_i} for each cell in block $S(A_j)$.

the cells in block A_i. If a cell c with size $S(c)$ in block A_j moves to A_i, then $S(A_i) = S(A_j) + 2S''$, and balancing gains of cells change to upper dashed line. That is, balancing gains of cells with larger size than S increase. If cell d moves from block A_i to A_j, then $S(A_i) = S(A_j) + 2S'$, and balancing gains of cells change to lower dotted line. That is, balancing gains of cells with larger size than S decrease. In case of cells in block A_j, balancing gains of cells does not changed whether a cell move to A_j or A_i. From this fact, we can see that relative changes of balancing gain between cells are small when small cell moves.

Fig. 4 shows *DO_PASS′* procedure which reduces balancing gain calculation time. Parameter T_{ref} is the number of balancing gain calculation in *DO_PASS′*. S_{ref} is value which divides sum of cell size W by T_{ref}. If the sum of a moved cell size S_{acc} exceeds S_{ref}, Bucket algorithm calculates balancing gain of all free cells. Consider a set of cell movements from a source block to a target block. All the cell movements in the set has the same balancing gain changes after a cell movement. That tells that scanning all free cells is redundant when a cell with the max gain is selected. *DO_PASS′* selects one among representative cells that are cells with the maximum partitioning gain in a source block and target block pairs. Partitioning gains of the representative cells not of all free cells are calculated after a cell is moved. Thus gain of $k(k-1)$ cells are computed after a cell movement. Procedure *select_max_gain* in Fig. 4 (b) calculates exact gains of representative cells and selects a cell with the largest gain among the representative cells. However, it is need to adjust gain of all cells periodically in order to prevent distortion of balancing gain. After S_{ref} amount of cells are moved, gains of all free cells are re-computed.

In case of 2-way partition, time complexity of line 9 in *DO_PASS′* shown in Fig. 4 (a) is $O(m)$

```
Procedure DO_PASS′
begin
1 $S_{ref} \leftarrow W/T_{ref}$;
2 $S_{acc} \leftarrow 0$;
3 Calculate initial cutset gain of all cells;
4 Calculate initial balancing gain of all cells;
5 Calculate initial partitioning gain of all cells;
6 while there are free cells do
7    $c \leftarrow$ Select_max_gain_cell();
8    Move and lock $c$;
9    Recalculate cutset gain of free cells which adjacent to $c$;
10   $S_{acc} \leftarrow S_{acc} + S(C)$;
11   if $S_{acc} > S_{ref}$ then
12      Recalculate the balancing gains of all free cells;
13      Recalculate the partitioning gains of all free cells;
14      $S_{acc} \leftarrow 0$;
     endif
  endwhile
15 return the best partition among produced $|C|$;
end.
```

(a)

```
Procedure select_max_gain
begin
max_gain ← min_value;
for each source block $A_f$ do
  for each target block $A_t$ do
    Cell $c_i \leftarrow$ max_gain cell in bucket $[A_f, A_t]$;
    Calculate partitioning gain of $c_i$;
    if partitioning gain of $c_i$ > max_gain then
      max_cell ← $c_i$;
      source_block ← $A_f$;
      target_block ← $A_t$;
    endif
  endfor
endfor
return max_cell; source_block, target_block;
end.
```

(b)

Fig. 4. (a) *DO_PASS′* of improved algorithm (b) *select_max_gain* procedure used by *DO_PASS′*.

during a *DO_PASS′*. Time complexity of line 12 and 13 in *DO_PASS′* is $O(T_{ref}|C|)$. Line 7, 8, and 9 in *DO_PASS′* has constant time complexity. Line 3, 4, and 5 has $O(|C|)$ time complexity. Thus time complexity of *DO_PASS′* is $O(|C|T_{ref})$ in 2-way partition.

In case of k-way partition, since max gain cell is selected among $k(k-1)$ buckets, *Select_max_* *gain_cell()* has $O(k^2|C|)$ time complexity. After a cell is moved, cutset gains of adjacent cells for each block are calculated. Thus time complexity of line 9 in *DO_PASS'* is $O(km)$. Line 12 and line 13 takes $O(kT_{ref}|C|)$ time. Thus time complexity of *DO_PASS'* is $O(k^2|C|+km+(k-1)T_{ref}|C|)$ in k-way partition.

Choe proposed balancing coefficient $R^* > p/(2 \cdot S)$ which guarantees perfectly- balanced partition in same cell size S [7]. $R' = p/(2*|S_{avg}-\sigma|)$ is proposed by Choe, when cell sizes are different, the average is S_{avg}, and standard deviation is σ. Since the balancing coefficient influences the range of gains and the size of bucket structure is determined by the range of gains, large balancing coefficient R causes large bucket size and increases time complexity. Though the balancing coefficient R' is not so large, we propose lower R'' to get more small cutset. Sparse networks have more freedom than dense network, and partitioning algorithms hardly get minimum cutset in sparse networks. Thus we consider density of network as parameter of balancing coefficient. We define density d' of network as

$$d' = \frac{m}{|C|^2}.$$

In graph, d' is equal to $\frac{n}{2|C|^2}$. We propose R'' as

$$R'' = d' \frac{|n_d|^{\max}}{(S_{ave}-\sigma)^2}.$$

5. Experimental Results

In order to evaluate the performance of our proposed algorithm, we used circuits obtained from Microelectronic Center of North Carolina (MCNC) benchmarks. Table 1 shows the properties of each circuit. Sanchis algorithm is considered for comparison. These are programmed in C, and are executed in Sparc 20. Each algorithms are executed 50 times.

Parameter T_{ref} highly influences the execution time of Bucket algorithm. Various values of T_{ref} are tested in order to yield suitable partition quality and computation time. Fortunately, Fig. 5 shows that cutset size and balance of resulting partitions are not largely influenced by T_{ref}. Circuits PrimSC1, PrimGA2, Test05 in Table 1 are tested in the case of 2-way and 12-way partitions and balancing coefficient R' and R''. In Fig. 5, cutset size decreases as T_{ref} increase only in the case of 2-way partition of PrimGA2 and Test05 with R'. Moreover, balancing cost is almost independent of T_{ref}. The reason that cutset size and balance are little influenced by T_{ref} is that partitioning gain is calculated prior to selecting max gain cell. We set T_{ref} to 10 for remaining experiments. Though Sanchis algorithm can move all cells by setting allowable unbalance size τ as $S^{\max}$ in 2-way partition, τ can make size difference between blocks too large in real circuit. So we restricted the block size to 2:3, that is, proportion of the smallest block and the largest block is 2:3. Thus

Table 1. circuits used in the experiment

circuits	No. cell $\|C\|$	No. net $\|N\|$	No. pin p	std.dev of cell size	max degree of cell	max size of cell
PrimSC1	833	1266	3303	1.55	9	5.63
PrimGA2	3014	3817	12014	1.83	9	10.36
Test04	1515	2189	7324	16.13	103	596.70
Test05	2595	3488	12150	19.95	100	970.80
Test06	1752	2048	7696	2.75	8	19.17
Test08	3804	4266	14119	3.55	13	35.80

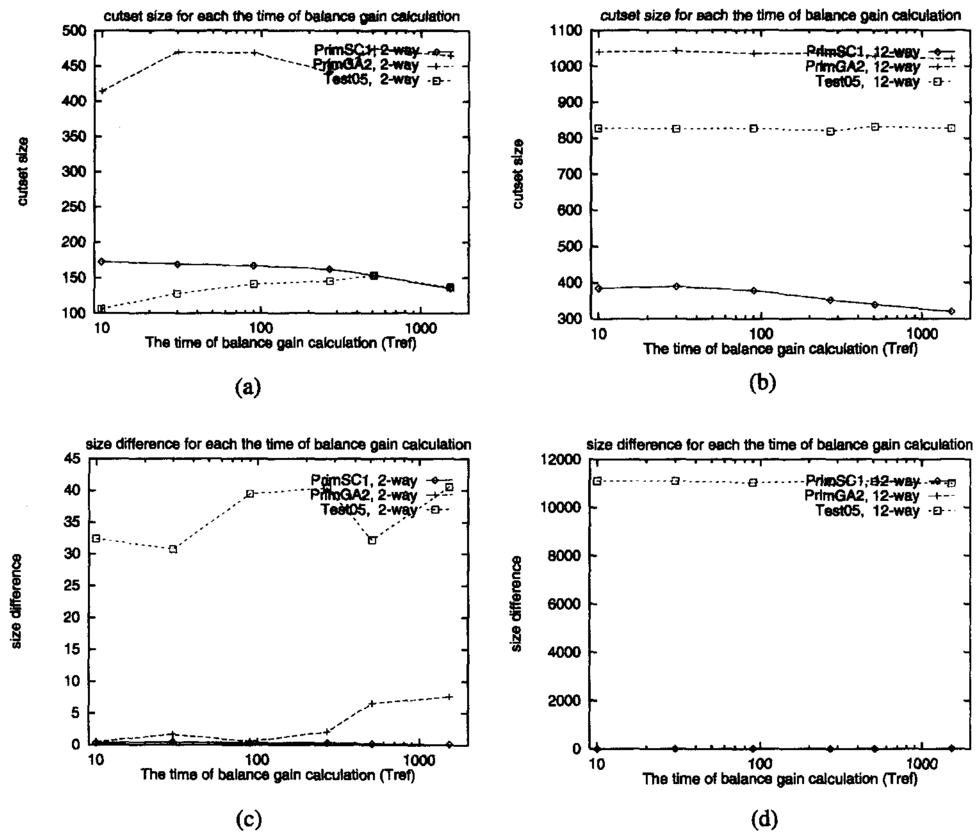

Fig. 5. Cutset for each balancing gain calculation time T_{ref} in case of Bucket algorithm: (a) cutset size for 2-way partition (b) cutset size for 12-way partition (c) size difference for 2-way partition (d) size difference for 12-way partition.

some cells are not considered as moving candidate during *DO_PASS* because of size constraint, even if they are free. *Gain level* in Sanchis algorithm is restricted to 1.

Fig. 6, 7, 8, 9, 10, and 11 show each minimum cutset size and sum of block size difference when

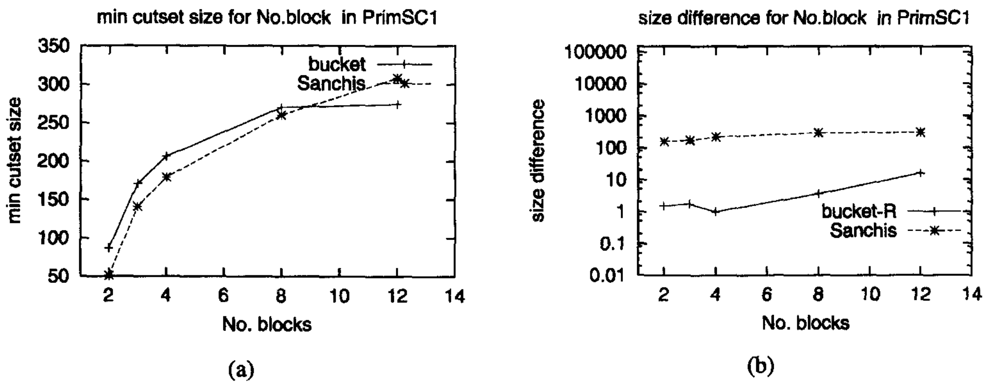

Fig. 6. Performance comparison between Sanchis algorithm, and Bucket algorithm in PrimSC1 (a)cutset size, (b)sum of block size difference.

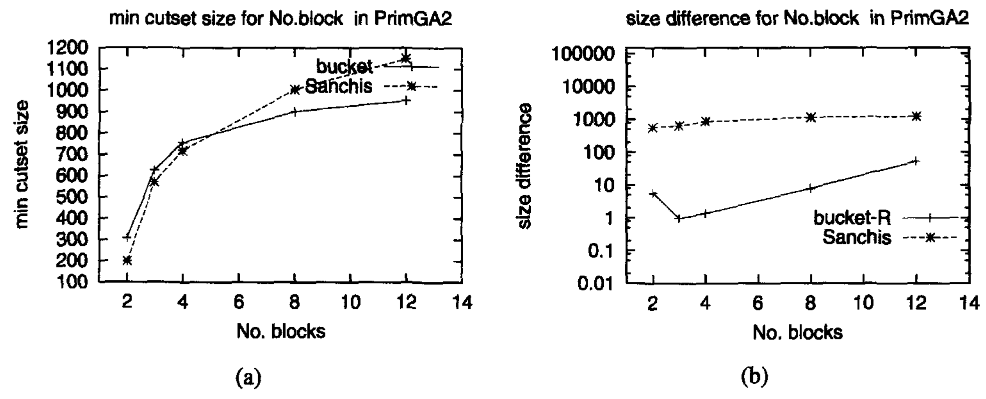

Fig. 7. Performance comparison between Sanchis algorithm, and Bucket algorithm in PrimGA2 (a) cutset size, (b) sum of block size difference.

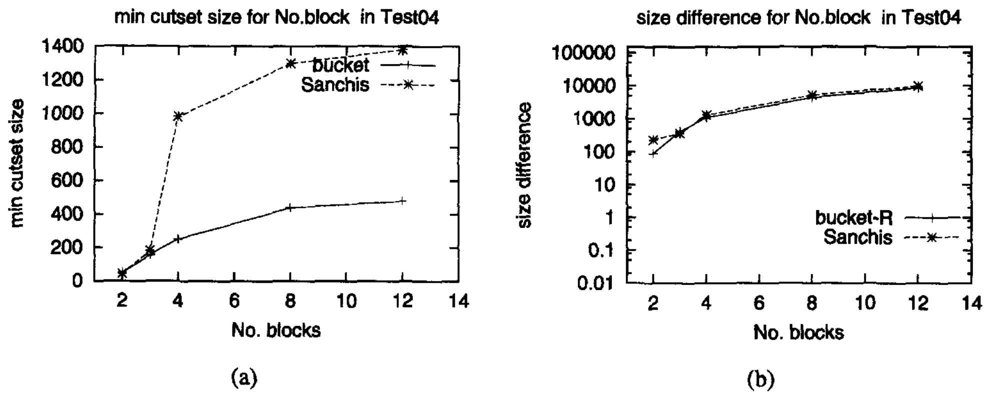

Fig. 8. Performance comparison between Sanchis algorithm, and Bucket algorithm in Test04 (a) cutset size (b) sum of block size difference.

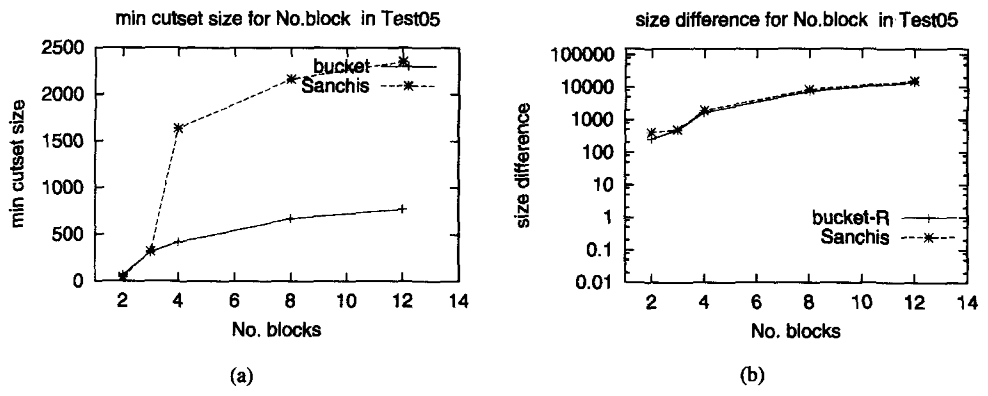

Fig. 9. Performance comparison between Sanchis' algorithm, and Bucket algorithm in Test05 (a) cutset size, (b) sum of block size difference.

the number of block changes. In 2-way partitioning, Sanchis algorithm has the smallest cutset, while size difference between blocks is the largest. In 12-way partitioning, Bucket algorithm generates the smallest cutset partition. In experiment, since deviation of circuit size is low while max

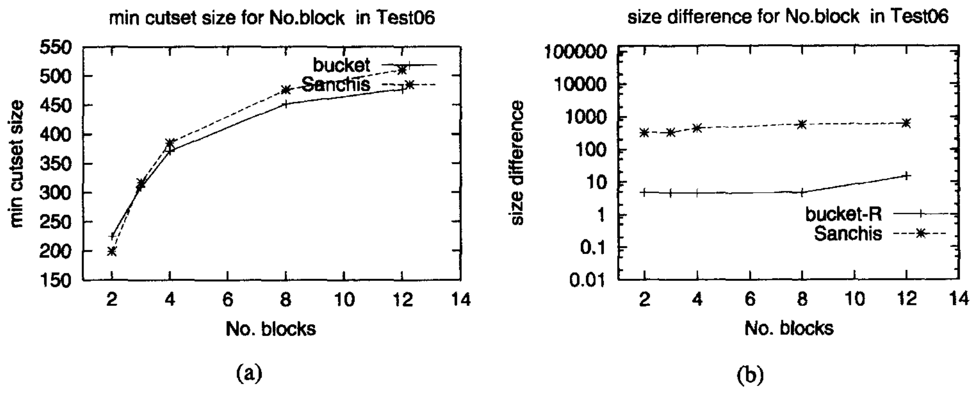

Fig. 10. Performance comparison between Sanchis algorithm, and Bucket algorithm in Test06 (a) cutset size, (b) sum of block size difference.

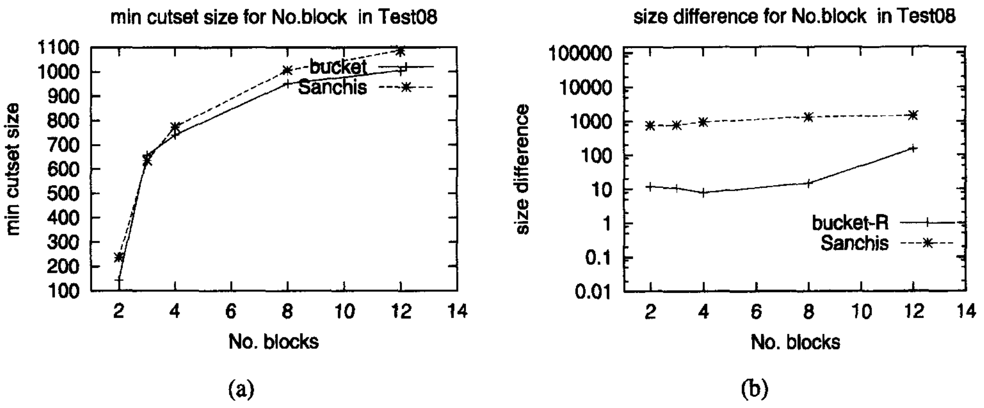

Fig. 11. Performance comparison between Sanchis algorithm, and Bucket algorithm in Test08 (a) cutset size, (b) sum of block size difference.

degree of cell are large, R' is larger than 1.

As we can see from Table 1, balancing constraint cannot be satisfied when there is a cell with size of 1/3, 1/4, 1/8 of circuit size. In Test04, and Test05, as the number of block k increases, size difference between blocks increases sharply.

Sanchis algorithm increases cutset size as well as size difference in case of Test04, and Test05. Since it gives higher priority to balance constraint, it cannot decrease cutset size when balance constraint is not satisfied. On the other hand, Bucket algorithm makes smaller cutset. It seems a property of algorithm like bucket which consider only cost function in partitioning.

Table 2 shows total execution times of two partitioning algorithm along the number of blocks. Our Bucket algorithm consumes more execution time than Sanchis algorithm does, because balancing gain computation is included. The execution time of Bucket algorithm increases more quickly

Table 2. Total execution times (second) of two partitioning algorithms: Ratio is the percentage of execution time of Bucket algorithm over that of Sanchis algorithm.

Algorithm	2-way	3-way	4-way	8-way	12-way
Sanchis	10.23	13.4	18.94	55.58	100.68
Bucket	13.3	27.8	42.9	142.3	341.7
ratio	130.01%	207.46%	226.50%	256.03%	339.39%

than that of Sanchis algorithm. Since each cell has more gains and chances in partitions with more blocks, the number of *DO_PASS* calls increases as the number of the block increases.

6. Conclusion

In this paper, we proposed a *linear-time* heuristic algorithm to solve network partitioning problem. Proposed Bucket algorithm generates perfectly balanced partition which cannot be guaranteed by the F-M family algorithms. Average cutset size of proposed algorithm is 63.33~92.38% of Sanchis algorithm.

In proposed algorithm, there are some parameters which user must determine:

- balancing coefficient R: For large R, more balanced and larger cutset sized partition is generated. For small R, less balanced and smaller cutset sized partition is generated.
- balancing gain calculation frequency: As more frequently balanced gain is calculated, more precisely max gain cell is selected.

We propose a balancing coefficient R based on statistical properties of given network. The balancing gain calculation frequency T_{ref} is set as 10 according to experiments.

We will examine the detail properties of those parameters and determine the relations with result partition as future works.

References

[1] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, pp. 291-307, February 1970.

[2] L. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Computers*, vol. 38, January 1989.

[3] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM J. Matrix Analysis and Applications*, vol. 11, pp. 430-452, July 1990.

[4] Y.-C. Wei and C.-K. Cheng, "Towards efficient hierarchical designes by ratio cut partitioning," in *Proc. IEEE Int Conf. Computer-Aided Design*, pp. 298-301, 1989.

[5] H. H. Yang and D. F. Wong, "Optimal min-area min-cut replication in partitioned circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 1175-1183, November 1998.

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

[7] T.-Y. Choe, "linear time *k*-way network partitioning algorithm," Master's thesis, POSTECH, 1996.

[8] E. R. Barnes, "An algorithm for partitioning the nodes of a graph," *SIAM J.Algorithm and Discrete Method*, vol. 3, pp. 541-550, December 1982.

[9] B. Hendrickson and R. Leland, "Multidimensional spectral load balancing," in *Proc. 6th SIAM Conf. Parallel Proc.*, pp. 953-961, 1993.

[10] J.Y.Zien, M.D.F.Schlag, and P.K.Chan, "Multilevel spectral hypergraph partitioning with arbitrary vertex sizes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1389-1399, September 1999.

[11] C.-I. Park and Y.-B. Park, "An efficient algorithm for vlsi network partitioning problem using a cost function with balancing factor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, November 1993.

최 태 영

1991년 고려대학교 수학교육과 학사
1996년 포항공과대학교 대학원 컴퓨터공학과 석사
2002년 포항공과대학교 대학원 컴퓨터공학과 박사
2002년~현재 금오공과대학교 컴퓨터공학과 전임강사
관심분야 : 병렬 및 분산 알고리즘, 시스템 소프트웨어