An Efficient Algorithm for VLSI Network Partitioning Problem Using a Cost Function with Balancing Factor

Chan-Ik Park, *Member, IEEE,* and Yun-Bo Park

***Abstract*—This paper presents an efficient algorithm for network partitioning problem, which improves Fiduccia and Mattheyses' (F-M's) algorithm [1].**

We have noticed that the main problem of F-M's algorithm is that the cell move operation is largely influenced by the balancing constraint. In order to handle this kind of inherent limitation in F-M's algorithm, a cost function similar to the penalty function used in [12] is adopted which reflects balance degree of a partition as well as its cutset size. The weighting factor R is introduced in the cost function to determine the relative importances of the two factors: cutset size and balance degree. Using this cost function, we propose an iterative improvement algorithm which has the time complexity of $O(b(m + c^2))$, where b is the number of blocks, m is the size of network, and c is the number of cells. It is proven that the proposed algorithm guarantees to find a balanced partition if the value of R satisfies a certain condition. Experimental results show that the proposed algorithm outperforms F-M's algorithm in most cases.

***Index Terms*—Balance, Cost Function, Iterative improvement, Network, NP-hard, Partitioning.**

I. Introduction

GIVEN A SET of cells (or circuit elements) and the net-lists (or signal lines) that interconnect these components, the *network partitioning problem* is to partition these cells into two or more disjoint subsets (called blocks) of specified sizes such that the number of global nets (that will be called cuts) which have cells into two or more blocks is minimized. In general, the resulting partition is required to be balanced (e.g., the size of blocks are equal when all cells have the same size). This problem arises in various aspects of VLSI design automation.

It is known that network partitioning problem is NP-hard [2]. Therefore, attempts to solve this problem have concentrated on finding heuristic algorithms which yield near-optimal solutions in polynomial time. Conventionally, Fiduccia and Mettheyses' (F-M's) algorithm [1] has been widely used to solve this problem.

We have noticed that the main problem of F-M's algorithm is that the cell move operation is largely influenced by the balancing constraint. Though there have been several methods [3], [5], [8], [10], [11] improving F-M's algorithm, the main limitation still remains unsolved in their improvements.

This paper proposes a new algorithm for network partitioning problem. The main idea is to consider the balance degree of a partition (i.e., the size constraint of blocks) as well as the number of cuts simultaneously in order to transform current partition while all the previous algorithms have considered the balance degree separately from the number of cuts—they have kept balancing by prohibiting any operations that break balancing constraint during the entire execution of the algorithms [1], [4], [5], [8], [9].

Experimental results show that in most cases the proposed algorithm finds better partitions than F-M's algorithm within reasonable time. It is also shown that the performance of the proposed algorithm gets better as the number of blocks increases, the size of the network decreases, and the density of the input network becomes more sparse. It is proven that the worst case time complexity of the algorithm is $O(b(m + c^2))$, where b is the number of blocks, m is the size of the network, and c is the number of the cells in the network. In addition, this time complexity is verified by experiments showing the actual running time of the algorithm.

In Section II, we define some notations and terminologies for network partitioning problem. In Section III, we investigate some limitations of F-M's algorithm, on which we have focused to improve the algorithm. In Section IV, an improved algorithm and its time complexity are described. In Section V, we describe the sample networks used to test the performance of the algorithm and present the results obtained from some experiments. Finally, conclusions and further research directions are given in Section VI.

II. Network Partitioning Problem

Definition 2.1: A *cell C* is an object having its size similar to the node or vertex in tree or graph. A *net N* is

Manuscript received January 23, 1993. This work was supported in part by the Agency for Defence Development, the Pohang Institute of Science and Technology, and the Korea Science and Engineering Foundation. This paper was recommended by Associate Editor R. Otten.

C.-I. Park is with the Department of Computer Science, Pohang Institute of Science and Technology, P.O. Box 125 Pohang, 790-600, Korea.

Y.-B. Park was with the Department of Computer Science, Pohang Institute of Science and Technology, Pohang, Korea. He is now with the TICOM III Development Department, Sysem Division, Samsung Electronics Co., Seoul, Korea.

IEEE Log Number 9212585.

a set of cells whch are connected by the net. A net has its weight. A *pin* is a connection point between them.

Definition 2.2: A *network* is a hypergraph $H = (\mathcal{C}, \mathfrak{N})$, where $\mathcal{C}$ is a set of cells and $\mathfrak{N}$ is a set of nets.

A network was proposed as a proper model for electrical circuits by Schweikert and Kernighan [9]. For simplicity, we assume that the weights of nets in a network are all 1 in the following description.

Notation 2.1: For a given network $H = (\mathcal{C}, \mathfrak{N})$,

1) $c \equiv$ the number of cells.
2) $n \equiv$ the number of nets.
3) $S(C) \equiv$ the size of a cell C.
4) $\mathfrak{N}_C = \{N \mid C \in N\} \equiv$ the set of nets incident on a cell C.
5) $n_C = |\mathfrak{N}_C| \equiv$ the number of nets incident on a cell C.
6) $\mathcal{C}_N = \{C \mid C \in N\} \equiv$ the set of cells on a net N.
7) $c_N = |\mathcal{C}_N| \equiv$ the number of cells on a net N.
8) $p = \max_{C \in \mathcal{C}} (n_C) \equiv$ maximum number of nets on any cell.
9) $q = \max_{N \in \mathfrak{N}} (c_N) \equiv$ maximum number of cells on any net.
10) $W = \Sigma_{C \in \mathcal{C}} S(C) \equiv$ the total sum of all cell sizes.

Note that the size of $\mathfrak{N}_C$, n_C, is the same as the number of pins in the cell C, because a pin is a connection point of a cell and a net.

Definition 2.3: For a given network H, we define the size of the network, m, as the total number of pins in the network. Thus,

$$m = \sum_{C \in \mathcal{C}} n_C = \sum_{N \in \mathfrak{N}} c_N.$$

Definition 2.4: A *density*, d, of a network is defined as follows:

$$d = \frac{\sum_{N \in \mathfrak{N}} c_N(c_N - 1)}{c^2}.$$

The above formula was obtained by dividing the number of two-cell connections of a network by $c^2/2$ in order to normalize, where a two-cell connection corresponds to an edge of a graph equivalent to the network.

Definition 2.5: A *b-way partition* of a network is described by the b-tuple $(\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_b)$ where $\mathcal{A}_i \cap \mathcal{A}_j = \phi$ for $i \neq j$ and $\cup_i \mathcal{A}_i = \mathcal{C}$. $\mathcal{A}_i$ is said to be a *i*th *block* of the partition.

Definition 2.6: Given a partition, a net is said to be *cut* if it has at least one cell in more than one block and *uncut* otherwise. A *cutset* is a set of the cut nets. A size of a given cutset, i.e., the number of cut nets, is denoted by χ.

Definition 2.7: The *b-way network partitioning problem* consists of finding a partition of b blocks ($b \geq 2$) such that the cutset size is minimized while the block sizes are constrained to a certain size. If the block sizes are not constrained in the partitioning problem, then the trivial partition (where all cells are in one of the blocks) would always be the optimal solution.

In order to consider the size balance among blocks, we define the following notations and definitions.

Notation 2.2: We denote the size of a block $\mathcal{A}_i$ by $S(\mathcal{A}_i)$. Thus

$$S(\mathcal{A}_i) = \sum_{C \in \mathcal{A}_i} S(C).$$

Definition 2.8: [1] Given $r_1, r_2, \cdots, r_b$ satisfying that $0 \leq r_i \leq 1$ for each i and $\Sigma_{i=1}^{b} r_i = 1$, block $\mathcal{A}_i$ is said to be *acceptable* if

$$r_i W - \tau \leq S(\mathcal{A}_i) \leq r_i W + \tau$$

where $\tau = \max_{C \in \mathcal{C}} (S(C))$. That is, the size of $\mathcal{A}_i$ can be as much as "τ offset" from $r_i W$. It is called τ *tolerance*. In addition, a partition is said to be *acceptable* if all blocks are acceptable. Especially, an acceptable partition is called a *well-balanced* partition when $r_i = 1/b$ for $1 \leq i \leq b$. The above definition allowing more flexible balance condition was originally proposed in [1].

In fact, we can say that a partition is *perfectly-balanced* if the sizes of all the blocks are exactly the same. However, such a balanced partition does not always exist for a given network. Therefore the definition of "perfectly-balanced" has to be generalized as in [6].

Definition 2.9: [6] 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 balanced if

$$\sum_{\mathcal{A}_i, \mathcal{A}_j \in P, i \leq j} |S(\mathcal{A}_i) - S(\mathcal{A}_j)|$$

$$= \min_{P' \in \mathcal{P}} \left(\sum_{\mathcal{A}'_i, \mathcal{A}'_j \in P', i < j} |S(\mathcal{A}'_i) - S(\mathcal{A}'_j)| \right),$$

where $\mathcal{P}$ is a set of all possible partitions of a network.

In this paper, we deal with perfectly-balanced partitions. From now on, "balanced" means "perfectly-balanced" if not explicitly specified. In fact, previous algorithms have dealt only with acceptable (or well-balanced) partitions stated in Definition 2.8.

III. Limitations of F-M's Algorithm

Many approaches have been proposed for solving multiple-way network partitioning problem. Iterative improvement algorithms have been widely used since Kernighan and Lin's (K-L's) algorithm [4] was presented. They have been proved to be quite useful in many partitioning applications. In network partitioning problem, F-M's algorithm is the latest one. Of course, several methods [3], [5], [8], [10], [11] have been proposed to improve F-M's algorithm. None of these algorithms, however, cope with the limitations of F-M's algorithm which are described later because they tried to improve only by slight modifications of original algorithm or even the problem itself.

In the following discussion, F-M's algorithm includes its multiple-way version similar to Sanchis' algorithm [8] except multilevel gain though the original algorithm in [1] was developed only for two-way partitioning—in fact, we think that both can not be separated because two-way is a special case of multiple-way.

In network partitioning problem, the key factors to be considered are the cutset size and the balance of the partition. As briefly stated in Section I previous iterative improvement partitioning algorithms have been dealt with both factors separately. For example, K-L's algorithm preserves the balancing requirement by allowing pairwise exchange of cells of equal size between two blocks. As a result, the cell move operations are quite restricted though the resultant partition is always guaranteed to be balanced as long as the initial partition is balanced.

F-M's algorithm relaxes the cell move restriction by adopting one-cell move instead of pairwise (two-cell) exchange. This relaxation is made possible by allowing more flexibility in balance definition as shown in Definition 2.8. Nevertheless, F-M's algorithm has still some limitations. One is that the final partition of the algorithm may be unbalanced because it finds only well-balanced partition satisfying the condition described in Definition 2.8. The other is that since F-M's algorithm allows only the cell moves satisfying the size constraint, restrictions are still applied to the cell moves which affects the performance of the algorithm though less significantly than in K-L's algorithm. Ths is the most serious one that limits the performance of F-M's algorithm.

We have concentrated on solving these problems in order to improve F-M's algorithm. The problems provide us with main motivation to propose a new algorithm.

IV. Proposed Algorithm

In this section, an efficient algorithm is proposed which improves F-M's algorithm by adopting a new cost function with balancing factor.

4.1. Cost Function with Balancing Factor

Definition 4.1: For a given network H, consider any partition P. The *partition cost* of P consists of *cutset cost* which corresponds to the cutset size (i.e., the number of cuts), and *balancing cost* which represents balance degree of the partition. We denote the partition cost, cutset cost, and balancing cost by C_{PAR}, C_{CUT}, and C_{BAL} respectively. Then,

$$C_{\text{PAR}}(P) = C_{\text{CUT}}(P) + RC_{\text{BAL}}(P),$$

where $C_{\text{CUT}}(P) = \chi$, $C_{\text{BAL}}(P) = \Sigma_{\mathcal{a}_i, \mathcal{a}_j \in P, i<j} |S(\mathcal{a}_i) - S(\mathcal{a}_j)|$, and R called *balancing coefficient* is a positive weighting factor which determines the relative importance between cutset cost and balancing cost.

Based on the definition of the partition cost, the value of R is very important since the quality of resultant partition is determined by the value. For example, if the value is too small, the resultant partition may be unbalanced, whereas if it is too large, the cutset size may be very large. Therefore, we have to determine the proper value of R in order to obtain the balanced partition with minimum cutset size. It will be described later.

Definition 4.2: For a given network H, let a partition $P = (\mathcal{a}_1, \mathcal{a}_2, \cdots, \mathcal{a}_b)$. The *gain* of cell C in block $\mathcal{a}_i$ to block $\mathcal{a}_j$, $\gamma_{\mathcal{a}_j}$, represents the amount of partition cost to be decreased after C is moved to $\mathcal{a}_j$, $1 \le i \ne j \le b$. Then,

$$\gamma_{\mathcal{a}_j}(C) = C_{\text{PAR}}(P) - C_{\text{PAR}}(P'),$$

where P' is a new partition generated by the move of C. Similarly, the *cutset gain*, $\kappa_{\mathcal{a}_j}(C)$, and the *balancing gain*, $\pi_{\mathcal{a}_j}(C)$, are defined as follows respectively:

$$\kappa_{\mathcal{a}_j}(C) = C_{\text{CUT}}(P) - C_{\text{CUT}}(P'),$$

$$\pi_{\mathcal{a}_j}(C) = C_{\text{BAL}}(P) - C_{\text{BAL}}(P').$$

That is, we have

$$\gamma_{\mathcal{a}_j}(C) = \kappa_{\mathcal{a}_j}(C) + R\pi\, \mathcal{a}_j(C)$$

for all C and $\mathcal{a}_j$, where $C \in \mathcal{a}_i$, $1 \le i \ne j \le b$.

4.2. Multiple-Way Partitioning

For a given network $H = (\mathcal{C}, \mathcal{N})$, the problem is to partition $\mathcal{C}$ into b balanced subsets $\mathcal{a}_1, \mathcal{a}_2, \cdots$, and $\mathcal{a}_b$ such that the cost of the partition is minimized.

Notation 4.1: For a given network $H = (\mathcal{C}, \mathcal{N})$,

1) $S_{\min} = \min_{C \in \mathcal{C}} (S(C)) \equiv$ the minimum cell size.
2) $S_{\max} = \max_{C \in \mathcal{C}} (S(C)) \equiv$ the maximum cell size.
3) $S_{\text{avg}} = (\Sigma_{C \in \mathcal{C}} S(C)/c \equiv$ the average cell size.
4) $\delta_{\min} = \min_{C_i, C_j \in \mathcal{C}} (|S(C_i) - S(C_j)|) \equiv$ the minimum cell size difference.
5) $\delta_{\max} = \max_{C_i, C_j \in \mathcal{C}} (|S(C_i) - S(C_j)|) \equiv$ the maximum cell size difference.
6) $\delta_{\text{avg}} = (\Sigma_{C_i, C_j \in \mathcal{C}} |S(C_i) - S(C_j)|)/c^2 \equiv$ the average cell size difference.

Lemma 4.1: Given a network H, consider any two partitions $P_1 = (\mathcal{a}_{1_1}, \mathcal{a}_{1_2}, \cdots, \mathcal{a}_{1_b})$ and $P_2 = (\mathcal{a}_{2_1}, \mathcal{a}_{2_2}, \cdots, \mathcal{a}_{2_b})$. Let P_1 be more balanced than P_2. Then,

$$\sum_{1 \le i < j \le b} S(\mathcal{a}_{1_i}) S(\mathcal{a}_{1_j}) > \sum_{1 \le i < j \le b} S(\mathcal{a}_{2_i}) S(\mathcal{a}_{2_j}).$$

Proof: Since P_1 is more balanced than P_2,

$$\sum_{i<j} |S(\mathcal{a}_{1_i}) - S(\mathcal{a}_{1_j})| < \sum_{i<j} |S(\mathcal{a}_{2_i}) - S(\mathcal{a}_{2_j})| \quad (1)$$

(refer to Definition 2.9). And since both partitions come from the same network H,

$$\sum_i S(\mathcal{a}_{1_i}) = \sum_i S(\mathcal{a}_{2_i}) = \sum_{\in \mathcal{C}} S(C) = \text{constant.} \quad (2)$$

From (1) and (2),

$$\sum_{i<j} S(\mathcal{a}_{1_i}) S(\mathcal{a}_{1_j}) > \sum_{i<j} S(\mathcal{a}_{2_i}) S(\mathcal{a}_{2_j}). \qquad \blacksquare$$

The balancing cost, $C_{\text{BAL}}(P)$, is the measure of balance degree of a partition P, which means that the more balanced the partition is, the smaller its balancing cost is. Therefore, from Lemma 4.1, it is clear that the partition cost is rewritten as

$$C_{\text{PAR}}(P) = \chi - R \sum_{i<j} S(\mathcal{A}_i)S(\mathcal{A}_j).$$

Lemma 4.2: Given a network H, let a partition P^* be balanced and P' not balanced. The absolute difference of balancing costs of the two partitions, w, has the value in the range as follows:

$$1 \le w \le \left(\frac{W}{b}\right)^2 \frac{b(b-1)}{2}.$$

Proof: Let $P^* = (\mathcal{A}_1^*, \mathcal{A}_2^*, \cdots, \mathcal{A}_b^*)$ and $P' = \mathcal{A}_1', \mathcal{A}_2', \cdots, \mathcal{A}_b')$. Then the balancing costs of the two partitions are

$$C_{\text{BAL}}(P^*) = -\sum_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*)$$

and

$$C_{\text{BAL}}(P') = -\sum_{i<j} S(\mathcal{A}_i')S(\mathcal{A}_j'),$$

respectively. Since $C_{\text{BAL}}(P^*) < C_{\text{BAL}}(P')$, thus the difference of balancing costs of the two partitions is

$$\begin{aligned} w &= C_{\text{BAL}}(P') - C_{\text{BAL}}(P^*) \\ &= \sum_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*) - \sum_{i<j} S(\mathcal{A}_i')S(\mathcal{A}_j'). \end{aligned}$$

In all cases, the value of w is maximized when $S(\mathcal{A}_i^*) = W/b$ for all i, $1 \le i < b$, and $S(\mathcal{A}_i') = W$ and $S(\mathcal{A}_j') = 0$ for any i and j, $1 \le j \ne i \le b$. Then,

$$w = \left(\frac{W}{b}\right)^2 \frac{b(b-1)}{2}. \tag{3}$$

It is obvious that the lower bound of w must be one. ■

As previously stated, it is very important to fix the appropriate value of R in the cost function since the quality of a resultant partition highly depends on the value. Clearly, the value should be minimized while keeping the sizes of blocks balanced. Theoretically, we have the following theorem.

Theorem 4.1: A multipl-eway partition with minimum cost corresponds to a balanced minimum cutset if

$$R > n.$$

Proof: Let a partition $P^* = (\mathcal{A}_1^*, \mathcal{A}_2^*, \cdots, \mathcal{A}_b^*)$ have minimum cost C^*_{PAR}. Assume that P^* is not balanced. Now consider any balanced partition $P' = (\mathcal{A}_1', \mathcal{A}_2', \cdots, \mathcal{A}_b')$ of which the cost is C'_{PAR}. Then, from Lemmas 4.1 and 4.2, $\Sigma_{i \le j} S(\mathcal{A}_i')S(\mathcal{A}_j') = \Sigma_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*) + w$, where $1 \le w \le (W/b)^2 b(b-1)/2$. Since C^*_{PAR} is a minimum cost,

$$\begin{aligned} \chi^* - R \sum_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*) &\le \chi' - R \sum_{i<j} S(\mathcal{A}_i')S(\mathcal{A}_j') \\ &= \chi' - R \sum_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*) - Rw. \end{aligned}$$

Then, $\chi' - \chi^* \ge Rw$. This is contradictory since the minimum value of Rw is greater than n, while the maximum value of $\chi' - \chi^*$ is n. Therefore the partition P^* is balanced.

In order that C^*_{PAR} $(= \chi^* - R\,\Sigma_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*))$ may be minimum, χ^* must be minimum among those of all balanced partitions on H since the balancing cost $(= -\Sigma_{i<j} S(\mathcal{A}_i^*)S(\mathcal{A}_j^*))$ is constant for all balanced partitions and it is also maximum among all possible partitions of the network (refer to Lemma 4.1). This proves the theorem. ■

Unfortunately, a partition with minimum cost cannot be found within polynomial time [2]. We propose an efficient heuristic algorithm to find a partition with near-minimum cost within linear-time. The basic approach is similar to F-M's algorithm; starting with an arbitrary partition, improve it iteratively by choosing one cell from one block and moving it to another block (*one-move* operation). The cell to be moved is called *candidate cell* and is chosen such that a maximum decrease in the partition cost can be obtained (or minimum increase if no decrease is possible). The algorithm consists of a series of passes; in each pass, each cell is moved one by one until all c cells have been moved, where the cells to be moved are chosen among the ones that have never been moved during the pass. Consequently, we have c partitions produced during the pass and the one with the minimum cost is chosen as the starting partition for the next pass. The algorithm executes in several passes until no further reduction in partition cost is possible.

The following is a schematic of the algorithm.

```
    procedure PARTITION-NETWORK($H$, $b$, $R$)
    // input: a network $H = (\mathcal{C}, \mathcal{N})$, the number of blocks $b$, and
    //        balancing coefficient $R$
    // output: b-way partition $P = (\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_b)$.
    begin
1)      read network description;
2)      construct an initial partition $P$;
        // main loop for iterative improvement.
3)      repeat
4)          initialize the gains for all cells;
5)          DO-PASS($P$);
6)      until there is no improvement in cutset size
    end // end of procedure PARTITION-NETWORK.
    procedure DO-PASS ($P$)
    // input: current partition $P$
    // output: a new partition $P'$.
    begin
1)      for $i \rightarrow 1$ until $c$ do
            begin
2)              select a free cell with highest gain as a candidate cell;
                // let the candidate cell be $C$, its source block be $\mathfrak{A}_i$,
                // the destination block be $\mathfrak{A}_j$.
3)              assign cell $C$ to block $\mathfrak{A}_j$;
4)              lock cell $C$;
5)              update the cutset gains for the affected cells;
6)              update the balancing gains for all free cells
            end
        select the best partition among produced $c$ partitions and set $P'$;
7)      return $P'$
    end // end of procedure DO-PASS.
```

The main operation of the procedure DO-PASS includes the selection of a candidate cell based on cell gain (step 2) and the gain update of affected cells (step 5 and 6). The gain update operation must be designed efficiently because cell gains are used in the selection of a candidate cell. As far as the cutset gain is concerned, we can simply design an efficient update routine by extending the work of [8]. For more detail on the extended version, refer to [7]. We describe an efficient update routine only for the balancing gain in the following.

Lemma 4.3: In a partition $(\mathfrak{A}_1, \mathfrak{A}_2, \cdots, \mathfrak{A}_b)$, the balancing gain associated with moving a cell C from block $\mathfrak{A}_i$ to block $\mathfrak{A}_j$ is

$$\pi_{\mathfrak{A}_j}(C) = S(C)(S(\mathfrak{A}_i) - S(\mathfrak{A}_j) - S(C)).$$

Proof: The balancing cost of the partition is

$$\begin{aligned} C_{\mathrm{BAL}} &= -\sum_{k<l} S(\mathfrak{A}_k)S(\mathfrak{A}_l) \\ &= -\Bigg\{ \sum_{k<l \neq i,j} S(\mathfrak{A}_k)S(\mathfrak{A}_l) + \sum_{k \neq i,j} S(\mathfrak{A}_k)S(\mathfrak{A}_i) \\ &\quad + \sum_{k \neq i,j} S(\mathfrak{A}_k)S(\mathfrak{A}_j) + S(\mathfrak{A}_i)S(\mathfrak{A}_j) \Bigg\}. \end{aligned}$$

Let a partition $(\mathfrak{A}'_1, \mathfrak{A}'_2, \cdots, \mathfrak{A}'_b)$ be obtained by moving C from block $\mathfrak{A}_i$ to block $\mathfrak{A}_j$. Then the balancing cost of the new partition is

$$\begin{aligned} C'_{\mathrm{BAL}} &= -\sum_{k<l} S(\mathfrak{A}'_k)S(\mathfrak{A}'_l) \\ &= -\Bigg\{ \sum_{k \leq l \neq i,j} S(\mathfrak{A}_k)S(\mathfrak{A}_l) + \sum_{k \neq i,j} S(\mathfrak{A}_k)(S(\mathfrak{A}_i) \\ &\quad - S(C)) \\ &\quad + \sum_{k \neq i,j} S(\mathfrak{A}_k)(S(\mathfrak{A}_j) + S(C)) + (S(\mathfrak{A}_i) \\ &\quad - S(C))(S(\mathfrak{A}_j) + S(C)) \Bigg\}. \end{aligned}$$

Thus, in the partition $(\mathfrak{A}_1, \mathfrak{A}_2, \cdots, \mathfrak{A}_b)$, the balancing gain of C for $\mathfrak{A}_j$ is

$$\begin{aligned} \pi_{\mathfrak{A}_j}(C) &= C_{\mathrm{BAL}} - C'_{\mathrm{BAL}} \\ &= S(C)(S(\mathfrak{A}_i) - S(\mathfrak{A}_j) - S(C)). \end{aligned}$$ ■

A candidate cell, say C in $\mathfrak{A}_i$, is selected such that the cell gain of C, $\gamma_{\mathfrak{A}_j}(C)$, is maximum among all free cells. Each time a cell C is moved, both gains (cutset gain and balancing gain) should be updated for cells who are affected by the move of the cell C. Following lemma show how balancing gain is updated.

Lemma 4.4: Let a cell C be in block $\mathfrak{A}_i$. After C moves to block $\mathfrak{A}_j$, the balancing gains of all free cells are updated as follows: for each free cell C',

1) $C' \in \mathfrak{A}_i$

$$\pi^{\mathrm{new}}_{\mathfrak{A}_k}(C') = \pi^{\mathrm{old}}_{\mathfrak{A}_k}(C') - S(C)S(C'),$$

$$\pi^{\mathrm{new}}_{\mathfrak{A}_j}(C') = \pi^{\mathrm{old}}_{\mathfrak{A}_j}(C') - 2S(C)S(C'),$$

2) $C' \in \mathfrak{A}_j$

$$\pi^{\mathrm{new}}_{\mathfrak{A}_k}(C') = \pi^{\mathrm{old}}_{\mathfrak{A}_k}(C') + S(C)S(C'),$$

$$\pi^{\mathrm{new}}_{\mathfrak{A}_i}(C') = \pi^{\mathrm{old}}_{\mathfrak{A}_i}(C') + 2S(C)S(C'),$$

3) $C' \in \mathfrak{A}_k$

$$\pi^{\mathrm{new}}_{\mathfrak{A}_i}(C') = \pi^{\mathrm{old}}_{\mathfrak{A}_i}(C') + S(C)S(C'),$$

$$\pi^{\mathrm{new}}_{\mathfrak{A}_j}(C') = \pi^{\mathrm{old}}_{\mathfrak{A}_j}(C') - S(C)S(C'),$$

where $1 \leq k \neq i, j \leq b$.

Proof: The balancing gain updates of all cells are required only for the blocks the sizes of which are affected by the move—that is, blocks $\mathfrak{A}_i$ and $\mathfrak{A}_j$. Therefore, the following have to be considered.

1) $C' \in \mathfrak{A}_i$

a) $\pi^{\mathrm{new}}_{\mathfrak{A}_k}(C')$

$$\begin{aligned} &= S(C')\{(S(\mathfrak{A}_i) - S(C)) - S(\mathfrak{A}_k) - S(C')\} \\ &= S(C')\{(S(\mathfrak{A}_i) - S(\mathfrak{A}_k) - S(C')) - S(C)\} \\ &= \pi^{\mathrm{old}}_{\mathfrak{A}_k}(C') - S(C)S(C'), \end{aligned}$$

b) $\pi^{\text{new}}_{\mathcal{Q}_j}(C')$

$$= S(C')\{(S(\mathcal{Q}_i) - S(C)) - (S(\mathcal{Q}_j) + S(C)) - S(C')\}$$
$$= S(C')\{(S(\mathcal{Q}_i) - S(\mathcal{Q}_j) - S(C')) - 2S(C)\}$$
$$= \pi^{\text{old}}_{\mathcal{Q}_j}(C') - 2S(C)S(C').$$

2) $C' \in \mathcal{Q}_j$

a) $\pi^{\text{new}}_{\mathcal{Q}_k}(C')$

$$= S(C')\{(S(\mathcal{Q}_j) + S(C)) - S(\mathcal{Q}_k) - S(C')\}$$
$$= S(C')\{(S(\mathcal{Q}_j) - S(\mathcal{Q}_k) - S(C')) + S(C)\}$$
$$= \pi^{\text{old}}_{\mathcal{Q}_k}(C') + S(C)S(C'),$$

b) $\pi^{\text{new}}_{\mathcal{Q}_i}(C')$

$$= S(C')\{(S(\mathcal{Q}_j) + S(C)) - (S(\mathcal{Q}_i) - S(C)) - S(C')\}$$
$$= S(C')\{(S(\mathcal{Q}_j) - S(\mathcal{Q}_i) - S(C')) + 2S(C)\}$$
$$= \pi^{\text{old}}_{\mathcal{Q}_i}(C') + 2S(C)S(C').$$

3) $C' \in \mathcal{Q}_k$

a) $\pi^{\text{new}}_{\mathcal{Q}_i}(C')$

$$= S(C')\{S(\mathcal{Q}_k) - (S(\mathcal{Q}_i) - S(C)) - S(C')\}$$
$$= S(C')\{(S(\mathcal{Q}_k) - S(\mathcal{Q}_i) - S(C')) + S(C)\}$$
$$= \pi^{\text{old}}_{\mathcal{Q}_i}(C') + S(C)S(C'),$$

b) $\pi^{\text{new}}_{\mathcal{Q}_j}(C')$

$$= S(C')\{S(\mathcal{Q}_k) - (S(\mathcal{Q}_j) + S(C)) - S(C')\}$$
$$= S(C')\{(S(\mathcal{Q}_k) - S(\mathcal{Q}_j) - S(C')) - S(C)\}$$
$$= \pi^{\text{old}}_{\mathcal{Q}_j}(C') - S(C)S(C'). \quad ■$$

Though balanced partitions cannot be achieved in general, it is shown in Theorem 4.2 that there is a case that the proposed algorithm guarantees to find a balanced partition.

Theorem 4.2: When the cells of a network are all of the same size S, the proposed algorithm always finds a balanced partition on the network, if $R > p/S^2$.

Proof: Let the resulting partition from the algorithm be $(\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_b)$. Then, the value of the gain of each cell under the resulting partition must be nonpositive, because if there is any cell with a positive value of gain, the algorithm must transform the partition into another partition. Without loss of generality, assume that the partition is not balanced. Then $S(\mathcal{Q}_i) \geq S(\mathcal{Q}_j) + 2S$ for some i and j, $1 \leq i \neq j \leq b$. Then, for each cell C in $\mathcal{Q}_i$, $\gamma_{\mathcal{Q}_j}(C)$ is as follows:

$$\gamma_{\mathcal{Q}_j}(C) = \kappa_{\mathcal{Q}_j}(C) + RS(S(\mathcal{Q}_i) - S(\mathcal{Q}_j) - S)$$
$$= \kappa_{\mathcal{Q}_j}(C) + R\{S(S(\mathcal{Q}_i) - S(\mathcal{Q}_j)) - S^2\}$$
$$\geq \pi_{\mathcal{Q}_j}(C) + RS^2.$$

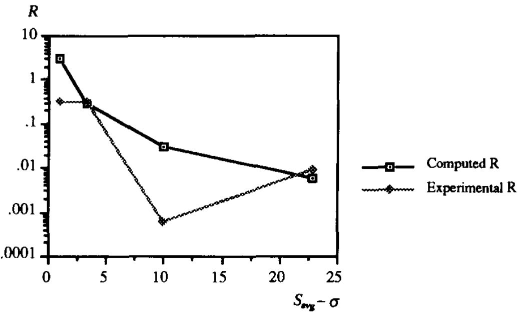

Fig. 1. Comparison of computed R and experimental R.

Since $-p \leq \kappa_{\mathcal{Q}_j}(C) \leq p$, $\gamma_{\mathcal{Q}_j}(C) > 0$. In other words, if the partition $(\mathcal{Q}_1, \mathcal{Q}_2, \cdots, \mathcal{Q}_b)$ is not balanced and $S(\mathcal{Q}_i) \geq S(\mathcal{Q}_j) + 2S$, then all the cells in $\mathcal{Q}_i$ have a positive gain. This is a contradiction. Therefore, the resulting partition is balanced. ■

In Theorem 4.2, we have shown that our algorithm guarantees to find a balanced partition if R is set to be larger than p/S^2 for the network of which all the cells are of the same size of S. However, for a general network with cells of different sizes, our algorithm does not guarantee to find a balanced partition. In fact, even finding equal sized blocks for a general network is NP-hard [2]. We have done some experiments to find the rough range of R that generates the balanced partition for general networks with cells of various sizes. In these experiments, we could see the fact that the minimum value of R which generates the balanced partition is proportional to the standard deviation of the sizes of the cells, σ, in the network and inversely proportional to their average size, S_{avg}. The value seems to be independent of the number of blocks. Thus we have used R value in the range of $R > p/(S_{\text{avg}} - \sigma)^2$ (it seems to be quite well working since if all cells in the network have the same size, then σ is 0 and thus $R > p/S^2_{\text{avg}}$, which is the same range as that in Theorem 4.2). Fig. 1 shows the relation between the values of R computed by above equation ($= p/(S_{\text{avg}} - \sigma)^2$) and the values obtained by experiments on several networks. The experimentally obtained values are the minimum ones that yield balanced partitions for each networks. We can see in Fig. 1 that the minimum value of R in the above range is quite a good estimate of the real value experimentally obtained.

The computing time per pass of the proposed algorithm is $O(b(m + c^2))$ since $O(c^2b)$ is needed to select candidate cell with maximum gain, and $O(mb)$ and $O(c^2b)$ are required to update cutset and balancing gain, respectively. For more detail, refer to [7].

V. Experimental Results and Discussions

In order to evaluate the performance of the proposed algorithm, several experiments were performed on three types of random networks: *sparse*, *normal*, and *dense* networks defined as networks with density (See Definition

TABLE I
RESULTS ON SPARSE NETWORK WITH 100 CELLS[1]

	Fiduccia & Mattheyses' Algorithm						*Proposed Algorithm*					
	Cutset Size				Mean	Mean	Cutset Size				Mean	Mean
b	min	max	mean	s.d	Pass	Time (s)	min	max	mean	s.d.	Pass	Time (s)
2	**16**	20	18.22	1.00	3.66	0.078	**16**	21	17.93	1.16	6.62	0.701
4	*26*	37	30.24	2.26	4.30	0.201	**23**	31	26.90	2.04	6.22	1.196
8	*35*	45	39.42	2.34	3.34	0.541	**28**	39	32.90	2.37	7.04	2.252
10	*35*	46	41.10	2.13	3.20	0.816	**30**	40	34.26	2.12	7.19	2.908
20	*40*	45	43.68	1.39	2.38	2.945	**36**	42	39.29	1.58	6.24	3.854

[1]Italic types in the column of minimum cutset from proposed algorithm represent the worse results than those from F-M's (s.d = standard deviation).

TABLE II
RESULTS ON NORMAL NETWORK WITH 400 CELLS

	Fiduccia & Mettheyses' Algorithm						*Proposed Algorithm*					
	Cutset Size				Mean	Mean	Cutset Size				Mean	Mean
b	min	max	mean	s.d	Pass	Time (s)	min	max	mean	s.d	Pass	Time (s)
2	**289**	301	296.56	2.82	5.10	0.926	*290*	305	298.00	2.81	8.50	14.432
4	*329*	350	338.38	4.58	6.12	2.082	**316**	331	322.64	3.53	12.06	37.448
8	*355*	369	361.76	3.39	4.34	4.675	**332**	349	340.79	3.32	8.64	57.847
10	*357*	369	364.16	2.64	3.84	6.320	**338**	358	346.88	3.72	9.84	65.566
20	*364*	375	371.24	2.22	3.10	21.586	**346**	364	355.38	3.73	10.24	108.821

TABLE III
RESULTS ON DENSE NETWORK WITH 1000 CELLS

	Fiduccia & Mettheyses' Algorithm						*Proposed Algorithm*					
	Cutset Size				Mean	Mean	Cutset Size				Mean	Mean
b	min	max	mean	s.d	Pass	Time (s)	min	max	mean	s.d	Pass	Time (s)
2	**1696**	1719	1708.04	4.43	7.04	7.698	*1702*	1724	1713.32	4.63	12.44	143.104
4	*1807*	1851	1825.80	9.37	9.32	14.920	**1763**	1806	1786.78	8.41	18.46	387.837
8	*1870*	1910	1892.24	7.88	6.62	29.218	**1811**	1844	1829.02	7.69	18.86	748.656
10	*1890*	1919	1904.56	7.56	5.86	38.088	**1821**	1858	1840.38	7.29	17.88	752.755
20	*1911*	1934	1923.78	4.65	4.26	102.491	**1848**	1878	1863.38	6.82	19.38	1374.786

2.4) $d \leq 0.1$, $0.1 < d < 0.5$, and $d \geq 0.5$, respectively. For each type, we used networks with 100, 200, 400, and 1000 cells. For each network, the algorithm was run to generate two-way, four-way, eight-way, 10-way, and 20-way partitions. We implemented F-M's algorithm for comparison referring to [1] for two-way partitioning and [8] for multiple-way partitioning.

The results obtained by executing both algorithms 50 times each with different initial partitions are shown in Tables I, II, and III for networks with 100, 400, and 1000 cells, respectively. Both the minimum and average cutset size were used for the purpose of careful comparison. It appears that in most cases the proposed algorithm is able to find smaller cutset than F-M's algorithm.

The graphs in Fig. 2 show the average improvements according to the number of blocks, network size, and network density, respectively. It is revealed from Fig. 2(a) that the improvement becomes highly significant as the number of blocks increases. All in all, we could say that the performance of the proposed algorithm gets better as the number of blocks increases, the size of the network decreases, and the density of the network becomes more sparse. Fig. 2 shows us that 3–9% average improvements can be obtained in most cases.

Note that the number of passes is about ten in most cases and slightly increases as the size of network increases. Fig. 3 shows the running time according to the number of blocks, the network size, and the number of cells (average running time over 50 runs). It is shown that the running time increases more or less linearly with the number of blocks and network size in both algorithms. The proposed algorithm usually takes more time than F-M's. We can see From Fig. 3 that the time complexity of the proposed algorithm is $O(b(m + c^2))$.

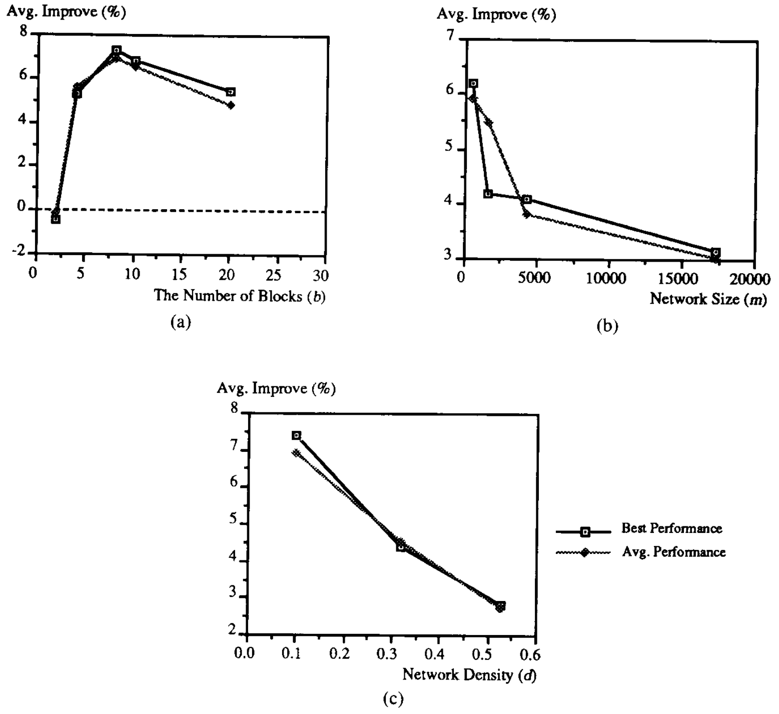

Fig. 2. Average improvements on F-M's algorithm; (a) for various number of blocks, (b) for various network size, and (c) for various network density.

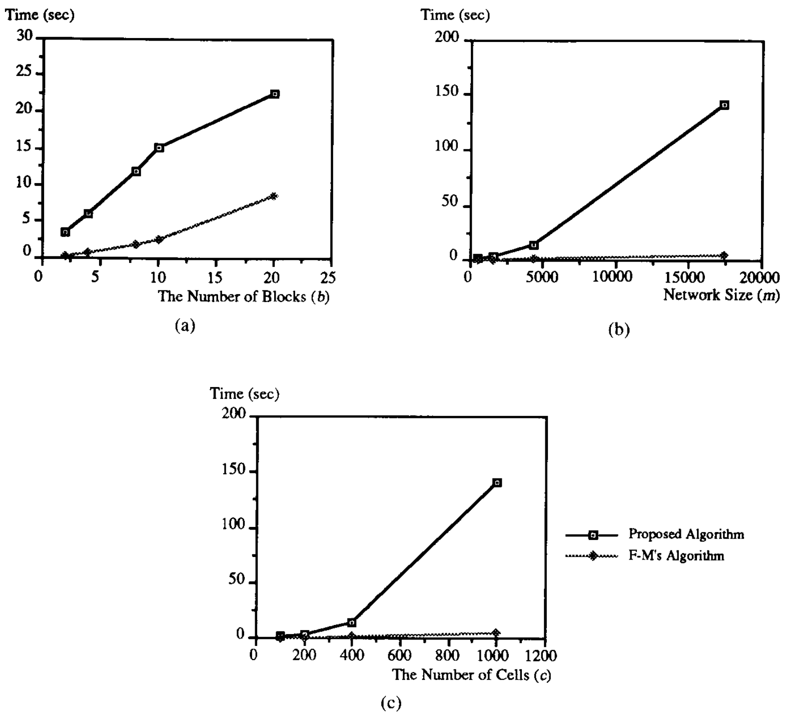

Fig. 3. Running time; according to (a) the number of blocks, (b) the network size, and (c) the number of cells.

VI. Conclusions

In this paper we have presented an improved algorithm to solve network partitioning problem, whose time complexity is $O(b(m + c^2))$. This improvement is achieved by adopting a new cost function which reflects balance degree of a partition as well as its cutset size. The weighting factor R called balancing coefficient was introduced into the new cost function in order to combine these two factors into one form.

The balancing coefficient R plays an important role in the proposed algorithm—the value determines the relative importance of the balancing constraint against the cutset size. It has been proven that the proposed algorithm guarantees to find the balanced partition if $R > p/S^2$. Note that the proposed algorithm is able to find the partitions with various balance constraints only by adjusting the R value.

Experiments showed that the proposed algorithm outperforms F-M's algorithm in most cases. We observed that the performance of the proposed algorithm gets better as the number of blocks increases, the size of the network decreases, and the density of the network becomes more sparse.

Further research requires more accurate investigations of R—the range of R value for an arbitrary network with cells of different sizes or for various block size constraints. Moreover, the function which determines the value of R dynamically to the passes is worth further research. The modification of gain to level gain, which was proposed in [5] and [8], may be used to find further improvements. Finally, other partition constraints which reflect the more practical VLSI design problems can be considered.

References

[1] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. 19th Design Automation Conf.*, Las Vegas, NV, June 1982, pp. 175-181.
[2] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman, 1979.
[3] M. K. Goldberg and M. Burstein, "Heuristic improvement technique for bisection of VLSI networks," Tech. Rep., IBM T. J. Watson Research Center, July 1983.
[4] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291-307, Feb. 1970.
[5] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438-446, May 1984.
[6] C. H. Lee, C. I. Park, and M. H. Kim, "An efficient algorithm for graph partitioning problem using a problem transformation method," *The CAD J.*, vol. 21, no. 10, pp. 611-618, Dec. 1989.
[7] Y. B. Park and C. I. Park, "An improved algorithm for VLSI network partitioning problem using a cost function with balancing factor," CS-TR-92-1, Dept. Computer Science, POSTECH, Feb. 1992.
[8] L. A. Sanchis, "Multiple-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 62-81, Jan. 1989.
[9] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. 9th Annual Design Automation Workshop*, 1972, pp. 57-62.
[10] C. Sechen and D. Chen, "An improved objective function for mincut circuit partitioning," in *Proc. Int. Conf. on Computer-Aided Design*, 1988, pp. 502-505.
[11] Y. C. Wei and C. K. Cheng, "Ratio cut partitioning for hierarchical designs," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 911-921, July 1991.
[12] D. S. Johnson, *et al.*, "Optimization by simulated annealing: An experimental evaluation: Part I, graph partitioning," *Operations Research*, pp. 865-892, 1989.

Chan-Ik Park (S'83-M'83-M'90) received the B.S. degree in electronics engineering from the Seoul National University, Seoul, Korea, in 1979, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Seoul, Korea, in 1985 and 1988, respectively.

He is currently an Assistant Professor in the Department of Computer Science, Pohang Institute of Science and Technology, Pohang, Korea. He was a visiting scientist in Massively Parallel Systems Group at IBM Thomas J. Watson Research Center in 1991. His research interests include operating system, parallel processing, distributed real-time system, and combinatorial optimization.

Yun-Bo Park received the B.S. degree from the Hanyang University, Seoul, Korea, in 1990, and the M.S. degree from Pohang Institute of Science and Technology, Pohang, Korea, in 1992, both in computer science.

In 1992, he joined the Samsung Advanced Institute of Technology, Kiheung, Korea, as a research engineer. Since April 1993, he has been working for Samsung Electronics. His research interests include parallel processing, neural networks, combinatorial optimization, and operating system.