Globally Optimal Floorplanning for a Layout Problem

Teng-Sheng Moh, *Member, IEEE,* Tsu-Shuan Chang, and S. Louis Hakimi

***Abstract*—In this paper, the floorplanning problem for a layout problem is formulated as a global optimization problem. The area of each building block is assumed to be fixed. However, its width and height are allowed to vary subject to aspect ratio constraints. Also, a block may be arbitrarily oriented in parallel to the xy orthogonal axes, subject to partition constraints with associated adjacency relationships. The objective is to minimize the rectangular area of the entire layout. By formulating the problem appropriately, it becomes a geometric programming problem. Its global minimum can then be found by using standard convex optimization techniques. The problem formulation and its conversion to a convex optimization problem are first illustrated through a simple example. The general procedure is then described and the effectiveness of the approach demonstrated through numerical examples.**

I. Introduction

CIRCUIT layout is part of the VLSI design process. Briefly speaking, the final product of the circuit design is a set of functional modules (blocks) which are rectangles in shape, or shapes composed of two or three rectangles pasted together. The objective of circuit layout is to first place the modules on different parts of the chip and then to connect their pins by mutually noninterfering wires according to a given wiring list.

There are different ways for dividing the layout problems into steps [1], [3], and [6]. Roughly speaking, these steps are partitioning, placement, and routing. First, partitioning is a process of creating the physical hierarchy of the hardware elements used to realize the design; it is essentially the subdivision of the logic complexes into smaller subcomplexes [6]. Second, the physical rectangular building blocks are placed into the subrectangles such that blocks do not overlap and their relative position is preserved. Finally, routing is used to interconnect the pins.

Once a partition is given, there are various ways to perform placement (e.g., [7], [8], and [10]). In [10], the placement problem is formulated as an optimization problem. The area of a building block is assumed to be fixed. However, the width and the length of its block are allowed to vary, and a block may be arbitrarily oriented in parallel to the xy orthogonal axes, subject to partition constraints. This type of placement is usually called floorplanning; the objective of floorplanning is to minimize the rectangular area of the entire layout. Although their framework does not take care all the factors, an interesting and unexpected result of the existence and uniqueness of a zero wasted area layout has been proven.

However, there are two major restrictions in their framework. First, once the aspect ratio is considered, they immediately face with a global optimization problem which is typically difficult, if not impossible, to solve. Another drawback in their approach is the violation of the adjacency relationship as illustrated later. In this paper, we will present a way to solve the global optimization problem of the floorplanning which is computationally efficient and does not violate the required adjacency relationship.

Generally speaking, what we face with in such a floorplanning problem is a global optimization problem. To deal with such problems, either we can reformulate the problem in an easier solvable format or we need to directly solve the global optimization problem. In this paper, we shall deal with the problem in [10] by reformulating its model and including adjacency constraints and aspect ratios. Although it seems impossible to solve such a global optimization, we can actually formulate it appropriately as a geometric programming problem. Since a geometric programming problem can be transformed into an easier convex optimization problem, it can be solved by using any standard convex optimization technique. Therefore, we can obtain globally optimal solution for the floorplanning problem.

Since the main contribution of the paper is on the formulation of a seemingly insoluble floorplanning problem into a solvable problem, we shall concentrate on the illustration of how the problem is formulated in Section II through simple examples, just to get the main ideas across. In Section III, the general problem formulation is presented by extending the steps presented in Section II. In Section IV, numeral examples successfully illustrate the effectiveness of our approach. In Section V, a short conclusion is given.

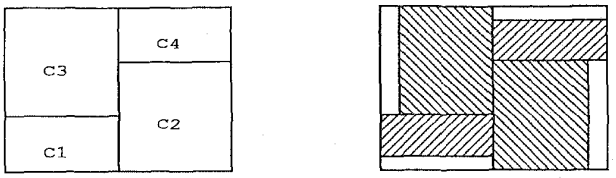
II. Overview

To introduce our approach, we first use a simple example to discuss our motivation, to outline the problem formulation, and to present its associated solution procedure. The general problem formulation and procedures will be presented in Section III corresponding to each step given in the simple example.

Manuscript received September 27, 1994. A patent application including part of the materials in the paper is in progress. This paper was recommended by Associate Editor S. Mattisson.

The authors are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA.

Publisher Item Identifier S 1057-7122(96)06851-1.



(a) (b)

Fig. 1. The simple example.

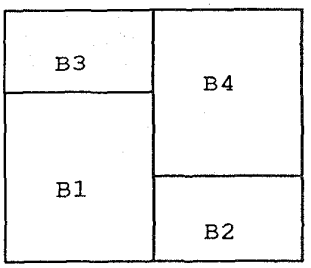


Fig. 2. A different partition.

A. Motivation

In the placement problem considered, a partition such as that in Fig. 1(a) is given *a priori*. Each cell i in the partition has an associated building block with area a_i. The aspect ratio of each building block is required to lie within a specified range $[m_i, M_i]$. In a feasible placement such as that in Fig. 1(b), each cell is big enough to contain its building block (shaded area). Our objective is then to find a feasible placement with minimum overall area.

Let us start with a brief description of the model presented in [10], where each box has the same size as its corresponding building block. A given partition (floorplan) is first converted into a graph representation. An optimization problem is then derived from the representation. Consider the given partition in Fig. 1(a), where building block i has width w_i and height z_i. The width of the enclosing rectangle is v and height h. By following the procedure [10], we can obtain the following optimization problem.

$$\min\ v \cdot h \tag{2.1}$$

subject to the following constraints: area constraints

$$w_i z_i = a_i, \quad i = 1, \cdots, 4, \tag{2.2}$$

width constraints

$$\begin{aligned} w_1 + w_2 &\le v, \\ w_3 + w_4 &\le v, \\ w_3 + w_2 &\le v, \end{aligned} \tag{2.3a}$$

height constraints

$$\begin{aligned} z_1 + z_3 &\le h, \\ z_2 + z_4 &\le h, \end{aligned} \tag{2.4a}$$

aspect ratio

$$m_j \le \frac{z_j}{w_j} \le M_j, \quad j = 1, \cdots, 4. \tag{2.5}$$

In [10], the aspect ratio is not considered when their solution was derived. They can then prove the existence and uniqueness of zero wasted area layout under their framework. This may end up with a very narrow building block. Furthermore the procedure presented does not preserve the adjacency relationship in the original partition. For example, although the placement in Fig. 2 will give us an optimization problem with a different width constraints, the height constraint is the same. In other words, their height constraint can not distinguish the two different adjacency relationship in Figs. 1(a) and 2.

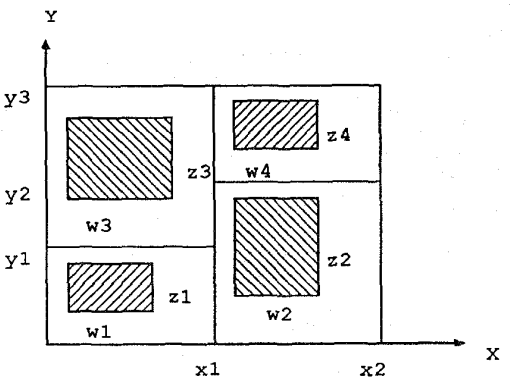


Fig. 3. Problem reformulation.

B. Problem Formulation

Our goal in this paper is to solve the complete placement problem (2.1) subject to all the constraints from (2.2)–(2.5) and additional adjacency requirements. To deal with the adjacency relationship, we label the given partition in Fig. 1(a) as shown in Fig. 3. Given the xy coordinate system, x_i represents the distance of ith vertical line segment from the origin and similarly y_j is defined. The partition constraints with correct adjacency relationship can now be obtained as follows. Please note that they can be obtained by following the procedure in Section III.

$$x_1 \le x_2 \quad \text{or} \quad x_1 x_2^{-1} \le 1 \tag{2.6a}$$

$$y_1 \le y_2 \quad \text{or} \quad y_1 y_2^{-1} \le 1 \tag{2.6b}$$

$$y_2 \le y_3 \quad \text{or} \quad y_2 y_3^{-1} \le 1. \tag{2.6c}$$

If Fig. 3 is indeed a feasible partition, we can place each building block within its corresponding cell, as indicated by the shaded area. Note that w_i and z_i represents the width and height of building block i, $i = 1, \cdots 4$, respectively. We then have the area and aspect ratio constraints as usual. We also have containment constraints since each building block has to be contained within its corresponding cell.

area constraints

$$w_i z_i \geq a_i, \quad \text{or} \quad a_i w_i^{-1} z_i^{-1} \leq 1, \text{ for all } i. \tag{2.7}$$

aspect ratio constraints

$$m_i \leq \frac{z_i}{w_i} \leq M_i \quad \text{or}$$

$$\{M_i^{-1} w_i^{-1} z_i \leq 1 \quad \text{and} \quad m_i w_i z_i^{-1} \leq 1\} \quad \text{for all} \quad i \tag{2.8}$$

containment constraints

$$\{x_1 \geq w_1,\ y_1 \geq z_1\} \quad \text{or} \quad \{w_1 x_1^{-1} \leq 1,\ z_1 y_1^{-1} \leq 1\} \tag{2.9a}$$

$$\{x_2 - x_1 \geq w_2,\ y_2 \geq z_2\} \text{or} \{w_2 x_2^{-1} + x_1 x_2^{-1} \leq 1,\ z_2 y_2^{-1} \leq 1\} \tag{2.9b}$$

$$\{x_1 \geq w_3,\ y_3 - y_1 \geq z_3\} \text{or} \{w_3 x_1^{-1} \leq 1,\ z_3 y_3^{-1} + y_1 y_3^{-1} \leq 1\} \tag{2.9c}$$

$$\{x_2 - x_1 \geq w_4,\ y_3 - y_2 \geq z_4\} \text{ or } \{w_4 x_2^{-1} + x_1 x_2^{-1} \leq 1,\ z_4 y_3^{-1} + y_2 y_3^{-1} \leq 1\}. \tag{2.9d}$$

Finally, the physical constraints demand all the quantities to be nonnegative, i.e.,

$$\begin{aligned} w_i &\geq 0, \\ z_j &\geq 0, \\ x_h &\geq 0, \\ y_k &\geq 0 \quad \text{for all} \quad i, j, h, k. \end{aligned} \tag{2.10}$$

Our placement problem is then to minimize the total area $x_2 y_3$,

$$\min_{w_i, z_j, x_h, y_k} x_2 y_3 \tag{2.11}$$

subject to all the constraints from (2.6)–(2.10).

C. Numerical Solution via Geometric Programming

Although the formulated problem seems highly nonlinear and to be a bona fide global optimization problem, it is actually a geometric programming problem. The geometric programming problem, after a simple transformation, can be converted into a convex optimization problem, which in turn can be solved efficiently by nonlinear programming. To see this, let us consider the transformation

$$\begin{aligned} w_i &= e^{\overline{w}_i}, \\ z_i &= e^{\overline{z}_i}, \\ x_i &= e^{\overline{x}_i}, \\ y_i &= e^{\overline{y}_i}, \\ a_i &= e^{\overline{a}_i}, \\ m_i &= e^{\overline{m}_i}, \\ M_i &= e^{\overline{M}_i}, \quad \text{for all} \quad i, \end{aligned} \tag{2.12}$$

and take the natural logarithm on both sides of the inequality constraints from (2.6)–(2.9), and also on the cost function.

We have the optimization problem with linear cost function

$$\min_{\overline{w}_i, \overline{z}_j, \overline{x}_h, \overline{y}_k} \overline{x}_2 + \overline{y}_3 \tag{2.13}$$

subject to all linear constraints derived from the original constraints (2.6)–(2.8), and those linear and convex constraints derived from (2.9). Some of the inequalities in (2.9) are not transformed into linear constraints. For example, the first inequality of (2.9d) is transformed into

$$\log\left(e^{\overline{w}_4 - \overline{x}_2} + e^{\overline{x}_1 - \overline{x}_2}\right) \leq 0. \tag{2.14}$$

It has been proved in [4] that the function in the left side of (2.14) is convex, and thus we have a convex optimization problem. Note also that, after the transformation (2.12), the variables are not necessarily nonnegative.

III. General Problem Formulation

There are two issues involved in the floorplanning problem. First we need to show that the globally optimal solution can be theoretically obtained by using our framework. Second we have to indicate if our approach is computationally feasible. Regarding the first issue, we will explain in this section how the general problem formulation follows the simple example in Section II-B as mentioned. To answer the second question, we will present various numerical examples in the next section.

Let us now consider the general problem formulation. The area, aspect ratio and physical constraints are the same as (2.7), (2.8), and (2.10). The most general containment constraint takes the same form as those in (2.9).

$$x_i - x_j \geq w_a, \quad \text{or} \quad y_h - y_k \geq z_a. \tag{3.1}$$

Equivalently, we have

$$w_a x_i^{-1} + x_j x_i^{-1} \leq 1 \quad \text{or} \quad z_a y_h^{-1} + y_k y_h^{-1} \leq 1. \tag{3.2}$$

For the partition constraints such as (2.6), we can obtain them directly from the given floor plan. Let us demonstrate how this can be done for the y-coordinate. For given two cells, they are only four possible cases for their adjacency relationship as shown in Fig. 4. The corresponding partition constraint with the correct adjacency relationship are as follows: a) Case 1: $y_3 \leq y_4$; b) Case 2: $y_1 \leq y_2$; c) Case 3: $y_3 \leq y_2$; and d) Case 4: $y_1 \leq y_4$.

Typically, we require the overlapping between two adjacent cells having a minimum contact, say "gap" units, the partition constraint then has the format below.

$$x_i + \text{gap} \leq x_j, \quad \text{or} \quad y_h + \text{gap} \leq y_k. \tag{3.3}$$

Note that all the partition constraints have the same form as those in (2.6). Thus the transformation used in Section II-C can still be applied to obtain an equivalent convex optimization problem.

So far, we have shown that the floorplanning problem can be formulated as the geometrical programming problem. For a

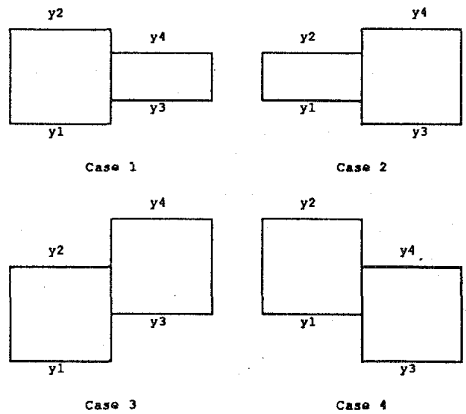


Fig. 4. Partition constraints for y-coordinate.

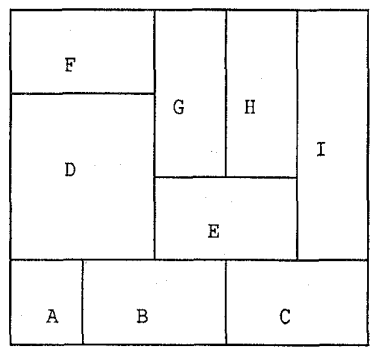


Fig. 5. The given partition for Section III and Example 2.

given partition, we need to generate its corresponding geometrical programming problem, and then solve it computationally. In the balance of the section, we will explain how the former can be performed by using an $O(n)$ algorithm. The latter will be illustrated in the next section.

To generate the problem, we basically follow the derivation in [2]. In [2], a partition is constructed from a given geometric figure. In this paper, the partition is given. We first represent the given partition by a geometric figure, and then follow the procedure in [2] to reconstruct the partition. In the process, the partition can be represented as two acyclic graphs. From the graphs, the partition constraints can be uniquely determined. Since [2] is not easily accessible, we repeat the relevant material from it for completeness and convenience.

A geometric figure $GF[S, W]$ is an abstract representation of a partition in the (X, Y)-plane, where S is a set of squares representing the cells, and W is a set of straight-lines representing the adjacencies between cells [9]. Moreover, bnd represents the enclosing rectangle of the cells. In the geometric figure, bnd is represented by four small circles representing the four sides of bnd. These four sides are labeled bnd^B, bnd^T, bnd^R, and bnd^L which are placed counter-intuitively at the top, bottom, left, and right sides of the geometric figure. For each $s \in S$, let s^L, s^R, s^T, and s^B represent the left, right, top, and bottom sides of s. For every s and $t \in S$, $(s^T, t^B) \in W$ is a straight-line from the top side of square s to the bottom side of square t. Similarly, (s^R, t^L) is a straight-line from the right side of square s to the left side of square t. Furthermore, a straight-line (s^T, bnd^B) is added to W from the top of each square s which lies on the top end of the geometric figure to bnd^B. Similarly, (s^R, bnd^L) (bnd^T, s^B) and (bnd^R, s^L) are added to W for each square s lying on the right, bottom, and left end of the geometric figure. Clearly, straight-lines in W are either in forms of (s^T, t^B), called vertical straight-line, or (s^R, t^L), called horizontal straight-line, for some s and t in either S or bnd.

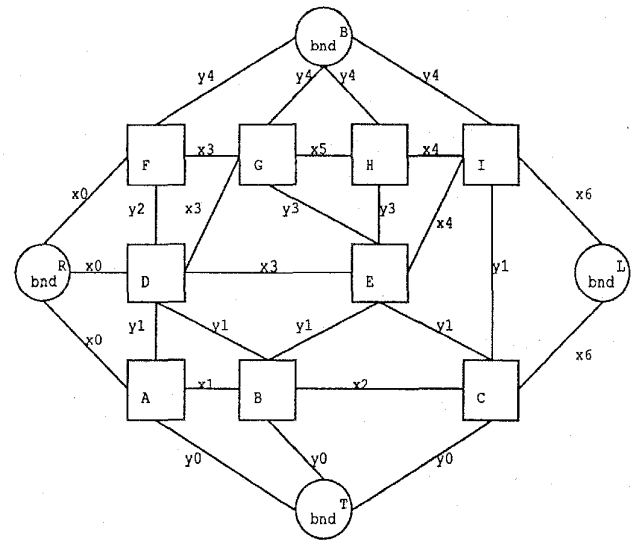


Fig. 6. The geometric figure for Fig. 5.

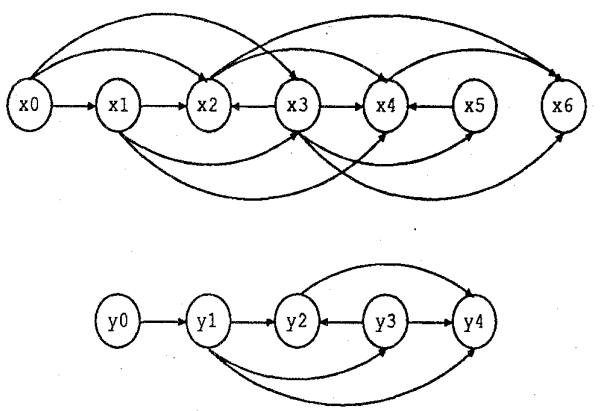


Fig. 7. The X-graph and Y-graph for Fig. 6.

To illustrate the procedure, we use the same partition as Fig. 5 in [2]. For a given partition in Fig. 5, its corresponding geometric figure is shown in Fig. 6. Observe that each square represents a cell and each straight-line represents an adjacency between cells. The four circles together represent the enclosing rectangle, bnd. It should be noted that we use bnd^B to represent the top of bnd and bnd^T, bnd^R, and bnd^L to represent the bottom, left, and right of bnd, respectively.

Once the geometric figure of the given partition is obtained, we can reconstruct the partition from the procedure below [2]. First we need to label the straight-lines in W. Each horizontal straight-line in W is assigned an X-variable and each vertical straight-line in W a Y-variable. Two horizontal

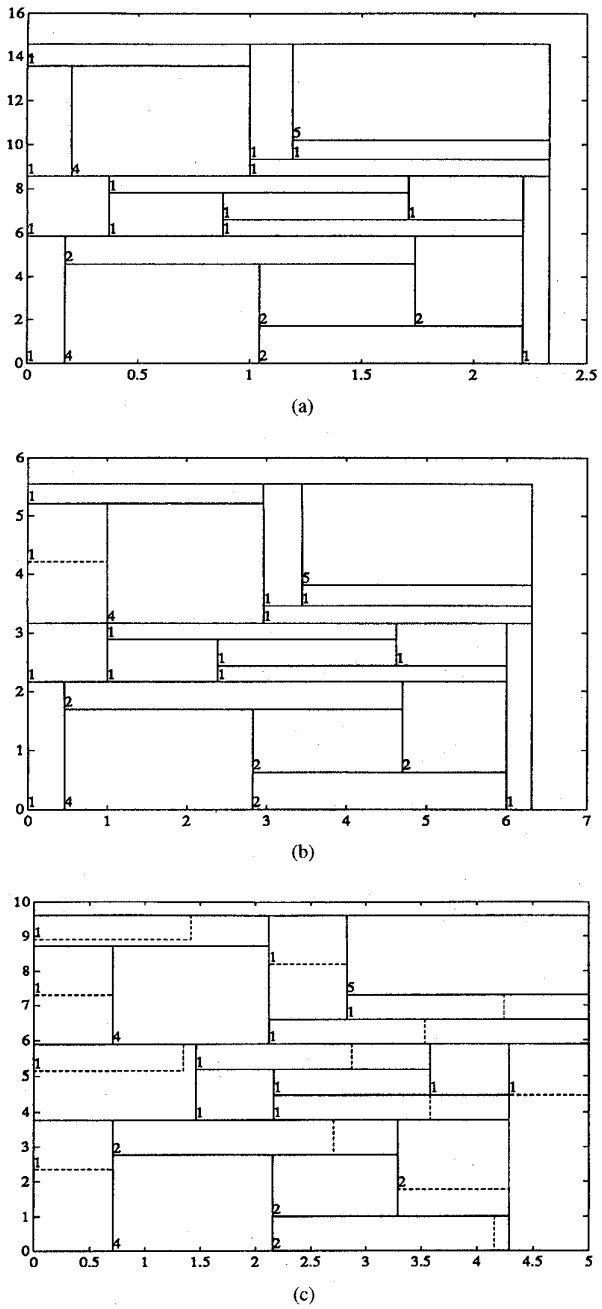


Fig. 8. Optimal floorplan for Example 1. (a) Solution without aspect ratio. (b) Slightly different partition. (c) Solution with aspect ratio.

straight-lines are assigned the same X-variable if and only if they are incident to the same side of some square. Similarly, two vertical straight-lines are assigned the same Y-variable if and only if they are incident to the same side of some square. In fact, when (s^R, t^L) is labeled x_i, the value of x_i represents the X coordinates of the right side of square s and the left side of square t. Similarly, when (s^T, t^B) is labeled y_i, the value of y_i can be treated as the Y coordinates of the top side of square s and the bottom side of square t. Since the geometric figure is planar, the number of straight-lines in W is $O(n)$ and the labeling can be done in $O(n)$ time. The labeling for the geometric figure in Fig. 5 is also shown.

We now construct two digraphs D^X and D^Y called X-graph and Y-graph, for the X-variables and the Y-variables, respectively, from the labeled geometric figure $GF[S, W]$ in the following manner. To construct the X-graph, we begin with the vertex set $V(D^X)$ consisting of all the X-variables on the horizontal straight-lines in W. An arc (x_i, x_j) is added to arc set $E(D^X)$ if and only if in the geometric figure $GF[S, W]$ one of the following two conditions occurs. 1) *Size Condition:* There are straight-lines (s^R, t^L) labeled x_i and (t^R, u^L) labeled x_j in W; i.e., there exists a square t where the horizontal straight-lines incident at its left side are labeled x_i and the horizontal straight-lines incident at its right side are labeled x_j. 2) *Contact Condition:* There are straight-lines (s^R, t^L) labeled x_i and (u^R, v^L) labeled x_j in W, and either (t^T, u^B) or (u^T, t^B) is also in W; i.e., there exist square t and u joined by a vertical straight-line where a horizontal straight-line with label x_i is incident at the left side of t and a horizontal straight-line with label x_j is incident at the right side of u. The reason for this is that the cells corresponding to t and u must touch each other along their horizontal sides. For this to happen, the coordinate of the left side of t must be smaller than the coordinate of the right side of u. Similarly, the coordinate of the left side of u must smaller than the coordinate of the right side of t. The Y-graph is constructed in a similar fashion. The X-graph and Y-graph for the GF in Fig. 5 is shown in Fig. 7.

Note that the time to construct the above two digraphs is of $O(n)$ time since the total possible number of the above conditions is at most $O(n)$. Therefore, the numbers of arcs in $E(D^X)$ and $E(D^Y)$ are both $O(n)$.

Once X-graph and Y-graph are obtained, it has been proven that the original partition can be constructed by using a slight variation of topological sort on the X-graph and Y-graph. The algorithm UNIFORM calculates the values associated with the vertices of digraph D in the following way:

```
procedure UNIFORM (D):

begin
    i < −0;
    while D is nonempty do
      begin
          let Z be the set of vertices
          in D with indegree 0;
          for all z in Z do value (z) < −i;
          D < −D − Z; (see below)
          i < −i + 1;
      end
end.
```

$D - Z$ is a digraph obtained by the deletion of a set of vertices Z from a digraph D; that is, $D - Z$ denotes a subdigraph with vertex set $V(D) - Z$ and whose arcs are those of $E(D)$ which are not incident at z, $z \in Z$. Applying UNIFORM(D^X) we obtain the X coordinates of the vertical sides of all cells and applying UNIFORM(D^Y) we obtain the Y coordinates of the horizontal sides of all cells.

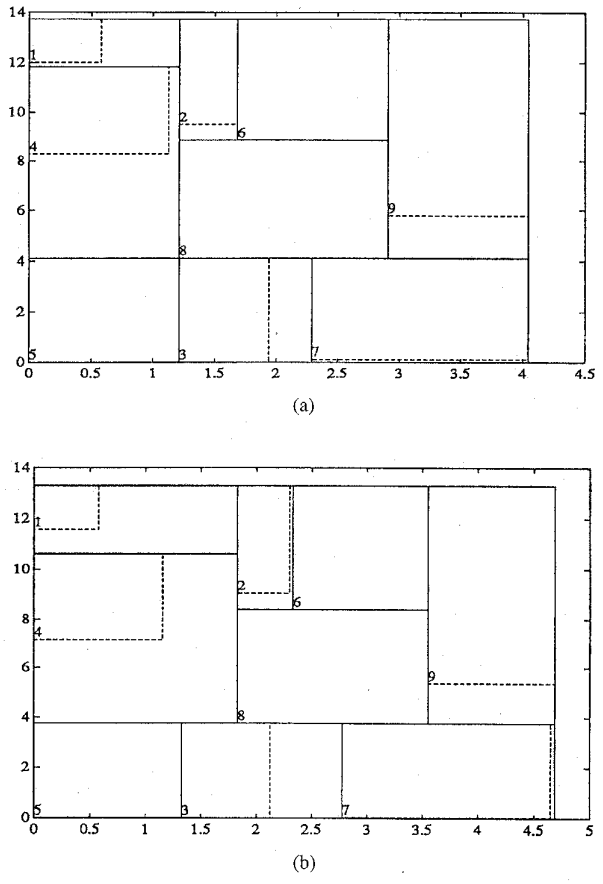


Fig. 9. Optimal floorplan for Example 2. (a) Solution without gap spec. (b) Solution with gap spec.

Since X-graph and Y-graph are sufficient to represent the original partition, we can obtain the partition constraints from them. Let x_i and x_j be the two nodes in X-graph with a connection from x_i to x_j, the partition constraint can then be written as

$$x_i \leq x_j. \tag{3.4}$$

To obtain a nonredundant set of partition constraints, we need to remove some connections from the graphs if necessary. For example, we have in Fig. 7

$$x_1 \leq x_3, x_3 \leq x_2 \quad \text{and} \quad x_1 \leq x_2. \tag{3.5}$$

Apparently, the third inequality is redundant. Once the nonredundant X-graph and Y-graph are obtained, it is clear that the unique set of partition constraints can be readily derived and has the form as before.

$$x_i \leq x_j \quad \text{or} \quad y_h \leq y_k. \tag{3.6}$$

IV. Numerical Testing

Since we have explained in Section III that the geometrical programming problem can be generated in $O(n)$ time with a feasible solution, we will discuss in this section if our approach is computationally feasible. Numerical examples are used to illustrate different aspects of our approach. Since we have access to the MINOS software in a HP9000-735 workstation, it is used to solve our convex optimization problem [5].

Given a partition and associated parameters a_i, m_i, M_i for each cell, the convex optimization problem and its corresponding FORTRAN program for MINOS are generated automatically. The optimal solution from MINOS is then displayed graphically. The exact procedure is given below.

Algorithm:

- *Data:* A partition in GF representation Cell parameters a_i, m_i, Mi
- *Output:* A globally optimal placement with minimum total area
- *Step 0:* Initialization: Input data;
- *Step 1:* Construct Connect-x which represents all the right-left (horizontal) connections;
- *Step 2:* Assign X-variable to each connection in Connect-x;
- *Step 3:* Construct the X-graph from Connect-x and remove redundancies;
- *Step 4:* Repeat steps 1–3 for the Y-graph;
- *Step 5:* Formulate the convex optimization problem;
- *Step 6:* Produce a feasible solution by using algorithm Uniform(D);
- *Step 7:* Generate the corresponding MINOS program MI00MAIN.f and its input file SPECS;
- *Step 8:* Run MINOS to obtain the optimal solution and display it graphically.

Example 1: To verify the correctness of our program, we first duplicate one result in [10]. Given the partition [10, Fig. 11(b)] without aspect ratio requirement, the optimal solution has zero wasted area as shown in Fig. 8(a). Note that Fig. 8(a) is the same as [10, Fig. 11(b)] where the number in each building block is the area specified.

However if we use the given partition [10, Fig. 11(a)] the optimal placement no longer has zero wasted area even without aspect ratio requirement. The result is given in Fig. 8(b). Note that the solid lines represent the partition and the dotted lines represent the building block, whose lower left corner has a number indicating its size. The only unoccupied area in this case is the region below the dotted line. If we further restrict that all building blocks have their aspect ratio in [0.5, 2], the result is in Fig. 8(c), where the unoccupied area is much larger.

Example 2: The given partition is the same as Fig. 5 in Section III. The parameters are arbitrarily given in Table I. The optimal placement is displayed in Fig. 9(a). At the first glance, it seems to have a different adjacency relationship from the given partition in the region near the left corner. A close examination indicates that the overlapping between two adjacent cells is zero. This is due to the adjacency relationship used in (3.6). To require the overlapping between two adjacent cells having a minimum contact, say gap = 0.5 units, Fig. 9(b) displays the optimal placement which does have the same partition as Fig. 5.

Example 3: In this example, twenty (20) different floorplans with various total number of cells are randomly generated. All the building blocks have $a_i = 1$, $m_i = 0.5$, and

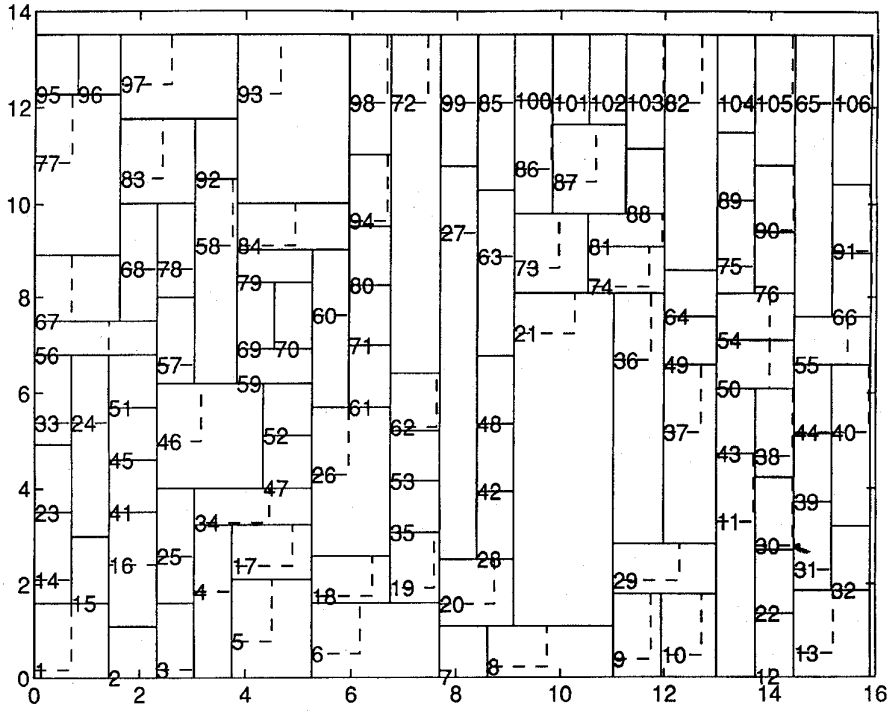


Fig. 10. Optimal floorplan for case 20 in Example 3.

TABLE I
SPECIFICATION FOR EXAMPLE 2

Cell	A	B	C	D	E	F	G	H	I
a_i	5	3	7	4	8	1	2	6	9
m_i	1	2	2	3	2	3	1	4	3
M_i	7	9	5	8	6	7	9	9	7

$M_i = 2$. The "time" command in UNIX is used to calculate the CPU time required for MINOS to solve the corresponding convex optimization problem, which equals the summation of user time and system time. The results are summarized in Table II.

The column "Cell" represents the total number of cells in the partition. The next three columns given the size of the convex problem. "Var" is the total number of variables. "Nonlin" and "Linear" are the total number of nonlinear and linear in equality constraints, respectively. The last two columns present the CPU time for two cases. In "Tinit," the feasible solution in Step 6 is used as the initial condition. In "Tnoinit," no initial condition is given to MINOS.

From Table II, we can observe that for large problems, the computation time is typically shorter when the initial feasible solution is provided. For those entries with an * under column Tnoinit, it means that the setup in MINOS for Tinit does not find the solution. To find the solution, we have to change the MINOS setup parameters. The highest computation time in column Tinit is less than 12 min for case 20. The results in Table II suggest that our approach is computationally feasible for problems with a reasonable size. Note that the computation time typically depends upon the complexity of the geometry of the floorplan, as seen in case 14 versus case 16. The floorplan for case 20 is shown in Fig. 10, where the cell number is printed in the lower left corner of the dotted lined building block. Note that we can not test any larger size examples due to the memory limitation in the setup of our workstation.

TABLE II
RESULTS FOR EXAMPLE 3

Case	Cell	Var	Nonlin	Linear	Tinit	Tnoinit
1	20	61	63	79	2.1	1.09
2	20	61	66	79	0.4	0.61
3	29	88	79	118	1.0	1.59
4	30	91	77	125	1.3	1.11
5	44	133	166	159	22.82	23.75
6	45	136	172	162	24.2	17.69
7	51	154	119	222	2.3	7.61
8	54	163	141	225	5.0	19.25
9	64	193	204	249	18.4	54.71
10	66	199	259	233	146.3	188.0
11	71	214	277	250	87.5	136.08
12	77	232	205	322	10.7	68.73
13	82	247	346	277	206.2	361.98
14	86	259	357	295	430.5	369.8*
15	92	277	361	319	161.8	442.23
16	94	283	216	405	11.9	55.62
17	101	304	375	366	174.6	508.57
18	102	307	391	357	232.8	575.25
19	105	316	449	352	513.9	925.4*
20	106	319	449	359	666.1	914.4*

V. CONCLUSION

A general nonsliced floorplanning problem has been discussed in this paper. By properly and carefully formulating the problem, an efficient procedure is derived which answers the question posted by Wimer *et al.*, in conclusion section of their paper [10]. That is, efficient solutions can indeed be

derived by taking advantage of the graph model. Moreover all the results can be extended to three-dimensional problems without any difficulties.

Based on the new formulation, the adjacency relationship among cells in the floorplan can be accommodated easily without sacrificing the run time performance of the procedure. The preservation of the adjacency relationship definitively increases the routability of the layout in later stage and improves the overall performance of the physical design. Note that we do not have to preserve any unnecessary adjacency relationship. This can be done just by removing the corresponding inequality from our problem formulation.

One remark is on the geometrical programming approach. Once a problem can be formulated as a geometrical programming, it is known to be solvable. Such an approach has been proven successful in many engineering applications. In fact, the result in [8] is very closely related to our result in the sense that it solves a subset of our floorplanning problems. Basically, it uses the geometrical programming to find the globally optimal floorplan subject to the height and width constraints in a given partition. However they can not deal with the adjacency relationship and the aspect ratio. On the other hand, we can easily add the height and width constraints if needed.

A final remark in on the conceptual difference between the problem formulation in [8] and ours. The formulation in [8] is performed from the cell's viewpoint. In our case, we separate the cell from its building block. By our way, it is easy to incorporate the aspect ratio since it is done on the building block directly. If we use the formulation in [8], it is impossible to accommodate the aspect ratio. To illustrate this point, let us consider a globally optimal floor plan where there is a very narrow cell with a small building block inside. Assume that the building block has the exact aspect ratio and thus the cell has a lot of wasted space. In this case, the aspect ratio for the cell may not be acceptable for the original aspect ratio spec. In other words, we really can not deal with the aspect ratio from the cell's viewpoint.

REFERENCES

[1] M. A. Breuer, Ed., "Design automation of digital systems," in *Theory and Techniques.* Englewood Cliffs, NJ: Prentice-Hall, 1972, vol. 1.

[2] S. L. Hakimi and T.-S. Moh, "Linear time optimization of partially solved floorplans," in *Proc. ISCAS,* 1990, pp. 327–331.

[3] T. C. Hu and E. S. Kuh, Eds., *VLSI Circuit Layout: Theory and Design.* New York: IEEE Press, 1985.

[4] G. P. McCormick, *Nonlinear Programming: Theory, Algorithms, and Applications.* New York: Wiley, 1983.

[5] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 user's guide," Standord Univ., Tech. Rep., SOL 83-20R, revised Jan. 1987.

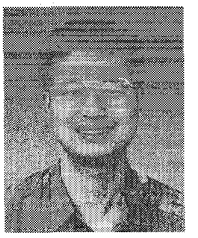
[6] T. Ohtsuki, Ed., *Layout Design and Verification.* The Netherlands: North-Holland, 1985.

[7] D. P. La Potin and S. W. Director, "Mason: A global floorplanning approach for VLSI design," *IEEE Trans. Computer-Aided Design,* vol. CAD-5, pp. 477–489, 1986.

[8] E. Rosenberg, "Optimal module sizing in VLSI floorplanning by nonlinear programming," *Methods Models Operations Res.,* vol. 33, no. 131–143, 1989.

[9] H. Watanabe and B. Ackland, "Flute: An expert floorplanner for VLSI," *IEEE Design Test Comput.,* vol. 4, pp. 32–41, 1987.

[10] S. Wimer, I. Koren, and J. Cederbaum, "Floorplans, planar graphs, and layouts," *IEEE Trans. Circuits Syst.,* vol. 35, pp. 267–278, Mar. 1988.



Teng-Sheng Moh (M'90) was born in Taipei, Taiwan, Republic of China. After graduating from the Department of Computer Science, National Taipei College of Business, Taipei, Taiwan, he received the B.S. degree in applied mathematics from Fu Jen Catholic University, Taipei, Taiwan. He also received the M.S.E. degree in information engineering from National Taiwan University, Taipei, Taiwan and the M.S. and Ph.D. degrees in computer science from University of California, Davis.

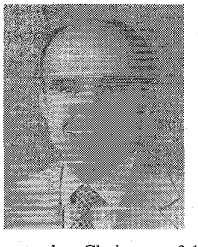
From 1984 to 1985, he was with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, as an Assistant Research Fellow working on the research and development of area-optimal layouts and logic minimization and PLA folding for VLSI. From 1986 to 1990, he was with the Department of Electrical and Computer Engineering, University of California, Davis, as a Teaching/Research Assistant working part time during the semesters and full time during the summers. From 1990 to 1994, he was with Cadence Design Systems, Inc., San Jose, CA, as a Senior Member of Technical Staff working on the research and development of layout compactor and symbolic checker. In 1994, he joined Silicon Valley Research, Inc., San Jose, as a Senior Technologist working on the research and development of physical design tools, where presently he is Vice President of Engineering. Besides IC CAD for physical design, his current research interests include parallel and distributed algorithms.

Dr. Moh is a member of ACM.

Tsu-Shuan Chang received the B.S. and M.S. degrees in electronic engineering from the National Chaio Tung University, Taiwan, Republic of China, in 1971 and 1973, respectively, and the M.S. and Ph.D. degrees in engineering from Harvard University, Cambridge, MA, in 1977 and 1981, respectively.

Since 1984, he has been with the University of California, Davis. He is currently an Associate Professor with the Department of Electrical and Computer Engineering. He was an Assistant Professor with the Department of Electrical Engineering, the State University of New York at Stony Brook from 1981 to 1984. His current interests are in the areas of Local and Global Optimization, Scheduling for IC Manufacturing Systems, VLSI Physical Design Automation, Controller Design, and Neural Networks.

Dr. Chang has been a member of Phi Tau Phi Scholastic Honor Society since 1971. He served as the registration chairman and publicity chairman for the IEEE Conference on Decision and Control in 1989 and 1992, respectively, and also served as an Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL from 1992 to 1994.



S. Louis Hakimi received the B.S. (Bronze Tablet), M.S., and Ph.D. degrees from the University of Illinois at Urbana, all in electrical engineering in 1955, 1957, and 1959, respectively.

He was an Assistant Professor of Electrical Engineering with the University of Illinois (Urbana) from February 1959 to August 1961. He joined Northwestern University as an Associate Professor of Electrical Engineering in September 1961 where he was promoted to Full Professor in September 1966. From September 1972 to September 1977, he served as Chairman of the Electrical Engineering Department at Northwestern. From September 1977 to August 1986, he was a Professor of electrical engineering and computer science, industrial engineering/management science, and applied mathematics at Northwestern University. In September 1986, he joined the University of California at Davis as Professor and Chairman of the Department of Electrical and Computer Engineering. His research interests lie in applications of graph theory and combinatorics to circuits, network theory, coding theory, operations research, and computer science.