FAST FLOORPLANNING FOR EFFECTIVE PREDICTION AND CONSTRUCTION

A. Ranjan[1] *K. Bazargan*[2] *M. Sarrafzadeh*[3]

[1,2,3]Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208-3118

[1]`abhi@ece.nwu.edu` phone: (847) 467-7852 fax:(847) 491-4455
[2]`kiarash@ece.nwu.edu` phone: (847) 491-9925 fax:(847) 491-4455
[3]`majid@ece.nwu.edu` phone: (847) 491-7378 fax:(847) 467-4144

ABSTRACT

Floorplanning is a crucial phase in VLSI Physical Design. The subsequent placement and routing of the cells/modules are coupled very closely with the quality of the floorplan. A widely used technique for floorplanning is Simulated Annealing. It gives very good floorplanning results but has major limitation in terms of running time. For more than tens of modules Simulated Annealing is not practical. Floorplanning forms the core of many synthesis applications. Designers need faster prediction of system metrics to quickly evaluate the effects of design changes. Early prediction of metrics is imperative for estimating timing and routability. In this work we propose a *constructive* technique for predicting floorplan metrics. We show how to modify the existing top-down partitioning based floorplanning to obtain a fast and accurate floorplan prediction. The prediction gets better as number of modules and flexibility in their shapes increases. We also explore the applicability of traditional Sizing Theorem when combining two modules based on their sizes and interconnecting wirelength. Experimental results show that our prediction algorithm can predict the area/length cost function normally within 5-10% of the results obtained by Simulated Annealing and is, on the average, thousand times faster.

1. INTRODUCTION

Given a circuit represented by a hypergraph, the *floorplanning problem* is to determine the approximate location of each module in a rectangular chip area. An important step in floorplanning is to decide the relative position of each module. A good floorplanning algorithm should minimize the total chip area, make the subsequent routing phase easy, and improve performance, by, for example, reducing signal delays. To facilitate subsequent routing phases, modules with relatively high connections should be placed close to one another. The set of nets thus defines the closeness of the modules. Placing highly connected modules close to each other reduces routing space/congestion. Wirelength minimization aims to do exactly that.

Several strategies have been adopted for floorplanning. The *rectangular dual graph* [22] approach to floorplanning is based on the proximity relation of a floorplan [14]. The algorithm for constructing a rectangular dual floorplan, if one exists, has been reported to run in linear time [1]. Other works on floorplanning by graph-dualization have been reported by Yeap et. al. [37][38]. The *hierarchical (bottom-up/top-down)* approach to floorplanning is based on a divide and conquer paradigm, where at each level of hierarchy, only a small number of rectangles are considered [4]. A hierarchical approach to floorplanning has been reported in [17], and research on this class of floorplans has some encouraging results [31][35][18][27]. The partitioning based hierarchical floorplanning has been studied in [4][15][16]. *Simulated Annealing* is a technique used to solve general optimization problems, floorplanning problems being one of them [32]. Simulated Annealing examines the configurations of the problem in sequence. Each configuration is a feasible solution of the optimization problem. Simulated Annealing is limited to very small number of modules (40 - 50) as reported in [33][34]. Simulated Annealing has been the best known floorplanner so far because others (like partitioning based hierarchical) have difficulty matching the shape. However, because of its semi-exhaustive nature Simulated Annealing takes substantial computation resources and is extremely slow.

Here, we address fast prediction of floorplan metrics. Floorplanning needs to be used in the heart of many synthesis applications (High Level Synthesis, Logic Synthesis for example). Designers want to evaluate the effects of early designing decisions on the final layout hence there is a real need for fast floorplan predictors. With large systems (hundreds of modules) Simulated Annealing fails. What is needed is a fairly accurate and very fast floorplanning technique to be able to convincingly predict the metrics of the layout i.e., area, wirelength, relative positions of the modules. The reduction in device sizes has lead to the integration of microprocessor and chipset. The concept of integrating whole system on a chip is fast gaining grounds. Quick floorplan predictors are needed in this domain as many different combinations exist for chipsets. It is predicted that the performance improvement in deep sub-micron, for the existing fabrication technologies, will come mainly from architecture and circuit [26]. Floorplan predictors that can assess viability of a chip at the architectural level are needed. Fast floorplan predictors will help the architects to decide the features and their impact on layout.

Prediction methods can broadly be classified into *statistical* and *constructive.* Major work on prediction has been towards *statistical* prediction. Wirelength estimation has been studied in [5][8][11][6][29] [23][25][10][19]. Kahng et. al. [2] discuss three basic types of wirelength estimations : *a priori* estimation is predicting wirelength of layout in advance, *a posteriori* estimation is used when for a fixed placement post-routing wirelength is predicted and *online* estimation attempts to predict wirelength as the hierarchical layout progresses. Rent's rule has been extensively studied and used for statistical estimation [24] [9][28]. Other works related to wirelength estimation have been e.g., MST-based [12], Bounding Box based methods [2]. Kahng et. al. [2] propose new bounding box estimators for on-line wirelength estimation and wirelength estimators using pure analytic techniques. The problem with statistical prediction is that circuits are not homogeneous. Circuits have been designed hierarchically and have high connectivity at low level. These features are difficult to model statistically.

Constructive prediction (algorithmic heuristics) has been given very little attention. General guideline on *constructive prediction* was outlined in SLIP'99 positional statement [21]. In this work we present constructive prediction of floorplan metrics (area/wirelength cost metric). Our claim is that fast algorithms can exploit the features of the system to act as better predictors and eventually as constructors. The difficult thing in floorplanning is shape-management (which Simulated Annealing does perfectly). Flexibility in representation of modules reduces shape management and lets one concentrate on other objectives (wirelength etc.). We propose a fast algorithm as a predictor based on modification of existing top-down hierarchical partitioning approach to floorplanning. We also show that the predictor can be extended to act as constructor when modules have high shape flexibility. Having constructive-predictor as the constructor enables one to control and guide the solution quality more effectively, a privilege that statistical predictors lack.

The rest of the paper is organized as follows. We present several relevant definitions in Section 2. Section 3 presents several models of flexibility. Various heuristics associated with our approach have been discussed in Section 4. Section 5 gives an outline of our algorithm. In Section 6 we discuss the time complexity of the algorithm. Section 7 discusses our experimental setup and results for prediction and construction. We present concluding remarks and direction for future work in Section 8.

2. DEFINITIONS

In this section we present definitions related to our work.

2.1. Flexibility

Flexibility, f, as the term indicates is a measure of how flexible a module is, in terms of shape and number of representations. The more flexible a module better are the chances of obtaining a compact floorplan or in other words more saving in dead-space. Intuitively we can say that if we have modules of infinite flexibility then we can obtain floorplans with zero dead-space.

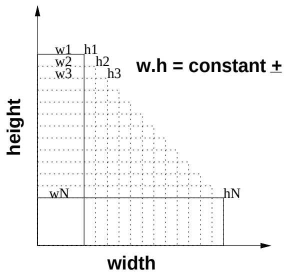

Figure 1. Shape curve of a flexible module

The shape curve for a flexible module [3][36] is shown in Figure 1. We did some experiments with TimberWolf to analyze the shape curve of some modules. We had TimberWolf place a cluster of cells in rectangular areas with varying aspect ratios and observed the width/height relation of the final placement. From our experiments and some real data on flexible modules [20], we observed that the shape curve is almost constant, i.e., $width.height = constant \pm X\%$, where X generally varies between 5 - 10 %.

2.2. Slicing Tree

A **slicing tree** is the binary tree representation of a sliceable floorplan. The leaves of the tree correspond to modules and an internal node defines how its child floorplans are combined to form a partial floorplan. Based on the type of internal node we define three types of slicing trees.

2.2.1. Empty slicing tree

As the name indicates, an *empty slicing tree* has vacant internal nodes, i.e., there is no cut or orientation (relative position of the children) associated with the internal nodes.

The floorplan for an empty slicing tree has just the positional information of the modules. No decision is yet made on how these modules will combine (vertical/horizontal) to produce (sub)floorplan. An example is shown in Figure 2.

2.2.2. Cut-based slicing tree

Each internal node of a *cut-based slicing tree* denotes the cut of the children nodes (vertical | or horizontal —). There is a complete freedom on the ordering of the child nodes.

The corresponding floorplan has only the cut information. The order in which modules will be placed (left/right for vertical sizing,

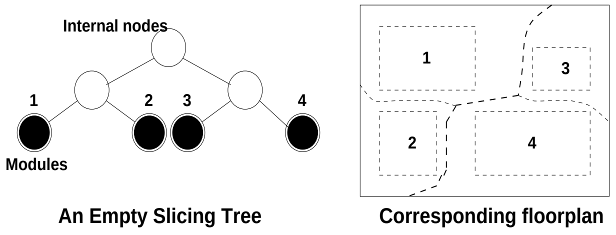

Figure 2. An Empty Slicing tree

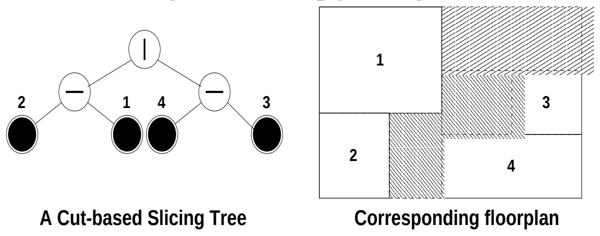

Figure 3. Cut-based slicing tree

top/bottom for the horizontal) is not yet fixed. Figure 3 shows such a slicing tree.

2.2.3. Oriented slicing tree

The *oriented slicing tree* is an extension of cut-based slicing tree where in addition to the cut the relative position (left/right, top/bottom) of the children nodes for an internal node is also fixed. We follow the convention that for the vertical cut, left child is placed left and right child is placed right. For the horizontal cut, left child is placed on the top and right child on the bottom.

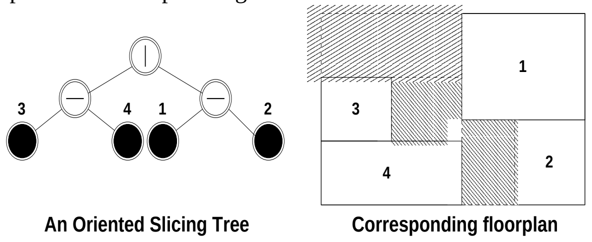

Figure 4. Oriented slicing tree

The corresponding floorplan is *complete* as far as the positions and the relative ordering of the modules are concerned. Oriented slicing tree and the corresponding floorplan is shown in Figure 4.

3. MODELING FLEXIBILITY

Enlisted below are some models to calculate flexibility of a module. As we shall see in the next section, an accurate assessment of flexibility of a module is important for our work.

3.1. Based on maximum dimensions of the module

We assume that we have an ordered listing of representations of modules, i.e., $w_1 < w_2 < w_3 < ..... < w_N$ and $h_1 > h_2 > h_3 > ..... > h_N$ where N is the number of distinct representations of the module.

One can say that flexibility in width is

$$f_{width} = w_{max}/w_{min}$$

and flexibility in height is

$$f_{height} = h_{max}/h_{min}$$

And total flexibility is

$$f_{maxD} = f_{width} + f_{height}$$

There are several shortcomings of this model. One can easily come up with examples where a module with only few distinct representations is estimated as more flexible than one with more number of representations.

3.2. Based on incremental sum

This model calculates the ratio of width and height of successive representations and sums them all to obtain flexibility

$$f_{overall} = \sum_{i=1}^{N} w_{i+1}/w_i + \sum_{i=1}^{N} h_i/h_{i+1}$$

Though this model fares better than the previous one in terms of giving a fairly accurate picture of flexibility but fails to take into account the number of distinct representations, N, of the modules.

3.3. Flexibility as a function of incremental sum and number of distinct implementations

What is needed is some kind of flexibility function F which depends on $f_{overall}$ and N,

$$F(f_{overall}, N)$$

We also need to define a transitive relation on this function to be able to say that one module is more flexible than the other, i.e.,

$$F_1(f_{overall}, N) \geq F_2(f_{overall}, N)$$

Also the property of sum '+' needs to be defined on this function to be able to sum the flexibilities of the modules, i.e.,

$$F_3(f_{overall}, N) = F_1(f_{overall}, N) + F_2(f_{overall}, N)$$

In general it is difficult to come up with a unique flexibility function which will satisfy the above two properties for all the combinations of modules and tell us with certainty about the flexibility of any module. The problem lies more with trying to define flexibility as a number than with the function itself.

For our work we have chosen the flexibility function to be

$$F(f_{overall}, N) = f_{overall}.N$$

This function can assess the flexibility of a module with fair amount of certainty.

We will show the application of these models on three 2x6 area modules where module1 has a rigid (rotatable) 2x6, 6x2 representations; module2 has 2x6, 4x3, 6x2 representations and module3 has 2x6, 3x4, 4x3, 6x2 representations.

Table 1. Comparing the models of flexibility

module no.	f_{maxD}	$f_{overall}$	$F(f_{overall}, N)$
module1	6	6	12
module2	6	7	21
module3	6	8.67	34.67

As is clear from the data in Table 1 f_{maxD} does not truly reflect the flexibility of the module. A module with many representations is estimated to be having the same flexibility as the rigid one. $f_{overall}$ though better than the previous one fails to take into account the number of distinct representations of the module and flexibilities as estimated by it are error-prone. The last one is based on the functional relationship between $f_{overall}$ and N. The function chosen by us , $f_{overall}$.N, can tell with fair amount of certainty about the flexibility of various modules. The slight change in flexibility is reflected as a large change in the number corresponding to that. Addition of even one extra representation contributes large to the flexibility number. We have experimented with these three flexibility models and indeed the last one, $f_{overall}$.N, produces the best results.

4. PREDICTION HEURISTICS

In the previous section we discussed how to obtain a good measure of flexibility of a module. In this section we discuss related heuristics which make prediction more effective. These heuristics are added to an existing top-down hierarchical partitioning approach, discussed in Section 5.

4.1. Balance with respect to area and flexibility

After obtaining a measure of the flexibility of a module it is imperative to balance the two partitions with respect to area and flexibility. Balance on area helps in saving dead-space and balance on flexibility helps shape-management by pairing up rigid modules with the flexible ones.

The observation is that *combining a flexible module with a rigid one is better than combining a rigid module with another rigid module.*

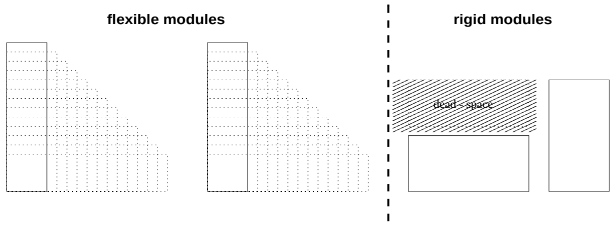

Figure 5. Partitioning based on area balancing

This argument is best explained by Figure 5. We partition the four modules into two regions, one region has all the flexible modules and the other one has all the rigid modules. Though these two partitions are balanced on area, we will get a lot of dead-space even if we take a lot of care in balancing area on further partitions. As is evident from the figure, the right half will result in a lot of dead-space. Though the modules in the left half will combine to produce a compact partition, the overall effect will be a bad floorplan because of the right half.

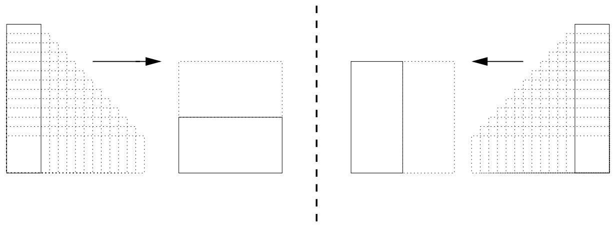

Figure 6. Partitioning based on flexibility plus area balancing

Now consider the case where the two regions were balanced on flexibility and area as shown in Figure 6. Here half of the flexible modules are on one side and half on the other. As is evident from the figure, the rigidity of the rigid module is annulled (may not be fully) by the flexibility of the flexible module. The overall effect is a much better floorplan than if we had balanced with respect to area only.

4.2. Sizing Theorem and Wirelength

Although area management is important, we also need to consider wirelength, for wirelength closely models routability [30]. We first present our lemma on sizing which forms an extension of the famous sizing theorem [18][27] *"given two subfloorplans, one with s and the other with t implementations, then the combination has at most $s + t - 1$ non-redundant implementations"*. The implementations of a module are *non-redundant* if for any pair of implementations i and j of the module either of these is true: $w_i < w_j$ and $h_i > h_j$ or $w_i > w_j$ and $h_i < h_j$, where w_i and h_i are the width and height of ith implementation.

Lemma : For any cut (vertical or horizontal) the set of non-redundant implementations of the combination contains all the minimum center-to-center wirelength implementations.

Proof: We consider vertical sizing (horizontal sizing follows a symmetric argument) as shown in Figure 7. Suppose the possible implementations of the modules A and B are $(a_1, b_1), ..., (a_s, b_s)$ and $(x_1, y_1), ..., (x_t, y_t)$, respectively. We can assume that the sets are sorted such that $a_i < a_{i+1}$ and $b_i > b_{i+1}$ for $i = 1, ..., s$

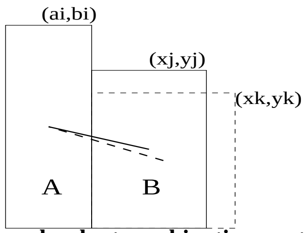

Figure 7. A non-redundant combination contains better wire-length

and similarly for x_js and y_js. Let us examine the implementations of the combination of A and B. Let (a_i, b_i) and (x_j, y_j) be the pairs selected to implement A and B respectively. The dimensions of the combination are given by $(a_i + x_j, max(b_i, y_j))$. Consider the case where b_i is the dominant height, that is, $max(b_i, y_j) = b_i$. Consider all pairs (x_k, y_k) with $k > j$. It is clear that all choices of (a_i, b_i) and (x_k, y_k) are inferior because (a_i, b_i) and (x_k, y_k) will not decrease the height but will increase the width. Thus, in the case when b_i is the dominant height of(a_i, b_i) and (x_j, y_j), all the implementations of (a_i, b_i) and (x_k, y_k) can be ignored for $k > j$. This way we have generated the non-redundant width and height implementations of the combination.

Thus far we have not included wirelength of the combination of the pair (a_i, b_i) and (x_j, y_j) when choosing non-redundant implementations. It may so happen that a redundant width and height implementation, pair (a_i, b_i) and (x_k, y_k) with $k > j$, has better center-to-center wirelength than the non-redundant pair (a_i, b_i) and (x_j, y_j). In the following discussion we will prove that such a thing can not happen. That is, the combination of (a_i, b_i) and (x_j, y_j) has better center-to-center wirelength than the redundant pair (a_i, b_i) and (x_k, y_k). For our wirelength model (center-to-center Euclidean distance) the distance between the centers of the non-redundant pair (a_i, b_i) and (x_j, y_j) is $(\frac{b_i - y_j}{2})^2 + (\frac{a_i + x_j}{2})^2$ and that between the redundant pair (a_i, b_i) and (x_k, y_k) is $(\frac{b_i - y_k}{2})^2 + (\frac{a_i + x_k}{2})^2$. Since $y_k < y_j$ and $x_k > x_j$ the center-to-center wirelength of the redundant combination is greater than that of the non-redundant combination. However, it is still possible that the redundant combination of (a_i, b_i) and (x_k, y_k) has better wirelength than some other non-redundant combination, say (a_l, b_l) and (x_m, y_m). Our claim is that even in such cases we can safely discard the combination of (a_i, b_i) and (x_k, y_k). The argument is that even if the redundant pair (a_i, b_i) and (x_k, y_k) has better wirelength than the non-redundant pair (a_l, b_l) and (x_m, y_m), we have a non-redundant combination (a_i, b_i) and (x_j, y_j) which has better wirelength and width/height than the redundant pair (a_i, b_i) and (x_k, y_k). Hence we can discard the pair (a_i, b_i) and (x_k, y_k) from our list of implementations of the combination of modules A and B. $\Box$

Hence we need to retain only non-redundant area implementations as given by the sizing theorem. Such a list of non-redundant implementations contains all the minimum wirelengths for the combination. The proof can easily be extended to Bounding Box based wire models.

Note that this proof is valid only when we are combining two modules. That is, this wirelength minimization scheme by retaining non-redundant implementations is at best a local one. For global wirelength minimization we may still need to retain all the s x t implementations of the modules A and B. However, it is easy to see that such a choice will result in exponential growth in the number of implementations of the final floorplan. For our work we are retaining the non-redundant implementations at all levels of sizing. That is, even **when combining two sub-floorplans (each of these may itself be a combination of several modules) we retain only the non-redundant implementations**.

4.3. Determining the Cut-Orientation

The major bottle-neck of any hierarchical floorplanning approach is how to decide on the cut (horizontal or vertical). There are several possible ways of combining modules in the system and till now there is no universally agreed upon technique for that. A most common practice is to pre-decide on the cut-orientation. However, this technique overlooks the importance of exploring the various cut possibilities. The commonly used techniques "of retaining the cut which produces least area for the combination" can not be used with flexible modules. We propose following heuristic for deciding the cut-orientation of a combination of flexible modules.

Choose the cut-orientation which makes aspect-ratio of the combination closer to 1

The equivalent aspect-ratio of a flexible module is the product of the aspect-ratios of its individual implementations. It is not hard to see that the equivalent aspect-ratio of a highly flexible module will be almost 1.

The motivation for desiring the aspect ratio to be closer to 1 is to reduce the perimeter of the final floorplan. A square shaped module will obviously have lesser perimeter than a rectangular module of same area. We evaluate the equivalent aspect-ratio of the combined module for both horizontal and vertical cut and retain that cut which gives an aspect-ratio closer to 1.

4.4. Length Minimization by Swapping Sub-floorplans

Our model of length estimation is based on the center-to-center Euclidean distance between the modules. Same model is being used in the Simulated Annealing [Wong-Liu] [33] floorplanner that we use for comparing our results.

The bi-partitioning approach may itself at times aid in increasing the wirelength. This kind of situation arises because after initial partitioning the sub-partitions behave as independent partitions and have no constraint on where the connected modules in the two sub-partitions be placed to minimize the wirelength. The situation is best depicted in Figure 8 where this independent partitioning has put connected modules in diagonally opposite blocks. One can loose highly on wirelength if this happens for several sub-partitions.

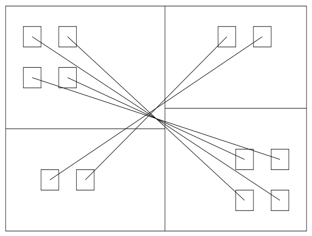

Figure 8. Bipartitioning may push some connected modules further apart

We propose following Partition-Swapping heuristic which can be utilized as a post-processing step to correct (may be partially) such undesirable situation.

Recursively swap two sub-floorplans (a top-down approach)

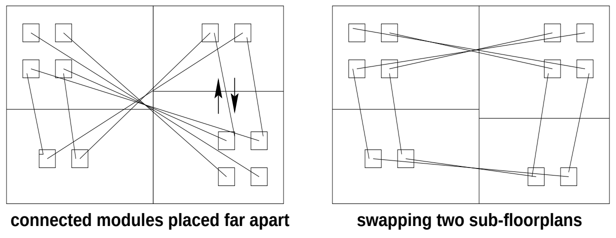
connected modules placed far apart
swapping two sub-floorplans

Figure 9. Swapping two sub-floorplans

The procedure is shown in Figure 9. The idea is to swap the sub-partitions (starting from top), retain the configuration that gives wirelength improvement and proceed to lower sub-floorplans. This may lead to substantial wirelength improvements (at times 30-40% improvement was observed).

5. A PREDICTION ALGORITHM

```
Partition ($G, V_0, g_0$)
# G is the netlist connectivity
# $V_0$ are the modules with group $g_0$
# $g_0$ is group 0 (initial group)
        $g_0 \rightarrow (g_1, g_2)$
            # group $g_0$ splits into $g_1$ and $g_2$
        Area+FlexiBalance($V_0$,$g_1$,$g_2$)
        $\Longrightarrow V_0 \rightarrow (V_1, V_2)$
                # hMetis used to partition with area
                # and flexibility of the two halves balanced
        Partition ($G, V_1, g_1$)
        Partition ($G, V_2, g_2$)
                    # recursively partition $V_1$ and $V_2$ till we
                    # reach the basic modules $m_1$ and $m_2$
Cut-orientation($list_1, list_2$)
# $list_1$ is the representation list of $m_1$
# $list_2$ is the representation list of $m_2$
        listVert $\leftarrow$ verticalCut($list_1$,$list_2$)
        listHori $\leftarrow$ horizontalCut($list_1$,$list_2$)
        return best(listVert,listHori)
                    # cut heuristic discussed earlier
Wire improvement(subfloorplan)
        leftHalf $\leftarrow$ left(subfloorplan)
        rightHalf $\leftarrow$ right(subfloorplan)
        Swap(leftHalf,rightHalf)
        if(improvement)
                $\Longrightarrow$ retain the configuration
        Wire improvement(leftHalf)
        Wire improvement(rightHalf)
                # recursively apply Wire improvement
```

Our algorithm is based on hierarchically partitioning the circuit in a top-down fashion. In the process various *sizing* and *wirelength improvement* heuristics (discussed in previous section) are applied to obtain a reasonably accurate floorplan in terms of area and wirelength cost. The algorithm has three main phases. The three phases demonstrate the building up of the floorplan through slicing trees, from empty to cut-based to oriented.

5.1. Hierarchical Partitioning Phase

The circuit is recursively bi-partitioned using a variation of Fiduccia-Mattheyses min-cut heuristic [7] called hMetis Hypergraph Partitioner [13]. The basic idea behind using min-cut based partitioning is to minimize the connectivity between the two partitions (a restriction imposed by the fact that wires take a substantial portion of the chip area). While partitioning we need to ensure that the two halves are balanced with respect to area and flexibility. In Section 4 we made several observations which justify the need for distributing flexible modules to ensure that rigid modules get paired with flexible ones.

5.1.1. Using hMetis as the partitioner

hMetis is a software package for partitioning large hypergraphs. The algorithms in hMetis (based on multilevel hypergraph partitioning) reduce the size of the graph (or the hypergraph) by collapsing vertices and edges, partition the smaller graph and then construct a partition for the original graph. These highly tuned algorithms allow hMetis to quickly produce high-quality partitions.

Since we are using the library functions provided by the hMetis package it is difficult to have both area and flexibility balance in the same hMetis run. Instead we run area-balanced hMetis several times and choose the most flexibility balanced run from these.

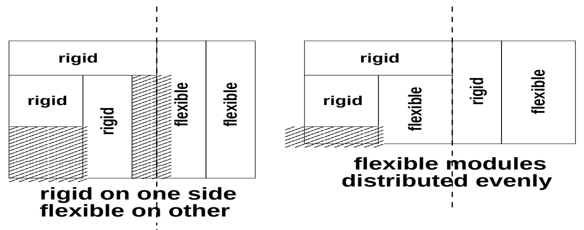

Figure 10. Distributing flexible modules to minimize deadspace

5.2. Deciding the cut-orientation

While recursively partitioning the circuit, as soon as we reach the sub-floorplan consisting of two modules we decide on the cut-orientation of this sub-floorplan based on the heuristics discussed earlier. Thus deciding the cut-orientation of the nodes proceeds in a bottom-up fashion.

In true sense the two phases, partitioning and cut-deciding, are coupled. We are building the empty slicing tree and the cut-based slicing tree at the same time. The top-down partitioning basically builds the empty slicing tree and the bottom-up sizing transforms it to a cut-based one. The cut-based slicing tree thus obtained has sizing information but the relative ordering of the modules/sub-floorplans is still undecided.

5.3. Wirelength Improvement Phase

The cut-based slicing tree has a default orientation (arbitrary). The final floorplan may itself have multiple non-redundant implementations. For any such implementation the sizes of the subsequent sub-floorplans/modules are unique. This phase does the job of assigning orientation to the internal nodes of the slicing tree so that the wirelength of the system gets minimized (for a particular floorplan size).

The slicing tree obtained after the partitioning and sizing of the nodes is a **cut-based** one. Till this step we do not consider the orientation of the internal nodes. The length minimization heuristic is basically trying the various orientations of the internal nodes in a greedy fashion (we always retain the current best and move to subsequent sub-partitions) to improve on the length.

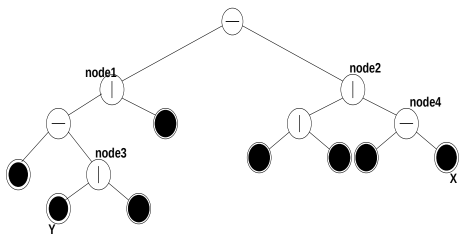

Figure 11. Slicing tree with default orientation

A typical situation that may arise because of the independent partitioning is shown in Figure 11. Here the two connected modules X and Y have been inadvertently put at a larger distance because of the independent partitioning discussed earlier.

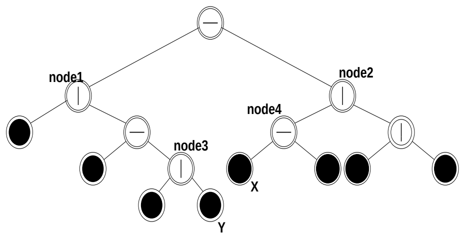

Figure 12. Orientation as we improve on length

By swapping proper internal nodes we can get rid of such situations. Of course any such sequence is retained only if it improves the overall length. The resulting **oriented** slicing tree is shown in Figure 12.

6. TIME COMPLEXITY OF THE ALGORITHM

The algorithm has following major steps,

6.1. Populating the data structures

With an efficient use of data structures the population time is guided by the number of (hyper)edges (hyperedge is defined as the edge with two or more modules connected to it) in the system n_e. Considering the fact that number of modules in each hyperedge is generally much smaller than the number of hyperedges itself, the time required for populating our graph data structure is $O(K.n_e)$ where K is a factor dependent on number of modules connected to each hyperedge (K is normally between 2 and 4).

6.2. Recursive bi-partitioning using hMetis and sizing

FM has been shown to be having a linear time complexity $O(t)$ [7] where t is the number of terminals in the system. Since hMetis also employs FM as its partitioner, the analysis of time complexity remains the same (other parts of hMetis partitioner such as clustering have linear time complexity). For slicing tree the height is $O(log(n))$, where n is the number of modules/vertices. Hence the contribution of recursive bi-partitioning to the complexity is $O(n.log(n))$. There are $O(n)$ internal nodes in the tree which need to be sized and their orientations need to be decided. If m is the average number of implementations for a module then the time complexity of sizing and deciding on the orientation of internal nodes is $O(mn.log(n))$.

6.3. Length minimization

We traverse down the slicing tree and try the swapping at each internal node. After any such swap we need to fix up the shapes of the nodes affected by the swap. This involves sizing of the affected nodes too. The complexity of such an operation is $O(mn.log^2(n))$.

7. EXPERIMENTAL SETUP AND TEST CIRCUITS

For our work we have chosen nine circuits (ind1, ind2, hway2, fract, prim1, prim2, prim1_small, prim2_small1, prim2_small2 and prim2_small3). The details about the number of modules and edges in the circuits are given in Table 2. The modules in our floorplan circuits are the cluster of cells from the circuits used in placement. We used TimberWolf1.4.0 to place these cluster of cells with varying aspect-ratios to generate shapes for the modules. The data for the shape and size of the flexible modules were generated based on the shape-curve given in Figure 1. The flexible modules in our floorplan circuits have aspect-ratios ranging from 3 to $1/3$. Each floorplan circuit has four different percentages of rigid modules in the system (From 10% to 50%. ind1 (10) implies that circuit ind1 has 10% rigid modules in it).

Table 2. Test Circuits

circuit	# of modules	# of edges
ind1	20	19
ind2	40	40
hway2	73	67
fract	149	147
prim1_small	252	553
prim1	750	830
prim2_small1	98	1410
prim2_small2	186	1482
prim2_small3	268	1536
prim2	2907	2961

Table 3. Comparing our implementation of Wong-Liu with TimberWolf

circuit	Wong-Liu		TimberWolf	
	area	time(s)	area	time(s)
ind1 (10)	10400	177	11858	156
ind2 (10)	26376	915	28322	863

Simulated Annealing floorplanner [Wong-Liu] [33] has been used for comparing the quality of our algorithm. We also compare the quality of our Wong-Liu floorplanner with TimberWolf1.4.0 floorplanner. Since the wirelength models of the two are different, we compare area and run-times. It turns out that our Wong-Liu Simulated Annealing floorplanner is as fast as TimberWolf Simulated Annealing floorplanner and the area results (TimberWolf produces a larger area because it leaves some space between modules for routing channels or pad spacing) are comparable, as given in Table 3. The main motivation for comparison with TimberWolf has been to make sure that our implementation of Wong-Liu floorplanner is fast and of good quality when compared to other Simulated Annealing implementations (we are reporting results for only two of the test circuits). All the experiments were carried out on Ultra Sparc 5 machine.

In general obtaining minimal values for all the objectives (area and wirelength etc.) of a multi-objective optimization problem is difficult. For a given floorplanning problem there may exist several optimal floorplans with different area and wirelength combinations. A commonly used metric for assessing floorplan quality is $area + \lambda.wirelength$, where λ is a balance factor.

We experimented with several values of λ, between $1/2$ and $1/4$, and found out that our prediction was consistent with that of Wong-Liu floorplanner. We are reporting results with λ having a constant value of $1/2$ so that we can compare the two approaches in a straight forward manner.

The results of Simulated Annealing [Wong-Liu] floorplanner, our predictor and constructor are given in Table 4 (we also present the results in form of bar-charts, Figure 13). The first column of Table 4 gives the name of the circuit and the number in the parentheses tells the percentage of rigid modules in that circuit. The floorplan metric ($area + \lambda.wirelength$) for the Wong-Liu floorplanner, predictor and constructor is given under the column “cost”. “% diff” column gives the percentage by which our predictor and constructor differ from Wong-Liu cost. “Speedup” column gives the factor by which our predictor and constructor are faster than Wong-Liu floorplanner.

In the following subsections we discuss the results associated with our prediction and construction.

7.1. The Algorithmic Predictor

In the previous sections we have discussed our algorithm and heuristics to make Partitioning based Floorplanning more effective, the objective being area and wirelength minimization. Our claim is (and the results show) that Partitioning based Floorplanning is a very good and fast predictor of system metrics (area-wirelength) and gets better as the shape management gets easier by

- increased number of soft modules/IPs.
- number of modules increases. So that area management of one module becomes less relevant.

The results are given in Table 4. As is clear from the results the quality of our predictor gets better with an increase in the number and flexibility of modules. It can be observed, in general for both Wong-Liu and Predictor, that the floorplan metric starts worsening with increase in the percentage of rigid modules in the system. However, even with varying percentage of rigid modules in the system, the floorplan metric as predicted by our modified partitioning based approach is almost always within $\pm 10\%$ to that by Wong-Liu Simulated Annealing floorplanner. The real gain is evident when we compare the speed. Our predictor is on the average 1000 times faster than Wong-Liu floorplanner.

7.2. Extending our Predictor to Constructor

Till now we have concentrated primarily on the prediction of system metrics by fast floorplanning. What is important in floorplanning with eased shape management ?

- It is not important to properly fit all the modules from the very beginning, like annealing does.
- It is important to avoid long wires (obviously bad for the length and other objectives, such as congestion).

Table 4. Comparing our Predictor and Constructor with Wong-Liu floorplanner. '*': For prim2 no results could be obtained even after 80 hours of Simulated Annealing

ckt(% rigid)	Wong-Liu		Predictor				Constructor			
	cost	time(s)	cost	% diff	time(s)	speedup	cost	% diff	time(s)	speedup
ind1 (10)	12039	177	13256	10.11	0.3	590	12182	1.18	8.6	21
ind1 (20)	12168	173	12871	5.78	0.3	577	12324	1.28	7.5	23
ind1 (30)	14971	122	18535	23.8	0.3	407	15400	2.86	5.9	21
ind1 (50)	16033	88	19935	24.34	0.3	293	17578	9.64	5.1	17
ind2 (10)	31010	915	31703	2.23	0.9	1017	30856	-0.50	46.5	20
ind2 (20)	32293	933	33711	4.40	0.8	1166	32242	-0.16	42.2	22
ind2 (30)	32705	956	34011	3.84	0.8	1195	31368	-4.10	49.4	19
ind2 (50)	33874	644	36327	7.24	0.8	805	34389	1.52	24.9	26
hway2 (10)	70571	3420	71981	2.00	1.7	2011	69611	-1.36	194	18
hway2 (20)	71838	3515	73390	2.16	1.7	2068	70448	-1.93	183	19
hway2 (30)	72274	3524	75530	4.50	1.6	2202	71900	-0.51	210	17
hway2 (50)	77653	2564	80273	3.37	1.6	1602	78696	1.34	105	24
fract (10)	131431	15651	129187	-1.7	4.5	3478	128388	-2.32	897	18
fract (20)	137044	12803	139125	1.52	4.6	2783	130984	-4.42	704	18
fract (30)	137084	14694	140192	2.27	4.6	3194	135869	-0.88	723	20
fract (50)	144072	9268	152436	5.81	4.7	1972	145392	0.91	549	17
prim1 (10)	831329	110491	863012	3.81	40.0	2762	832365	1.20	4629	24
prim1 (20)	867690	100010	864245	-0.4	39.4	2538	862657	-0.6	4911	20
prim1 (30)	870456	95299	881813	1.3	37.1	2569	871623	0.13	4214	23
prim1 (50)	897120	68303	957847	6.77	36.6	1866	931694	3.85	3617	19
prim1_small (10)	221847	35537	238420	7.47	15.4	2308	222788	0.42	2184	16
prim1_small (20)	218440	35303	220704	1.04	15.3	2308	206426	-5.5	2028	17
prim1_small (30)	219080	39221	215167	-1.8	15.8	2482	207357	-5.4	2142	18
prim1_small (50)	229675	27507	240214	4.60	15.4	1786	216396	-5.8	1772	16
prim2 (10)	*	*	4290382	*	660	*	3600380	*	67143	*
prim2 (20)	*	*	5100415	*	652	*	4113471	*	61157	*
prim2 (30)	*	*	5403593	*	658	*	4367985	*	55498	*
prim2 (50)	*	*	5802178	*	627	*	5128714	*	47902	*
prim2_small1 (10)	230703	11899	216478	-6.2	16.7	713	215193	-6.72	455	26
prim2_small1 (20)	235694	11141	227186	-3.6	16.9	659	217542	-7.70	439	25
prim2_small1 (30)	248317	9306	230093	-7.3	16.6	561	229349	-7.64	411	23
prim2_small1 (50)	249489	7445	254568	2.03	16.5	451	248566	-0.37	328	23
prim2_small2 (10)	323017	38416	299704	-7.21	29.8	1289	283188	-12.33	1477	26
prim2_small2 (20)	319897	30062	318005	-0.59	30.4	989	301768	-5.67	1389	22
prim2_small2 (30)	333313	29632	322356	-3.28	29.7	998	308023	-7.59	1506	20
prim2_small2 (50)	354387	21045	344175	-2.88	29.5	713	323188	-8.80	1138	18
prim2_small3 (10)	500150	69111	443082	-11.4	41.6	1661	437585	-12.51	3795	18
prim2_small3 (20)	508492	62206	450918	-11.3	41.4	1502	445128	-12.46	3542	18
prim2_small3 (30)	502128	59607	467333	-6.93	42.6	1399	445036	-11.15	3218	19
prim2_small3 (50)	531305	46333	510805	-3.86	42.7	1085	500750	-5.75	2864	16

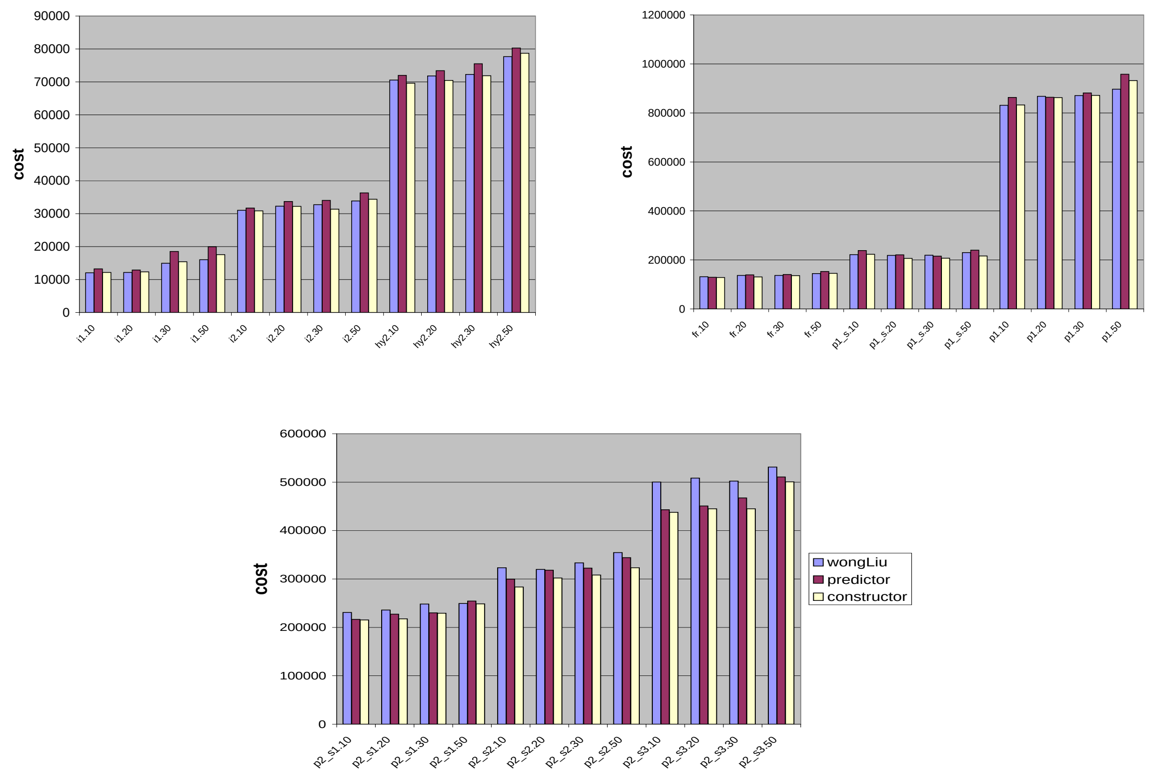

Figure 13. Bar-charts showing comparison of Wong-Liu with our appraoch

- However, in a small subset of the floorplanning area management is important.
- We need partitioning early on and then need to switch to length-area optimization at some point (use annealing for that).

We modify our predictor slightly to obtain a floorplan constructor even better than Wong-Liu Simulated Annealing floorplanner. We do Low Temperature Simulated Annealing as the post-processing on our predictor to obtain proper shape management. The partitioner has already done a good job of reducing wirelength. With large flexibility in the shapes of modules Low Temperature annealing in the end will further improve the floorplan by making it more compact. The results are given in Table 4. In almost all the cases we achieve a floorplan quality as good as or 5 - 10% better than the Wong-Liu floorplanner and at the same time achieving a speedup of 20 on the average.

8. CONCLUSION AND FUTURE WORK

As the design sizes grow by leaps and bounds, faster prediction of system metrics gains more and more importance. Floorplanning forms the early phase of Physical Design. System metrics at the floorplanning level are a good indicator of the quality of design. Unfortunately the best known floorplanner, Simulated Annealing, is extremely slow. In our work we have proposed an **algorithmic predictor** of floorplan metrics. Our work falls in the category of constructive prediction where we actually construct fast floorplans to estimate the quality of final floorplan. We have shown that the existing top-down hierarchical floorplanning approach can be modified suitably (through well defined heuristics) to obtain fast floorplans. Our floorplan predictor is almost thousand times faster than the traditional Simulated Annealing floorplanner and the prediction error is within 5 - 10 %. The quality of prediction gets better with the increase in number of modules and the flexibility in the representation of shapes of modules.

In this work our primary focus has been the prediction of area/wirelength floorplan metric. Further work in this direction will be to integrate prediction of other metrics like congestion and timing. We also intend to handle prediction under constrained floorplanning when certain module/pin locations are fixed.

REFERENCES

[1] J. Bhasker and S. Sahani. "A Linear Algorithm to find a Rectangular Dual of a Planar Triangulated Graph". *Algorithmica*, 3(2):274–278, 1988.

[2] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky. "On Wirelength Estimations for Row-Based Placement". In *International Symposium on Physical Design*, pages 4–11. ACM/SIGDA, 1998.

[3] S. Choi and C. Kyung. "A Floorplanning Algorithm Using Rectangular Voronoi Diagram and Force-Directed Block Shaping". In *International Conference on Computer-Aided Design*, pages 56–59, 1991.

[4] W. M. Dai and E. S. Kuh. "Simultaneous Floorplanning and Global Routing for Hierarchical Building-block Layout". *IEEE Transactions on Computer Aided Design*, 6(5):828–837, 1987.

[5] W. E. Donath. "Placement and Average Interconnection Lengths of Computer Logic". *IEEE Transactions on Circuits and Systems*, 26(4):272–277, 1979.

[6] W. E. Donath. "Placement and Average Interconnection Lengths of Computer Logic". In *IEEE Transactions on Circuits and Systems*, pages 272–277. IEEE, April 1979.

[7] C. M. Fiduccia and R. M. Mattheyses. "A Linear Time Heuristic for Improving Network Partitions". In *Design Automation Conference*, pages 175–181. IEEE/ACM, 1982.

[8] A. A. El Gamal. "Two-dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits". *IEEE Transactions on Circuits and Systems*, 28(2):127–138, 1981.

[9] C. V. Gura and J. A. Abraham. "Average Interconnection Length and Interconnect Distribution Based on Rent's Rule". In *Design Automation Conference*, pages 574–577. IEEE/ACM, 1989.

[10] T. Hamada, C. K. Cheng, and P. M. Chau. "A Wire Length Estimation Technique Utilizing Neighbourhood Density Equations". *IEEE Transactions on Computer Aided Design*, 15(8):912–922, 1996.

[11] W. Hebgen and G. Zimmermann. "Hierarchical Netlength Estimation for Timing Prediction". In *Physical Design Workshop*, pages 118–125. ACM/SIGDA, 1996.

[12] F. K. Hwang. "On Steiner Minimal Trees with Rectilinear Distance". In *SIAM Journal on Applied Mathematics*, pages 104–114, 1976.

[13] G. Karypis and V. Kumar. "hMetis: A Hypergraph Partitioning Package". Technical report, University of Minnesota.

[14] K. Kozminski and E. Kinnen. "Rectangular Dual of Planar Graphs". *Networks*, (15):145–157, 1985.

[15] U. P. Lauther. "A Min-cut Placement Algorithm For General Cell Assemblies Based On A Graph Representation". In *Design Automation Conference*, pages 1–10. IEEE/ACM, 1979.

[16] W. K. Luk. "Multi-Terrain Partitioning and Floorplanning for Data-Path Chip (Microprocessor) Layout". In *International Conference on Computer-Aided Design*, pages 492–495, 1989.

[17] W. K. Luk, P. Sipala, M. Tamminen, D. Tang, L. S. Woo, and C. K. Wong. "A Hierarchical Global Wiring Algorithm for Custom Chip Design". *IEEE Transactions on Computer Aided Design*, 6(4):518–533, 1987.

[18] R. H. J. M. Otten. "Efficient Floorplan Optimization". In *International Conference on Computer Design*, pages 499–503. IEEE/ACM, 1983.

[19] M. Pedram and B. Preas. "Interconnection Length Estimation for Optimized Standard Cell Layouts". In *International Conference on Computer-Aided Design*, pages 390–393. IEEE, 1989.

[20] M. Pedram and B. T. Preas. "Benchmarks for General Cell Floorplanning". *The Benchmark Archives at CBL, NCSU*.

[21] M. Sarrafzadeh and M. Wang. "Can Fast Algorithms be Used as Good Predictors ?". In *System Level Interconnect Prediction*, page 125, 1999.

[22] M. Sarrafzadeh and C. K. Wong. *"An Introduction to VLSI Physical Design"*, pages 50–54. McGraw-Hill, 1996.

[23] S. Sastry and A. C. Parker. "Stochastic Models for Wireability Analysis of Gate Arrays". *IEEE Transactions on Computer Aided Design*, 5(1):52–65, 1986.

[24] D. C. Schmidt. "Circuit Pack Parameter Estimation Using Rent's Rule". *IEEE Transactions on Computer Aided Design*, 1(4):186–192, 1982.

[25] C. Sechen. "Average Interconnection Length Estimation for Random and Optimized Placements". In *International Conference on Computer-Aided Design*, pages 190–193. IEEE, 1987.

[26] Naveed A. Sherwani. "SRC Top Ten Physical Design Problems". International Symposium on Physical Design, 1999.

[27] L. Stockmeyer. "Optimal Orientation of Cells in Slicing Floorplan Designs". *Information and Control*, 57(2):91–101, 1983.

[28] D. Stroobandt, H. Van Marck, and J. Van Campenhout. "An Accurate Interconnection Length Estimation for Computer Logic". In *Great Lakes Symposium on VLSI*, pages 50–55. IEEE, 1996.

[29] L. C. Suen. “A Statistical Model for Netlength Estimation”. In *Design Automation Conference*, pages 769–774. IEEE/ACM, 1981.

[30] M. Wang and M. Sarrafzadeh. “On the Behavior of Congestion Minimization During Placement”. In *International Symposium on Physical Design*, pages 145–150. IEEE/ACM, 1999.

[31] T. C. Wang and D. F. Wong. “An Optimal Algorithm for Floorplan Area Optimization”. In *Design Automation Conference*, pages 180–186. IEEE/ACM, 1990.

[32] D. F. Wong, H. W. Leong, and C. L. Liu. “*Simulated Annealing for VLSI Design*”. Kluwer Academic, 1988.

[33] D. F. Wong and C. L. Liu. “A New Algorithm For Floorplan Design”. In *Design Automation Conference*, pages 101–107. IEEE/ACM, 1986.

[34] D. F. Wong and C. L. Liu. “Floorplan Design for VLSI Circuits”. *Algorithmica*, 4:263–291, 1989.

[35] D. F. Wong and P. S. Sakhamuri. “Efficient Floorplan Area Optimization”. In *Design Automation Conference*, pages 586–589. IEEE/ACM, 1989.

[36] D. F. Wong and K. The. “An Algorithm for Hierarchical Floorplan Design”. In *International Conference on Computer-Aided Design*, pages 484–487, 1989.

[37] K. H. Yeap and M. Sarrafzadeh. “Floorplanning by Graph Dualization: Two-concave Rectilinear Modules”. *SIAM Journal on Computing*, 22(3):500–526, 1993.

[38] K. H. Yeap and M. Sarrafzadeh. “Sliceable Floorplanning by Graph Dualization”. *SIAM Journal on Discrete Mathematics*, 8(2):258–280, 1995.