GRAPH-BASED ALGORITHMS FOR FLOORPLANNING, ROUTING AND COMPACTION

BY

SAI-KEUNG DONG

B.S., University of Illinois at Urbana-Champaign, 1984
M.S., University of Minnesota, Minneapolis, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

ABSTRACT

In this thesis, four problems in floorplanning, routing and compaction are addressed. The first three problems involve the use of graphs in their solutions while the fourth problem is a generalized version of a graph-theoretic problem.

In floorplanning for flexible blocks, constraints on relative positions and separation requirements between the blocks are modeled by vertical and horizontal constraint graphs. An iterative improvement algorithm is used to obtain a feasible floorplan with minimal area. The algorithm determines the dimensions and the positions of the flexible blocks by adjusting the weights of vertices and computing the longest paths in the constraint graphs.

In routing for Quickly Customized Logic, one predefined metal layer and one custom metal layer are available for interconnection in a channel. A branch-and-bound algorithm that makes use of river routing and single-row routing algorithms is proposed. The algorithm uses simple criterion to limit the search space. It also maintains an overlap graph during the search process and backtracks if the overlap graph is not two-colorable.

In symbolic layout compaction, minimum and maximum spacing constraints among circuit elements can be expressed in terms of a system of linear inequalities. When all the inequalities are *graph-inequalities*, i.e. of the form $x_i - x_j \geq c$ where x_i, x_j are variables and c a constant, they can be represented by a weighted, directed constraint graph G. The presence of (directed) positive cycles in G means that there are no solutions to the corresponding system of inequalities. The problem of modifying the edge weights of G such that G (with new edge weights) has no positive cycles is studied. Polynomial-time algorithms that incur the minimum amount of change in edge weights are given.

Given a system of linear inequalities S, inequalities can be removed from and/or added to S to obtain a new system S' that has the same solution space as S. When all the inequalities in S are graph-inequalities, it is possible to find a smallest system of graph-inequalities that has the same solution space as S. This result can be generalized to the case where all the inequalities in S are *2VPI-inequalities*, i.e. of the form $ax_i + bx_j \geq c$ where a, b and c are constants. The generalized

result is that a smallest system of 2VPI-inequalities that has the same solution space as S can be found in polynomial time.

In memory of my beloved grandmother, Ven Tsing Chang.

ACKNOWLEDGEMENTS

I am deeply indebted to my advisor, Professor C. L. Liu, for the opportunity to pursue a Ph.D. degree. I appreciate very much for his patience and support. His wisdom and guidance have enlightened me significantly in research as well as my outlook in life. I am very fortunate to have such a scholarly advisor.

I am grateful to Professors Prithviraj Banerjee, Sung-Mo Kang, William Kubitz and Pravin Vaidya for serving on my doctoral committee. I thank them for their time and comments.

I am also very grateful to Professor Jane Liu for offering me a research assistantship in the last semester of my graduate study.

I thank Cheng Chang, Ki-Seok Chung, Tong Gao, Louis Mak, Anmol Mathur, Peichen Pan, Chaeryung Park, Yachyang and Hewijin Sun, Too-Seng Tia, Drs. David Binger, Kuang-Chien Chen, Jingsheng Cong, Taewhan Kim, David Knapp, Ran Libeskind-Hadas, Kee Yin Ng and Xiaojun Shen for their friendship and help they provide me during my graduate study at the University of Illinois.

I am very grateful to my parents, sister and uncle Henry for their love and understanding. I owe them a lot in spending so many years away from them to pursue my dream. Finally, I thank my dear wife Susan. Her love, support and companionship have made my journey to graduation a lot easier.

TABLE OF CONTENTS

Chapter

Chapter 1

Introduction

In Chapter 2, the problem of constrained floorplan design for flexible blocks is studied. In this problem, the dimensions of the flexible blocks and their locations are to be determined subject to constraints on the relative positions and separations between the blocks and the constraint on the aspect ratio of the bounding rectangle. The objective is to obtain a feasible floorplan with minimal area and small total wire length. We modeled the constraints on the relative positions and separations between the blocks in the horizontal and vertical directions by two acyclic, weighted, directed graphs, respectively. We present a two-phase, iterative improvement algorithm. In the first phase, the algorithm iteratively computes the dimensions of the blocks. The goals are to reduce the chip area and the total wire length. The computation is based on the length of the longest paths passing through a block and on an estimate of the length of wires straddling a block in the constraint graphs. In the second phase, the blocks are placed to satisfy the relative position and the separation requirements. If there are no overlaps among the blocks, the algorithm stops; otherwise additional constraints are introduced to resolve the overlap between one pair of blocks. The algorithm then goes back to the first phase to recompute the dimensions of all blocks. Such an iteration is repeated until a placement with no overlaps is obtained.

In Chapter 3, a channel routing problem in Quickly Customized Logic (QCL) is studied. In QCL technology, there is one predefined metal layer and one custom metal layer for routing. The predefined metal layer constains fixed-position wire segments and its mask is pre-designed. The custom metal layer is for customized routing and its mask will be custom-designed. Since only fixed-position vias are used in QCL technology, the two masks for vias and contact holes are also pre-designed. In comparison with conventional two-layer channel routing, the number of custom-designed masks is reduced from four to one. Experience showed that turn-around time was shortened to two to five days. We propose a new QCL channel model which generalizes the one in [SuDS91].

In the new model, a channel is divided into three regions: the upper, middle and the lower regions. River routing techniques are used in the upper and the lower regions while single-row routing techniques are used in the middle region. We give a branch-and-bound algorithm to enumerate the *ranges* of positions of terminals connecting wires crossing from the upper/lower region to the middle region. We show that the ranges can be computed in linear time. The branch-and-bound algorithm also maintains an overlap graph during the search process and backtracks if the overlap graph is not two-colorable.

In Chapter 4, the problem of constraint relaxation in graph-based compaction is studied. In symbolic layout compaction, minimum and maximum spacing constraints among circuit elements can be expressed in terms of a system of linear inequalities. When all the inequalities are *graph-inequalities*, i.e. of the form $x_i - x_j \geq c$ where x_i, x_j are variables and c a constant, they can be represented by a weighted, directed constraint graph G. The presence of (directed) positive cycles in G means that the positions of some circuit elements cannot be decided because of the existence of over-constraints. To eliminate such over-constraints, previous approaches examine positive cycles in G one at a time and apply heuristics to modify some of the edge weights. Such a local approach produces suboptimal results and takes exponential time in the worst case. We study the problem of modifying the edge weights of G such that G (with new edge weights) has no positive cycles; and such that the total change in edge weights and the length of the longest path from a ''source'' vertex to a ''sink'' vertex are kept to a minimum. We show that the problem can be solved in polynomial time by linear programming. Moreover, we show that a special case of the problem has a linear program whose dual corresponds to that of the minimum cost flow problem and hence can be solved efficiently.

In Chapter 5, the reduction problem of a system of *2VPI-inequalities* (linear inequalities with at most 2 Variables Per Inequality, i.e. inequalities of the form $ax_i + bx_j \geq c$ where x_i, x_j are variables and a, b and c are constants) is studied. Given a system of linear inequalities S, the reduction problem is to find a smallest system (i.e. a system with the least number of inequalities) S' such that S and S' have the same solution space. When there is no restriction on the form of the inequalities in S and S', it is known that the reduction problem can be solved in polynomial time.

When S and S' are systems of graph-inequalities, the reduction problem can be solved by pure graph-theoretic techniques in low-order polynomial time [PaDL93]. We study the case when S and S' are systems of 2VPI-inequalities. We show that for a given system of 2VPI-inequalities S, S' can be obtained in polynomial time via linear programming.

Chapter 2

Constrained Floorplan Design for Flexible Blocks

2.1. Introduction

In the floorplan design problem, a chip is divided into regions to accommodate a given set of flexible blocks with the objective that the overall chip area and/or the total connecting wire length be minimized. We consider a version of the problem in which the following assumptions are made:

(1) The blocks are flexible in that their dimensions have not been fixed and are to be selected from a number of possible candidates. The dimensions of a block can be either discrete variables or continuous variables. For example, if a block is to be implemented by standard cells, the number of rows used must be an integer. Thus, its dimensions can be chosen only from a finite number of candidates. On the other hand, the dimensions of some blocks can be continuous variables if they are to be implemented by full-custom designs.

(2) There may be constraints on the relative positions of the blocks. For example, in the design of a data path, a certain block must be placed above another block, or a certain block must be placed to the right of another block. These constraints are imposed either by the designer or by a given floorplan when our algorithm is used as a post-processor for other floorplan design algorithms.

(3) There may be minimum separation requirements between blocks in both the x-direction and the y-direction. These requirements are estimates of routing areas that should be reserved between blocks. (They are either held constant throughout the execution of the algorithm or varied as the algorithm adjusts the dimensions of the flexible blocks. We shall discuss this aspect of our algorithm in more details later on.)

(4) The overall aspect ratio of the chip is prespecified. In the hierarchical design methodology, one usually adopts a top-down approach to design a ciruit on a level by level basis. It is quite

possible that at a certain level, the overall aspect ratio of a chip (or a super block) has already been determined at the previous level. Consequently, one needs the capability of generating floorplans with a predetermined overall aspect ratio.

Various aspects of this problem have been studied in the literature [MaMH82, Ot83, St83, WoLi86, MoMT87, EsDK88, WiKC88, Ro89]. In [WoLi86], the method of simulated annealing was used to design slicing floorplans for flexible blocks. The algorithm in [WoLi86], however, cannot handle constraints on the relative positions of the blocks. In [St83], the problem of choosing optimal orientations for blocks that can be rotated by 90° for a given slicing floorplan was studied. In [WiKC88], a branch and bound approach incorporating Stockmeyer's ideas [St83] was used to determine the dimensions of the blocks for slicing as well as non-slicing floorplans in order to minimize the overall chip area. In [EsDK88], an algorithm for resizing flexible blocks based on the longest paths in the horizontal and vertical directions was discussed. In this approach, resizing was done after the initial placement and global routing phase. In [MoMT87] and [Ro89], the problem of determining the dimensions of flexible blocks with given constraints on their relative positions was studied. The former formulated the problem as a quadratic programming problem which was in turn approximated as a linear programming problem. (It should be noted, however, that the solution produced by the algorithm in [MoMT87] does not guarantee a floorplan with no overlaps among blocks.) The latter formulated the problem as a convex nonlinear programming problem with the assumptions that *all* the relative positions of the blocks were known (and hence no overlaps among blocks) and no routing space was explicitly reserved since it was included in the blocks.

We propose a two-phase iterative improvement algorithm. In the first phase, the algorithm iteratively compute the dimensions of the blocks. The goals are to reduce the chip area as well as the total wire length. The computation is based on the length of the longest paths passing through a block and on an estimate of the length of wires that would straddle the block in the constraint graphs. In the second phase, the algorithm place the blocks to satisfy the relative position and the separation requirements. If there are no overlaps among the blocks, the algorithm stops; otherwise additional constraints are introduced to resolve the overlap between one pair of blocks. Our algorithm then goes back to the first phase to recompute the dimensions of all blocks. Such an iteration is repeated until a

placement with no overlaps is obtained.

Our algorithm is quite versatile. It can handle a mixture of flexible blocks, some of which have continuous choices of dimensions and some of which have discrete choices of dimensions. In particular, the case of rotating a block by 90° can be captured by a flexible block with two choices of dimensions. (Although in this case we cannot guarantee the optimality of the resultant floorplan as in [St83], experimental results are close to optimal.) Our algorithm can also be used as a post-processor or compactor to improve (partial) floorplans produced by other algorithms or human designers. For example, one might want to design a floorplan using a bottom-up approach to place blocks that should be close to one another in clusters. In that case, one can first use clustering techniques to obtain an initial floorplan and then use our algorithm as a post-processor to obtain a better floorplan. In addition, since our algorithm produces solutions that are not necessarily slicing structures, it can be used to explore non-slicing structures for improving floorplans produced by other slicing floorplanners. Our algorithm runs very fast. It is about 25 times faster when compared with the simulated annealing based floorplanner in [WoLi86]. Thus, using our algorithm as a post-processor is an inexpensive way to obtain further improvements on floorplans produced by other floorplan design algorithms.

2.2. Problem Formulation

A *flexible block* is a rectangular block with adjustable height and width. For example, a flexible block might be one that has a fixed area, arbitrary height and width with its aspect ratio (height/width) lying within a given range (typically from r_1 to r_2 for given r_1 and r_2). On the other hand, a flexible block might be one that has only a finite number of allowable discrete combinations of height and width. We shall call these two types of blocks *continuous* blocks and *discrete* blocks, respectively. In general, the area and dimension requirements of a flexible block can be specified by a *shape curve* [WoLL88]. Figures 2.1a and 2.1b show a shape curve for a continuous block and a discrete block, respectively. As mentioned in the last section, a discrete block can be used to model a block that can be rotated by 90°. The shape curve in this case will consist of only two points. A discrete block can also be used to model a block that is rigid, i.e. the width and the height of the

block are fixed. In this case, the shape curve will consist of only one point.

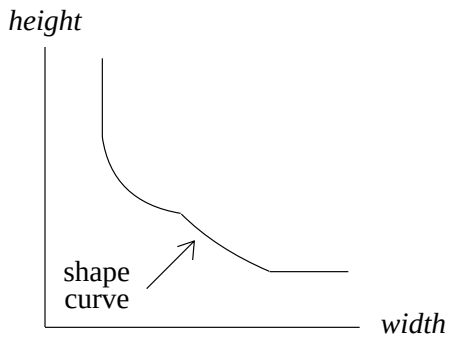

Fig. 2.1a Shape curve for a continuous block.

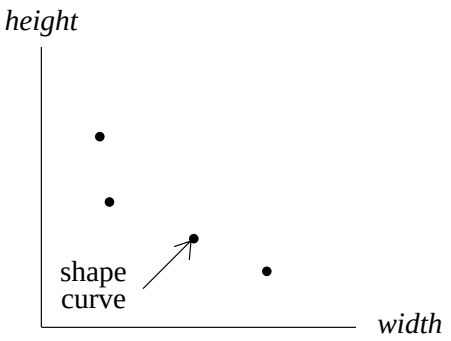

Fig. 2.1b Shape curve for a discrete block.

Constraints on the relative positions of the blocks as well as on the separations between the blocks are specified by two acyclic, weighted, directed graphs $G_H = (V_H, E_H)$ and $G_V = (V_V, E_V)$, which are referred to as the *horizontal* and *vertical constraint graph*, respectively. Let S denote the set of blocks to be placed. V_H, the vertex set of the graph G_H, consists of the blocks in S and two dummy blocks L and R. V_V, the vertex set of the graph G_V, consists of the blocks in S and two dummy blocks B and T. The dummy blocks L, R, B and T represent, respectively, the left, right, bottom and top boundary of a bounding rectangle [1]. The heights and widths of the dummy blocks are all zero. For blocks u and v in S, if v must be placed to the right of u, then there will be an edge (u,v) in E_H, the edge set of graph G_H. Similarly, if v must be placed above u, then there will be an edge (u,v) in E_V, the edge set of graph G_V. The weight of the edge (u,v) represents the minimum separation requirement between blocks u and v. To ensure that all blocks in S will be placed within the bounding rectangle, we have dummy edges (L,u), (u,R) in E_H; (B,u) and (u,T) in E_V for all u in S. Figures 2.2a and 2.2b show the constraint graphs G_H and G_V for a 6-block example. Dotted arrows denote dummy edges and solid arrows denote non-dummy edges. In the figures, the weights of all dummy edges are zero and are not shown; the weight of each solid edge is shown next to the edge.

[1] A bounding rectangle is the smallest rectangle enclosing all the blocks.

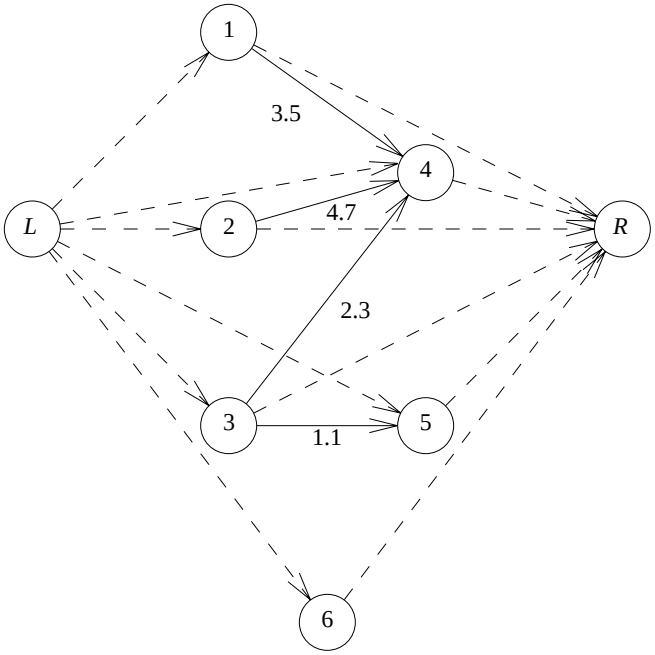

Fig. 2.2a G_H.

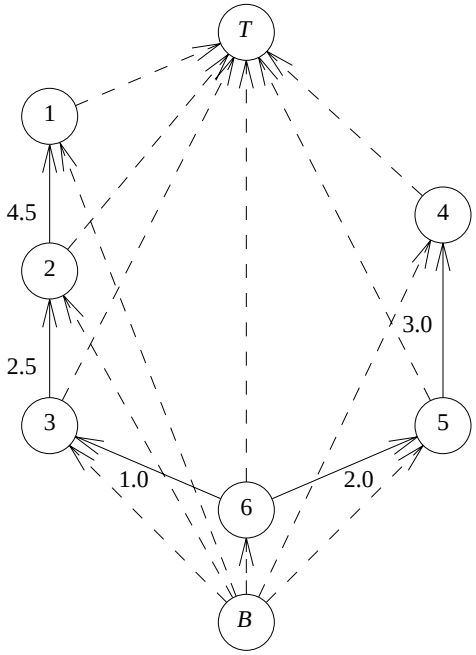

Fig. 2.2b G_V.

The horizontal and vertical constraint graphs are either given by the designer or derived from a given floorplan. In the latter case, edges in the constraint graphs are derived from the given floorplan in the manner described above, namely, for two blocks that are to the immediate left or right of each other, there is an edge between the corresponding vertices in the horizontal constraint graph; for two blocks that are immediately above or below each other, there is an edge between the corresponding vertices in the vertical constraint graph. Additional edges can be included to preserve certain relative positions among the blocks in a given floorplan. For example, Figure 2.3a is a floorplan consisting of 24 blocks. Figure 2.3b is the horizontal constraint graph derived from Figure 2.3a (dummy edges and the dummy vertices L and R are not shown in Figure 2.3b). In addition to edges obtained from the adjacency relationships among blocks, edges (11,3), (11,8), (11,17), (11,24), (16,3), (16,8) and (16,24) are inserted although block 11 is not adjacent to blocks 3, 8, 17 and 24 and block 16 is not adjacent to blocks 3, 8 and 24 in Figure 2.3a. These edges are included to preserve the vertical cut line (Figure 2.3a) that cuts through the entire bounding rectangle.

The number of (non-dummy) edges in the constraint graphs varies for different design situations. For example, when our algorithm is used for initial floorplan design, the designer might

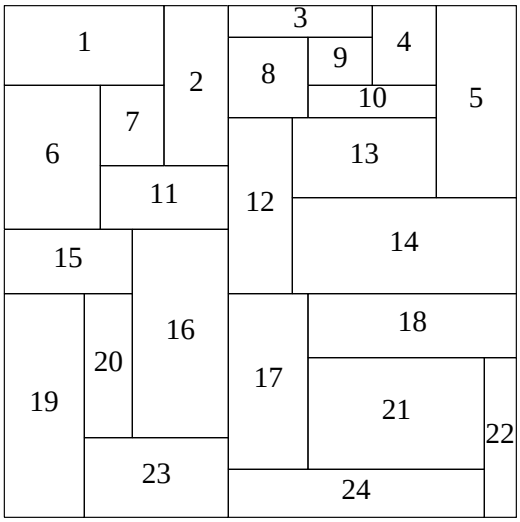

Fig. 2.3a.

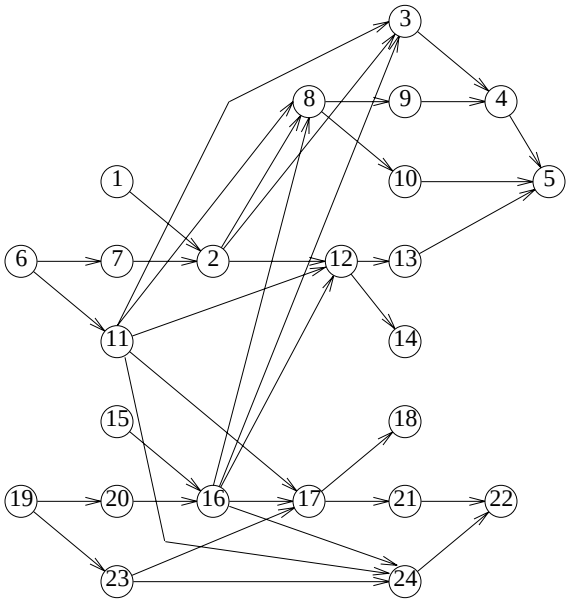

Fig. 2.3b G_H derived from Fig. 2.3a.

want to specify the relative ordering of the blocks in a data path. In this case, the constraint graphs might be relatively sparse. On the other hand, when our algorithm is used to modify an existing floorplan, we might want to use dense constraint graphs to keep most of the relative positions of the blocks intact while the positions and dimensions of a few of the blocks are changed. As will be seen below, our algorithm adjusts the dimensions of the flexible blocks according to the lengths of paths passing through the blocks in the horizontal and vertical constraint graphs. Consequently, for a highly constrained floorplan (which corresponds to dense contraint graphs), our algorithm will produce very good choices of the dimensions of the blocks. On the other hand, when the horizontal and vertical constraint graphs are sparse, our algorithm will have more freedom to choose the relative positions of the blocks. Since determination of the relative positions of the blocks and determination of the dimensions of the blocks are interrelated, it is more difficult to assure the overall quality of the results because we are attempting to optimize in a less constrained system.

Our floorplan design problem can now be stated formally as follows : given n flexible blocks (specified by their shape curves), constraint graphs G_H, G_V, an overall aspect ratio ρ and the net list, determine the dimensions and the absolute positions of the blocks so that (i) there are no overlaps

between any two blocks, (*ii*) constraints on relative block positions and separations specified in G_H and G_V are satisfied, (*iii*) the bounding rectangle has an aspect ratio that is close to ρ, (*iv*) the bounding rectangle has small area and (*v*) the total wire length is small.

2.3. The Algorithm

Our algorithm is divided into two phases. In the first phase, an iterative procedure is employed to determine the dimensions of the blocks. Intuitively, because the choice of the dimensions of one block affects the choices of the dimensions of other blocks as well as the total wire length, an iterative computation is an effective way to carry out a step-by-step adjustment of the solution. We introduce the concept of *scaling factors* for a block which are used to compute the dimensions of the blocks in each iteration. The scaling factors for a block are functions of the length of the longest path going through the block and the number of nets that straddle the block in the horizontal and vertical constraint graphs. (A net (which is a set of blocks) is said to *straddle* a block B_i in a constraint graph G if *both* the set of predecessors and the set of successors of B_i in G contain some blocks that belong to the net.) In attempting to minimize the lengths of the longest paths in the horizontal and vertical constraint graphs, our algorithm, in effect, tries to minimize the chip width and height which in turn reduces the chip area. In addition, by considering the number of nets that straddle a block, the dimensions of the block are adjusted to reduce the total wire length. In the second phase, blocks are placed to satisfy the separation requirements. However, since the constraint graphs might be relatively sparse, a placement of blocks that satisfies all the constraints in the constraint graphs might yield a floorplan with overlapping blocks. Instead of simply shifting the blocks to avoid overlaps, our algorithm separates two overlapping blocks and recomputes the dimensions of all blocks. This is accomplished by introducing an additional edge to one of the constraint graphs to separate two overlapping blocks and then returning to the first phase again. The advantage of returning to the first phase is that the blocks can be resized so that the aspect ratio of the bounding rectangle can be close to ρ. The details are given in Section 2.3.2. Figure 2.4 shows the steps in our algorithm. (After the dimensions of the blocks are computed, one might want to dynamically adjust the separation requirements between the blocks to reflect possible changes in

routing area requirements. This optional step is shown as a dashed box in Figure 2.4. The separation requirements can be computed as functions of the dimensions of the blocks and routing space estimated by global routing. For example, if the width of a block X is increased and the height decreased, we might want to increase the corresponding separation requirements (weights of edges incident with X) in G_V and decrease the separation requirements in G_H since there will be more connection pins along the top and bottom edge of the block and fewer connection pins along the left and right edge of the block.)

2.3.1. Computation of Block Dimensions

The first phase of our algorithm is based on the idea of iterative improvement. The initial dimensions of the blocks can either be prespecified or be randomly selected.

Let $PRED_H(B_i)$ and $SUCC_H(B_i)$ be the set of predecessors and successors of block B_i in G_H. The set of nets that straddle B_i in G_H is defined to be

$$STRADDLE_H(B_i) = \{\ N\ :\ N \text{ is a net, } N \cap PRED_H(B_i) \neq \varnothing \text{ and } N \cap SUCC_H(B_i) \neq \varnothing\ \}.$$

($PRED_V(B_i)$, $SUCC_V(B_i)$ and $STRADDLE_V(B_i)$ are defined analogously.)

In each iteration, we compute for each flexible block B_i two scaling factors s_x^i and s_y^i which are used to determine how the dimensions of B_i should be changed. The scaling factors s_x^i and s_y^i are computed according to the formulae :

$$s_x^i = \rho \cdot l_H(B_i) + \lambda \cdot |\ STRADDLE_H(B_i)|$$

$$s_y^i = \quad l_V(B_i) + \lambda \cdot |\ STRADDLE_V(B_i)|$$

where $l_H(B_i)$ is the length of the longest path from L to R through B_i in G_H, $l_V(B_i)$ is the length of the longest path from B to T through B_i in G_V, λ is a constant specified by the user and $|\ |$ denotes the cardinality of a set. By the length of a path in G_H (respectively G_V), we mean the sum of the edge weights along the path and the widths (respectively heights) of the blocks in the path. For example, in Figure 2.2a, let the widths of blocks 1, 2, 3, 4, 5 and 6 be 10, and recall that the width of

blocks L and R are 0 and the weights of all dummy edges are 0. The length of the longest path in G_H is 24.7 (achieved by path $L \rightarrow 2 \rightarrow 4 \rightarrow R$), whereas the length of the longest path through block 3 is 22.3 (achieved by path $L \rightarrow 3 \rightarrow 4 \rightarrow R$) and the length of the longest path through block 6 is 10 (achieved by path $L \rightarrow 6 \rightarrow R$).

The aspect ratio for the bounding rectangle, ρ, is used in the computation of s_x^j so that the overall aspect ratio will converge towards ρ in the final solution.

Since G_H and G_V are directed acyclic graphs, $PRED_H(B_i)$, $SUCC_H(B_i)$, $PRED_V(B_i)$ and $SUCC_V(B_i)$ $(1 \le i \le n)$ can be computed in $O(n^3)$ time by following the topological order in G_H and G_V. Once the sets of predecessors and successors are computed, they can be used to compute $STRADDLE_H(B_i)$ and $STRADDLE_V(B_i)$ in $O(n^2 \cdot p)$ time (p is the number of nets). This can be done by computing for each block the number of nets that straddle it. Since $STRADDLE_H(B_i)$ and $STRADDLE_V(B_i)$ are not changed in the first phase, they only need to be computed once each time the first phase is entered. On the other hand, the longest path lengths $l_H(B_i)$ and $l_V(B_i)$ $(1 \le i \le n)$ are changed in every iteration of the first phase. They can be computed in $O(n+e)$ time ($e = \max\{|E_H|, |E_V|\}$) by traversing G_H (respectively G_V) in topological order [ReND77]. Thus, after pre-computing $STRADDLE_H(B_i)$ and $STRADDLE_V(B_i)$, the scaling factors s_x^j and s_y^j can be computed in $O(n + e)$ time.

In computing the dimensions of a block, continuous blocks are handled in a slightly different way from that for discrete blocks. For practical considerations, a continuous block B_i is assumed to have a lower and an upper limit on its width. Let $[\alpha_i, \beta_i]$ be the allowable range of width for block B_i, i.e., the width of B_i must be chosen to be larger than or equal to α_i and be smaller than or equal to β_i. Let $\delta_i = c \cdot (\beta_i - \alpha_i)$ where c is a positive real number specified by the user to control how large δ_i should be. δ_i is the maximum change in width for B_i in one iteration. Let

$$f_i = (s_y^j - s_x^j) / (s_x^j + s_y^j) \qquad (1)$$

To determine the dimensions of B_i for the next iteration, let w_i be the current width. Then the new width w_i' is computed as follows :

$$w_i' = \begin{cases} \alpha_i & \text{if } w_i + f_i \cdot \delta_i < \alpha_i \\ \beta_i & \text{if } w_i + f_i \cdot \delta_i > \beta_i \\ w_i + f_i \cdot \delta_i & \text{otherwise} \end{cases} \quad (2)$$

The new height of B_i, h_i', can then be determined from the shape curve. (In most cases, the shape curve for a continuous block is a hyperbola and h_i' is computed as (area of B_i)/w_i'.)

In the case of a discrete block, we cannot use (2) to compute its new width since it has only a finite number of choices for its dimensions. Let the finite number of choices for B_i be $(a_1^i, b_1^i), \cdots, (a_{j_i}^i, b_{j_i}^i)$ where $a_1^i < \cdots < a_{j_i}^i$ are the possible widths and $b_1^i, \cdots, b_{j_i}^i$ are the corresponding heights. Let the current width of block B_i be a_p^i. Then the new width w_i' is $a_{p'}^i$, where

$$p' = \begin{cases} p-1 & \text{if } f_i < -\varepsilon \text{ and } p > 1 \\ p+1 & \text{if } f_i > \varepsilon \text{ and } p < j_i \\ p & \text{otherwise} \end{cases} \quad (3)$$

f_i is computed according to (1) and ε is a real constant defined by the user. The new height h_i' is $b_{p'}^i$. In other words, if $f_i > \varepsilon$ ($< \varepsilon$), the new width is the next available point on the shape curve that is to the right (left) of the current width. We summarize the above discussion in the following pseudo code.

Algorithm The First Phase of Constrained Floorplan;

find a topological labeling for G_H;
find a topological labeling for G_V;
compute $PRED_H(B_i)$ and $SUCC_H(B_i)$ $(1 \le i \le n)$ according to topological order in G_H;
compute $PRED_V(B_i)$ and $SUCC_V(B_i)$ $(1 \le i \le n)$ according to topological order in G_V;
compute $STRADDLE_H(B_i)$ and $STRADDLE_V(B_i)$ $(1 \le i \le n)$;

for each continuous block B_j **do**
    $\delta_j \leftarrow c \cdot (\beta_j - \alpha_j)$;

for $k \leftarrow 1$ to *num_of_iteration* (* user-specified *) **do**
    compute $l_H(B_i)$ $(1 \le i \le n)$ according to the topological order in G_H;
    compute $l_V(B_i)$ $(1 \le i \le n)$ according to the topological order in G_V;
    for each flexible block B_i **do**
        $s_x^i \leftarrow \rho \cdot l_H(B_i) + \lambda \cdot |\ STRADDLE_H(B_i)|$;
        $s_y^i \leftarrow l_V(B_i) + \lambda \cdot |\ STRADDLE_V(B_i)|$;
        $f_i \leftarrow (s_y^i - s_x^i)/(s_x^i + s_y^i)$;

if B_i is a continuous block **then**
    use (2) to compute new w_i;
else ($*$ B_i is a discrete block $*$)
    use (3) to compute new w_i;
look up new h_i from the shape curve of B_i;

2.3.2. Placement and Removal of Overlaps

After the computation of the dimensions of the blocks in the first phase, we shall place the blocks according to the constraints specified in the constraint graphs. If there are no overlaps in the placement, our algorithm stops. Otherwise, our algorithm will try to resolve the overlaps (one pair of overlapping blocks at a time) by introducing an additional edge in one of the constraint graphs. We then go back to the first phase to recompute the dimensions of each block.

We place the blocks in the first quadrant of the x-y plane by placing the lower left corner of block B_i at (x_i, y_i) where x_i is the length of the longest path from L to B_i in G_H and y_i is the length of the longest path from B to B_i in G_V. It is possible that the interior of some blocks B_i and B_j intersects. In this case, we try to remove the overlap by inserting *one* edge between B_i and B_j in one of the two constraint graphs, G_H and G_V. There are four candidate edges (all of weight zero) that can be inserted into the constraint graphs, namely, edges (B_i, B_j) and (B_j, B_i) in G_H; and edges (B_i, B_j) and (B_j, B_i) in G_V. The insertion of one of these four edges into the corresponding constraint graph is sufficient to remove the overlap between B_i and B_j. Thus, if there are k pairs of overlapping blocks, k edges might be selected from the $4k$ candidate edges for insertion into the constraint graphs. Instead of trying to resolve the conflicts among all the k pairs of overlapping blocks at the same time, we select one edge out of the $4k$ candidate edges to remove the overlap between one pair of blocks. This allows more flexibility when we go back to the first phase to recompute the dimensions of the blocks.

Before we discuss how to select one edge out of the $4k$ candidate edges, we consider the two candidate edges (B_i, B_j) and (B_j, B_i) in G_H for the pair of overlapping blocks B_i and B_j. (The two candidate edges in G_V are handled similarly.) In Figure 2.5a, B_i and B_j are two overlapping blocks; P_i (P_j) denotes a longest path in G_H that starts from the left dummy block L and ends at B_i (B_j);

and Q_i (Q_j) denotes a longest path in G_H that starts from B_i (B_j) and ends at the right dummy block R. If the edge (B_i, B_j) is inserted into G_H, block B_j is shifted to its right and Figure 2.5b is obtained. A longest path that passes through the edge (B_i, B_j) is $<P_i, B_i \rightarrow B_j, Q_j>$. Let h_{ij} be the length of this path. Similarly, if the edge (B_j, B_i) is inserted into G_H, block B_i is shifted to its right and Figure 2.5c is obtained. A longest path that passes through the edge (B_j, B_i) is $<P_j, B_j \rightarrow B_i, Q_i>$. Let h_{ji} be the length of this path. In Figure 2.5b, every block on path Q_j is shifted to the right. Similarly, in Figure 2.5c, every block on path Q_i is shifted to the right. In a similar fashion, insertion of the two edges (B_i, B_j) and (B_j, B_i) in G_V also produces two paths of lengths v_{ij} and v_{ji} that pass through the corresponding edge. Intuitively, one wants to select an edge such that its insertion into the corresponding constraint graph produces the least amount of shifting of the blocks. We select the edge that gives the smallest value in $\{\rho \cdot h_{ij}, \rho \cdot h_{ji}, v_{ij}, v_{ji}\}$. We weigh h_{ij} and h_{ji} by ρ, the overall aspect ratio, because we want the two longest path lengths in G_H and G_V to be close to the aspect ratio ρ eventually.

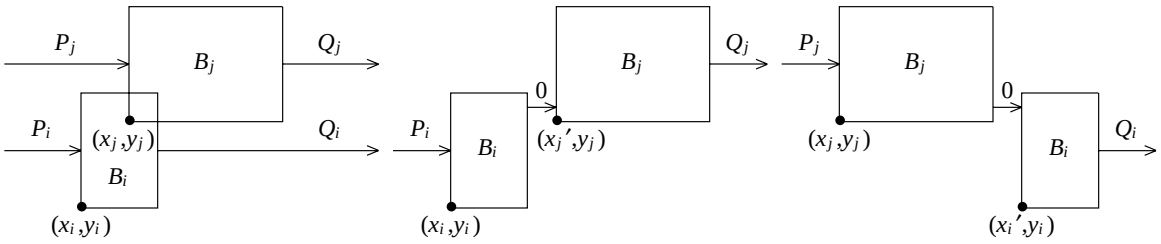

Fig. 2.5a. **Fig. 2.5b**. **Fig. 2.5c**.

For the k pairs of overlapping blocks, there are $4k$ numbers, $\{\rho \cdot h_{ij}, \rho \cdot h_{ji}, v_{ij}, v_{ji} \mid B_i$ intersects $B_j\}$ [2]. We select the edge (and hence the pair of intersecting blocks) which has the smallest corresponding value. In the case of a tie, we select the edge such that its two incident vertices have small degrees. This tends to distribute the edges in the contraint graphs evenly.

To detect block overlaps, one can use a naive $O(n^2)$ algorithm to check every pair of blocks. For better performance, there is an $O(n \log n + k)$ algorithm for reporting k intersecting pairs of a set of n rectangles [Ed83].

---

[2] Terms like $STRADDLE_H(B_i)$ and $STRADDLE_V(B_i)$ can also be used in the edge selection process.

Overlap detection can either be carried out after every iteration or after every m iterations in the first phase for some user-specified constant m. We adopt the latter approach because the $O(n^2)$ overlap detection procedure could be expensive. Furthermore, by allowing overlaps during intermediate iterations, there is more freedom for the blocks to change their dimensions and hence a better chance in finding a floorplan with the minimum overall area. Thus, the first phase takes $O(n^3 + n^2 \cdot p + m \cdot (n+e))$ time each time it is executed where e is the number of edges in the constraint graphs, n is the number of blocks and p is the number of nets. The second phase takes $O(n^2)$ time each time it is executed.

It should be noted that our algorithm also provides the designer with the capability of creating routing channels after an initial placement was obtained. We can change the edge weights or introduce additional constraints (in the constraint graphs) to ensure minimum channel widths between "rows" or "columns" of blocks and recompute the dimensions of the blocks by reentering the first phase of the algorithm.

Finally, if the total area of the blocks is small compared with the reserved routing area (as specified by the separation requirements between blocks), then changing the block dimensions during the first phase of the algorithm will have little effect on the area of the bounding rectangle. This is *not* a weakness of our algorithm but an intrinsic nature of such design examples. For the second phase, adding an edge to remove one overlapping pair of blocks might introduce new overlaps. However, this does not happen frequently in practical situations because the routing space reserved is enough to prevent overlaps. Moreover, the worst case that can happen is to insert an edge between any two blocks for a total of $O(n^2)$ edges, i.e. both the first phase and the second phase of our algorithm are executed at most $O(n^2)$ times.

2.4. Experimental Results

We give six examples to illustrate the quality of the solutions produced by our algorithm. For the first five examples, we concentrate on the effects of path lengths on the block dimensions and the overall area. Thus λ was set to 0 in the formulae for the scaling factors. In the sixth example, we set λ to seven different values to test how the terms $STRADDLE_H(B_i)$ and $STRADDLE_V(B_i)$ in the

scaling factors affect the total wire length. In the first five examples, the overlap removal phase of our algorithm was never executed because there were no overlaps in the solutions obtained after the execution of the first phase. The sixth example, however, went through the second phase four times when λ was set to 0. The flexible blocks in the first three and the last examples are continuous blocks while those in the fourth and the fifth examples are discrete blocks.

The first example is a simple 6-block example. The constraint graphs G_H and G_V are shown in Figures 2.6a and 2.6b with all edge weights equal to zero. The initial configuration is shown in Figure 2.6c. The configurations after 10, 20 and 50 iterations are shown in Figures 2.6d, 2.6e and 2.6f, respectively. Observe the reduction in the area of the bounding rectangle. Finally, after 260 iterations, we obtained the configuration shown in Figure 2.6g. The aspect ratio of the bounding rectangle in Figure 2.6g is 1.65. For comparison, we reran the example starting with the initial configuration in Figure 2.6c and changing the specification on the overall aspect ratio to 1. We obtained Figure 2.6h after 330 iterations which also has the minimum area.

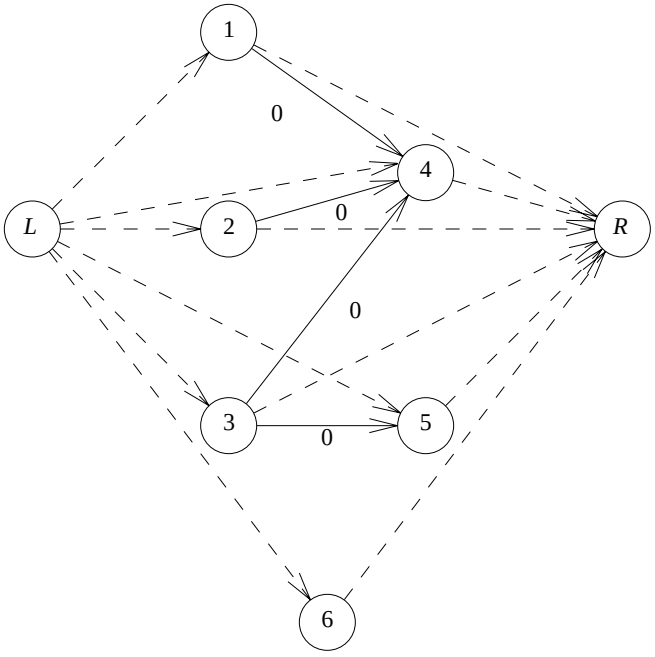

Fig. 2.6a G_H.

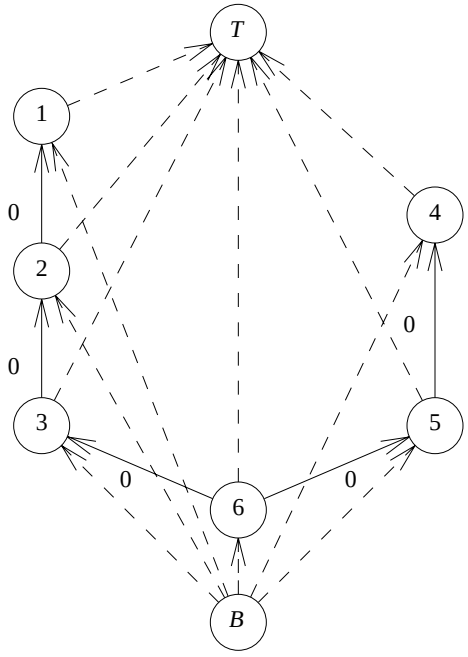

Fig. 2.6b G_V.

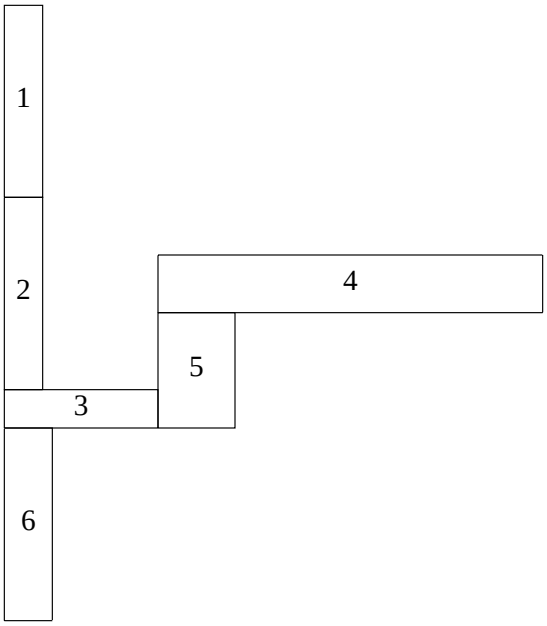

Fig. 2.6c Initial Configuration.

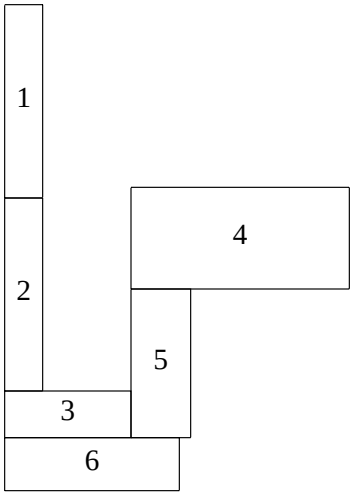

Fig. 2.6d After 10 iterations.

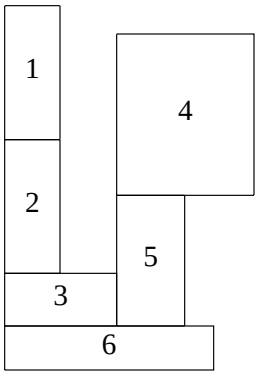

Fig. 2.6e After 20 iterations.

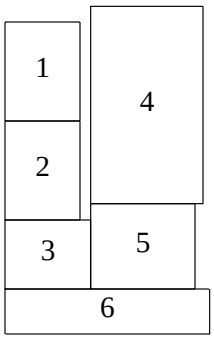

Fig. 2.6f After 50 iterations.

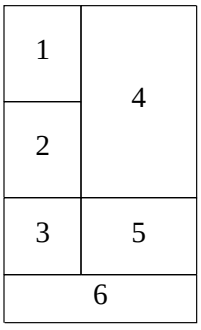

Fig. 2.6g $\rho = 1.65$, 260 iterations.

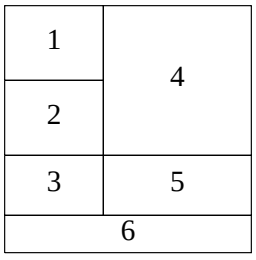

Fig. 2.6h $\rho = 1$, 330 iterations.

The second example is an 18-block example. The initial configuration is shown in Figure 2.7a which is obtained from the output of a simulated annealing floorplanner [WoLi86]. We used our algorithm as a post-processor. We preserved its aspect ratio (1.28) as well as the relative positions of the blocks and obtained an improvement of 6% in the area of the bounding rectangle (see Figure 2.7b). The initial configuration has 18% dead space and the final configuration only has 12% dead space.

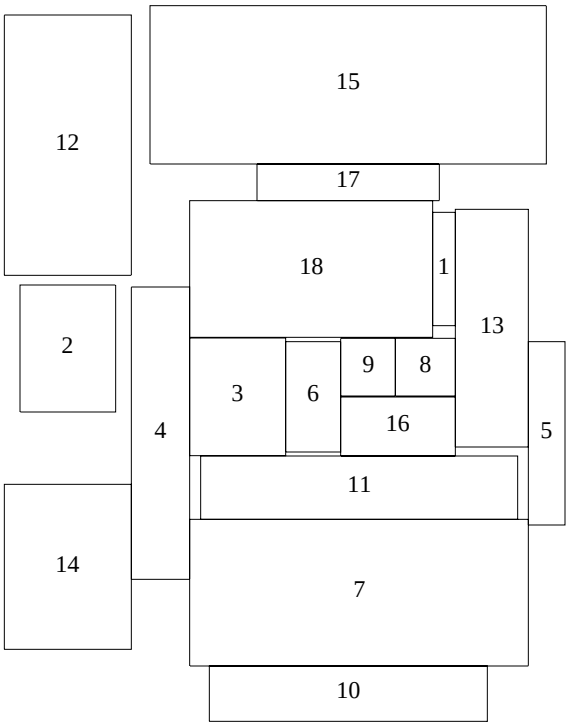

Fig. 2.7a.

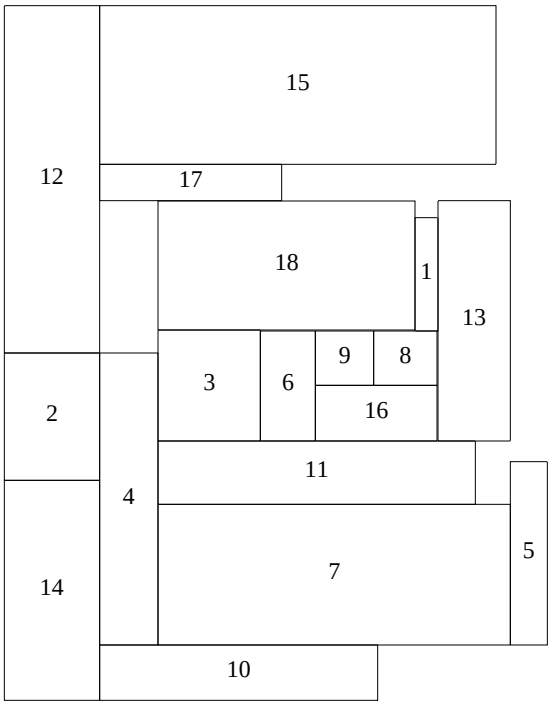

Fig. 2.7b.

The third example is a 41-block example. The dimensions of the blocks and the constraints on their relative positions and separation requirements came from an approximation of an example in [MoMT87]. The initial configuration is shown in Figure 2.8a. We preserved the relative positions of the blocks. After 250 iterations, we obtained the configuration shown in Figure 2.8b. The area of the bounding rectangle in Figure 2.8b is 14.3% smaller than that in Figure 2.8a.

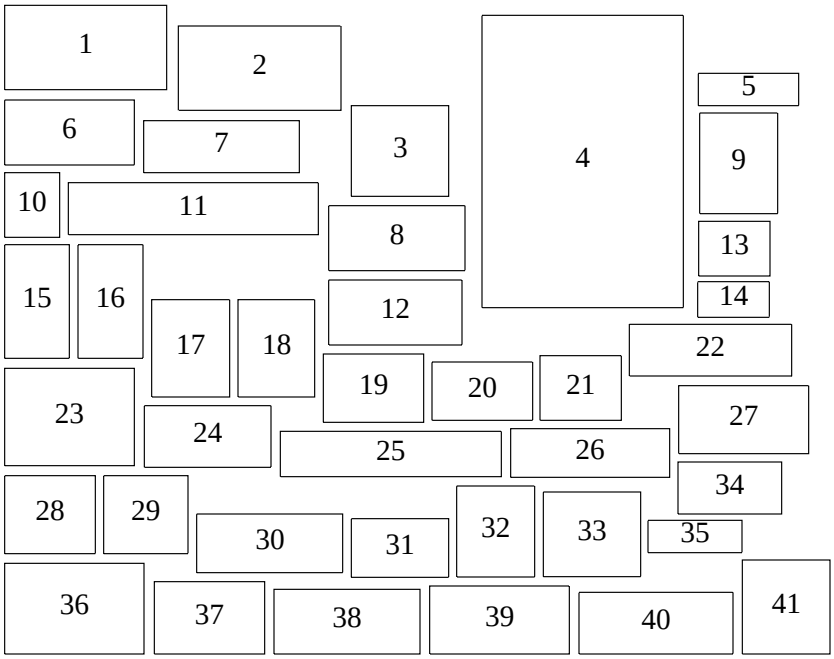

Fig. 2.8a.

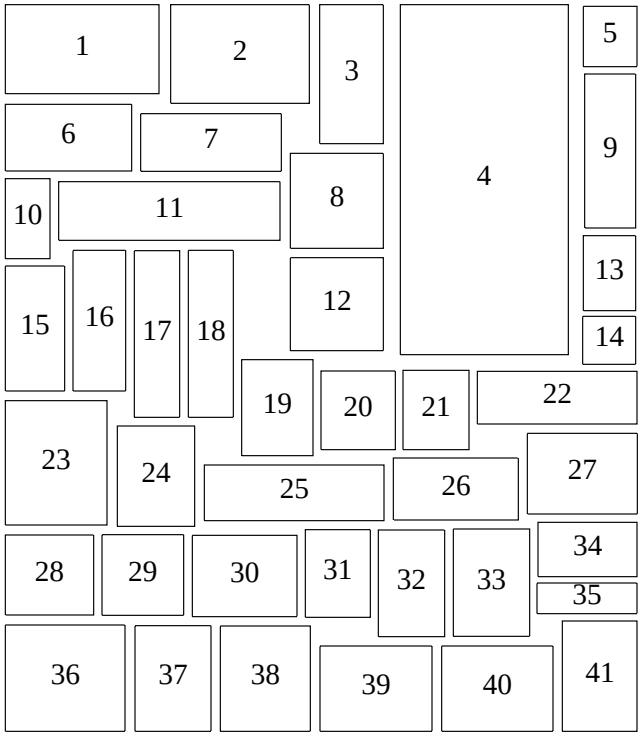

Fig. 2.8b.

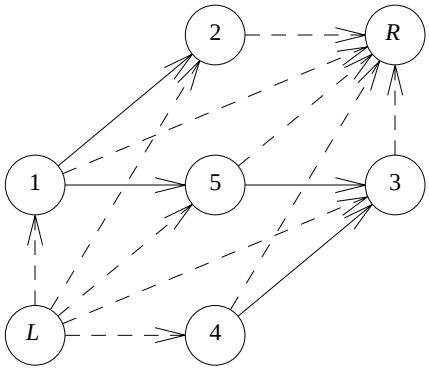

Fig. 2.9a G_H.

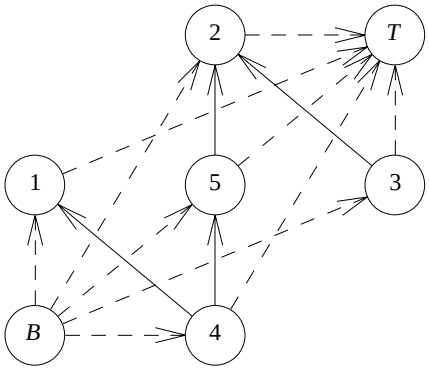

Fig. 2.9b G_V.

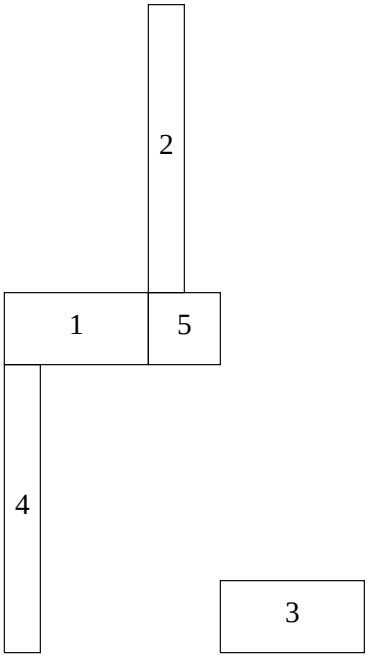

Fig. 2.9c.

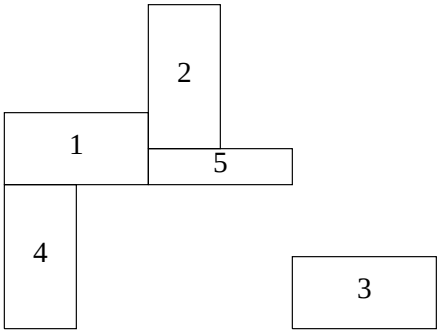

Fig. 2.9d.

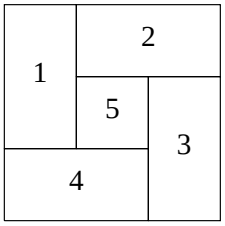

Fig. 2.9e.

The fourth example consists of five discrete blocks. Blocks 1, 2, 3 and 4 all have four choices for their dimensions. The choices are : (0.5,4), (1,2), (2,1) and (4,0.5) where the first (second) component in an ordered pair gives the width (height) of a block. Block 5 has three choices for its dimensions. The choices are (0.5,2), (1,1) and (2,0.5). The constraint graphs G_H and G_V are shown in Figures 2.9a and 2.9b, respectively. Figure 2.9c shows a random initial configuration of the five blocks. We specified the overall aspect ratio to be 1. The configurations after the first and the second

iteration are shown in Figures 2.9d and 2.9e, respectively. Note that the final solution in Figure 2.9e is a non-slicing floorplan. It is interesting to observe that when only discrete blocks are involved, the first phase in general takes less number of iterations than that for continuous blocks.

The fifth example is a 24-block example taken from [WiKC88]. All the blocks are discrete blocks and their choices of dimensions are exactly the same as those in [WiKC88]. (The number of choices of dimensions for a block ranges from 3 to 8.) The optimal solution is shown in Figure 2.10c. We derived the constraint graphs from Figure 2.10c but used random choices of widths and heights for the block dimensions. Figures 2.10a and 2.10b show the horizontal and vertical constraint graph derived from Figure 2.10c. In both constraint graphs, dummy edges and dummy vertices L, R, B and T are not shown. Figure 2.10d shows the initial configuration. After 6 iterations, the configuration shown in Figure 2.10e was obtained. Finally, after 10 iterations, we obtained the configuration shown in Figure 2.10f. Although the solution in Figure 2.10f is not optimal, it is only 5.5% larger than that of Figure 2.10c.

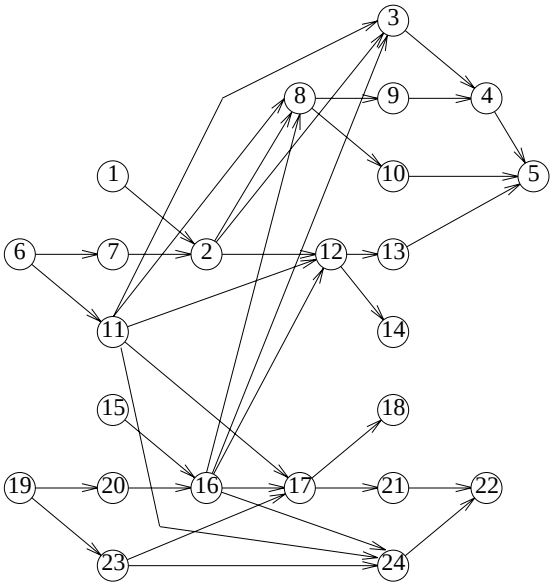

Fig. 2.10a G_H derived from Fig. 2.10c.

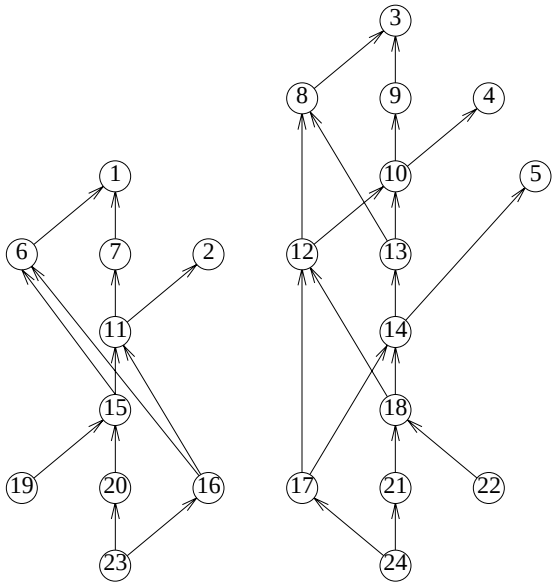

Fig. 2.10b G_V derived from Fig. 2.10c.

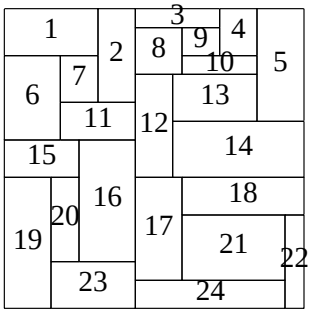

Fig. 2.10c.

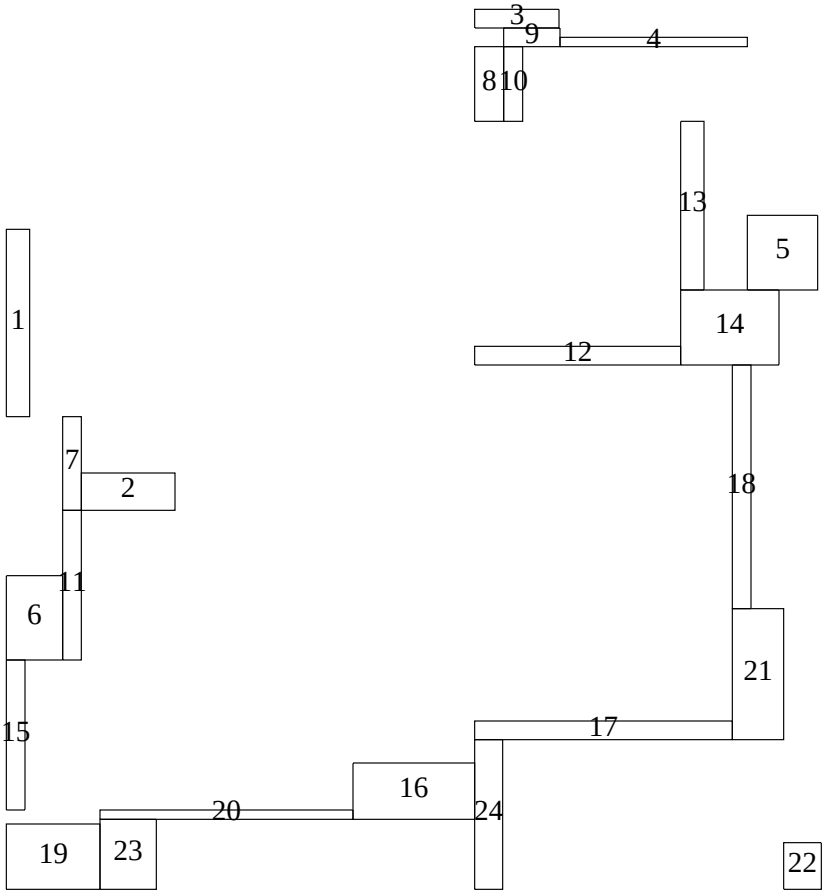

Fig. 2.10d.

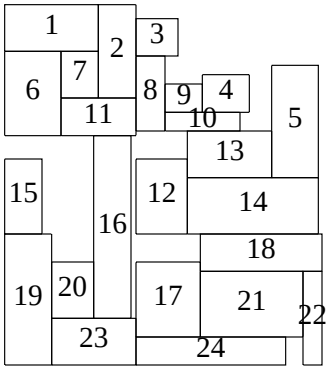

Fig. 2.10e.

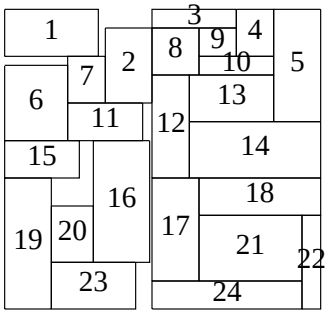

Fig. 2.10f.

The sixth example is an 18-block example. All edge weights in the constraint graphs are zero. The initial configuration is shown in Figure 2.11a. We specified the overall aspect ratio to be 0.84, λ to be 0 and the overlap detection algorithm (the second phase) was performed after every 300 iterations in the first phase. Figure 2.11b shows the configuration after 300 iterations. Overlaps were

detected between blocks 4 and 17 and blocks 12 and 13. The edge (4,17) was chosen and inserted into the horizontal constraint graph. After 300 more iterations, the configuration is shown in Figure 2.11c. The edge (13,12) was inserted into the vertical constraint graph. After another 300 iterations, we reached Figure 2.11d. The edge (7,4) was inserted into the vertical constraint graph. Figure 2.11e was obtained after another execution of the first phase. The edge (1,5) was then inserted into the vertical constraint graph. After 300 more iterations, we obtained what is shown in Figure 2.11f which has 3.6% dead space [3] and 0.84 as the overall aspect ratio. We also tried different values of λ to test its effects on the overall area and the total wire length. (Assuming that every net originates from the center of a block to which it is connected, we compute the length of a wire as the half perimeter of the bounding rectangle of the net.) The results are summarized in Table 2.1 (the overall area and the total wire length corresponding to $\lambda = 0$ are treated as 100 %). Note that as the value of λ increases, the overall area increases while the total wire length decreases. For $\lambda = 0.6$, the edge (13,12) was inserted into the vertical constraint graph and the edge (4,17) was inserted into the horizontal constraint graph. The floorplan obtained is shown in Figure 2.11g which is 1.1% larger than the one shown in Figure 2.11f but the total wire length in Figure 2.11g is 5.7% less than that in Figure 2.11f. As an illustration, one of the nets in this example is {7, 11, 15}. In Figure 2.11g, block 4 is to the left of block 7. This allows block 11 to be closer to block 15. As a result, the wire length of the net {7, 11, 15} in Figure 2.11g is less than that in Figure 2.11f.

λ	Overall area	Total wire length
0	100 %	100 %
0.1	100 %	99.4 %
0.2	100.2 %	98.9 %
0.3	100.6 %	98.5 %
0.4	100.5 %	97.7 %
0.5	100.7 %	97.9 %
0.6	101.1 %	94.3 %

Table 2.1

We implemented our algorithm in C on a Multimax computer. All the examples ran to completion in less than one minute of computation time.

[3] The presence of dead space in Figure 2.11f in spite of a slicing floorplan and flexible blocks is due to the limitations imposed by the shape curves of some of the blocks, i.e. some blocks cannot be stretched anymore.

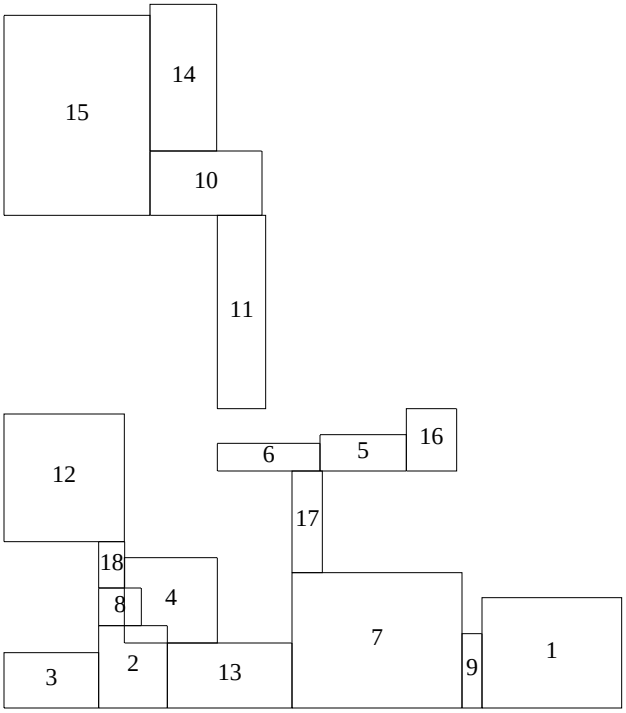

Fig. 2.11a.

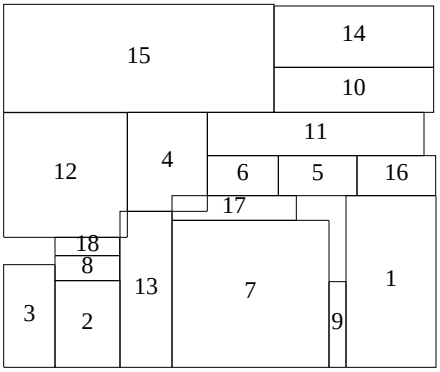

Fig. 2.11b.

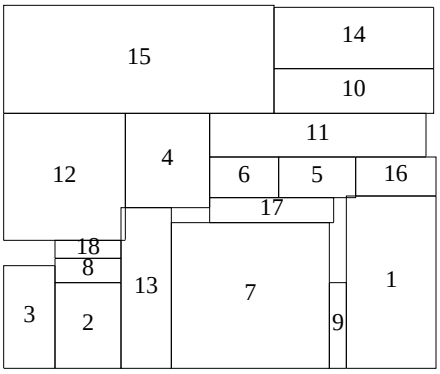

Fig. 2.11c.

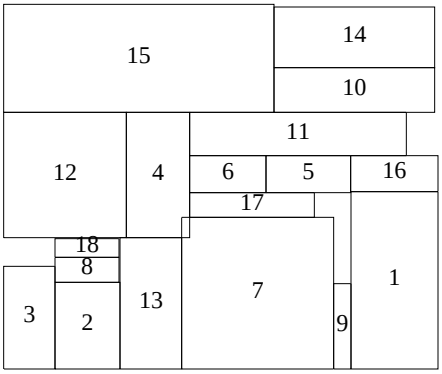

Fig. 2.11d.

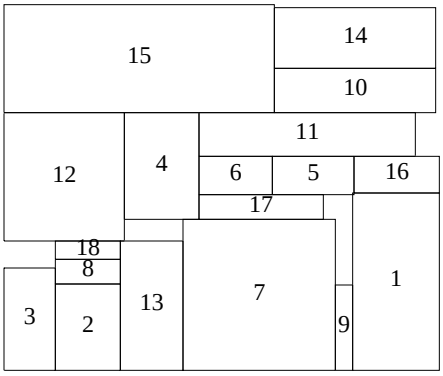

Fig. 2.11e.

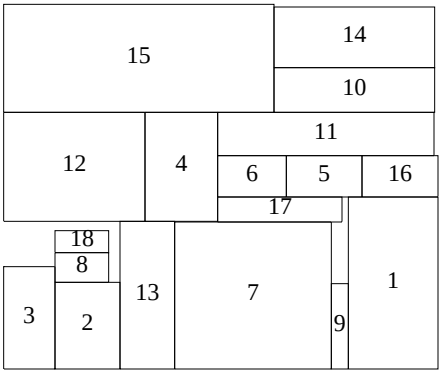

Fig. 2.11f.

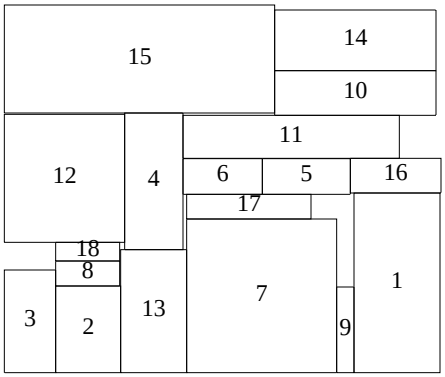

Fig. 2.11g.

2.5. Summary

We propose a simple and fast iterative improvement algorithm for solving the constrained floorplan design problem. The algorithm allows users to specify an aspect ratio for the bounding rectangle and constraints on the relative positions and separation requirements of blocks. In the first phase of the algorithm, two scaling factors for each flexible blocks are computed for adjusting the block dimensions iteratively. In the second phase, blocks are placed according to the constraint graphs. If no overlaps are detected, the algorithm stops; otherwise an edge is inserted into one of the constraint graphs to resolve the overlap between one pair of blocks. The algorithm then goes back to the first phase. Experimental results show that our algorithm tends to achieve the prespecified overall aspect ratio and produce floorplans with small overall area.

References

[AhGU72] A. V. Aho, M. R. Garey and J. D. Ullman, The Transitive Reduction of a Directed Graph, *SIAM Journal of Computing*, Vol. 1, No. 2, pp. 131-137, June 1972.

[BaVa92] C. S. Bamji and R. Varadarajan, Hierarchical Pitchmatching Compaction Using Minimum Design, *Proceedings of 29th Design Automation Conference*, pp. 311-317, June 1992.

[BaVa93] C. S. Bamji and R. Varadarajan, MSTC: A Method for Identifying Overconstraints during Hierarchical Compaction, *Proceedings of 30th Design Automation Conference*, pp. 389-394, June 1993.

[BoHS92] E. Boros, P. L. Hammer and R. Shamir, A Polynomial Algorithm for Balancing Acyclic Data Flow Graphs, *IEEE Transactions on Computers*, Vol. 41, No. 11, pp. 1380-1385, Nov. 1992.

[BoMu76] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, New York, N.Y., 1976.

[Boye88] D. G. Boyer, Symbolic Layout Compaction Review, *Proceedings of 25th Design Automation Conference*, pp. 383-389, June 1988.

[CaWo90] Y. Cai and D. F. Wong, An Optimal Channel Pin Assignment Algorithm, *Digest of Technical Papers, International Conference on Computer-Aided Design*, pp. 10-13, Nov. 1990.

[Cohe91] E. Cohen, Combinatorial Algorithms for Optimization Problems, PhD thesis, Department of Computer Science, Stanford University, June 1991.

[CoLi90] J. Cong and C. L. Liu, Over-the-Cell Channel Routing, *IEEE Transactions on CAD*, Vol. 9, No. 4, pp. 408-418, April 1990.

[CoLR90] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, New York, 1990.

[CoMe91] E. Cohen and N. Megiddo, Improved Algorithms for Linear Inequalities with Two Variables per Inequality, *Proceedings of 23rd Annual ACM Symposium on Theory of Computing*, pp. 145-155, May 1991.

[CoWL88] J. Cong, D. F. Wong and C. L. Liu, A New Approach to Three- or Four-Layer Channel Routing, *IEEE Transactions on CAD*, Vol. 7, No. 10, pp. 1094-1104, Oct. 1988.

[Edel83] H. Edelsbrunner, A New Approach to Rectangle Intersections, Part I, *Int. J. Computer Mathematics*, Vol. 13, pp. 209-219, 1983.

[EsDK88] B. Eschermann, W. Dai, E. S. Kuh and M. Pedram, Hierarchical Placement for Macrocells : A "Meet in the Middle" Approach, *Digest of Technical Papers, International Conference on Computer-Aided Design*, pp. 460-463, Nov. 1988.

[GaJo79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman and Company, New York, 1979.

[GaVL91] T. Gao, P. M. Vaidya and C. L. Liu, A New Performance Driven Placement Algorithm, *Digest of Technical Papers, International Conference on Computer-Aided Design*, pp. 44-47, Nov. 1991.

[Golu80] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, N.Y., 1980.

[HuHB91] X. Hu, R. G. Harber and S. C. Bass, Minimizing the Number of Delay Buffers in the Synchronization of Pipelined Systems, *Proceedings of 28th Design Automation Conference*, pp. 758-763, June 1991.

[King84] C. Kingsley, A Hierarchical, Error-Tolerant Compactor, *Proceedings of 21st Design Automation Conference*, pp. 126-132, June 1984.

[Leng82] T. Lengauer, The Complexity of Compacting Hierarchically Specified Layouts of Integrated Circuits, *23rd Annual Symposium on Foundations of Computer Science*, pp. 358-368, Nov. 1982.

[Leng84] T. Lengauer, On the Solution of Inequality Systems Relevant to IC-layout, *Journal of Algorithms*, Vol. 5, pp. 408-421, 1984.

[LeSa88] C. E. Leiserson and J. B. Saxe, A Mixed-integer Linear Programming Problem which is Efficiently Solvable, *Journal of Algorithms*, Vol. 9, pp. 114-128, 1988.

[LeTa92] J. F. Lee and D. T. Tang, HIMALAYAS - A Hierarchical Compaction System with a Minimized Constraint Set, *Digest of Technical Papers, International Conference on Computer-Aided Design*, pp. 150-157, Nov. 1992.

[LiCS93] A. Lim, S. W. Cheng and S. Sahni, Optimal Joining of Compacted Cells, *IEEE Transactions on Computers*, Vol. 42, No. 5, pp. 597-607, May 1993.

[LiWo83] Y. Z. Liao and C. K. Wong, An Algorithm to Compact a VLSI Symbolic Layout with

Mixed Constraints, *IEEE Transactions on CAD*, Vol. CAD-2, No. 2, pp. 62-69, April 1983.

[MaMH82] K. Maling, S. H. Mueller and W. Heller, On Finding Most Optimal Rectangular Package Plans, *Proceedings of 19th Design Automation Conference*, pp. 663-670, June 1982.

[Marp90] D. Marple, A Hierarchy Preserving Hierarchical Compactor, *Proceedings of 27th Design Automation Conference*, pp. 375-381, June 1990.

[Megi83] N. Megiddo, Towards a Genuinely Polynomial Algorithm for Linear Programming, *SIAM Journal of Computing*, Vol. 12, No. 2, pp. 347-353, May 1983.

[MoMT87] M. Mogaki, C. Miura and H. Terai, Algorithm for Block Placement with Size Optimization Technique by the Linear Programming Approach, *Digest of Technical Papers, International Conference on Computer-aided Design*, pp. 80-83, November 1987.

[NaES89] K. Nakamura, Y. Enomoto, Y. Suehiro and K. Yamashita, Advanced CMOS ASIC Design Methodologies, *Regional Conference on Microelectronics and Systems*, 1989.

[Otte83] R. Otten, Efficient Floorplan Optimization, *Proceedings of International Conference on Computer Designs*, pp. 499-502, 1983.

[PaDL93] P. Pan, S. K. Dong and C. L. Liu, Optimal Graph Constraint Reduction for Symbolic Layout Compaction, *Proceedings of 30th Design Automation Conference*, pp. 401-406, June 1993.

[PaSt82] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization : Algorithms and Complexity*, Prentice-Hall, N. J., 1982.

[PrKu89] S. Prasitjutrakul and W. J. Kubitz, Path-delay Constrained Floorplanning : A Mathematical Programming Approach for Initial Placement, *Proceedings of 26th Design Automation Conference*, pp. 364-369, June 1989.

[RaSa83] R. Raghavan and S. Sahni, Single Row Routing, *IEEE Transactions on Computers*, Vol. C-32, No. 3, pp. 209-220, March 1983.

[RaSa84] R. Raghavan and S. Sahni, The Complexity of Single Row Routing, *IEEE Transactions on Circuits and Systems*, Vol. CAS-31, No. 5, pp. 462-472, May 1984.

[ReND77] E. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Inc., Englewood Cliffs, 1977.

[ReWo86] M. Reichelt and W. Wolf, An Improved Cell Model for Hierarchical Constraint-graph

Compaction, *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pp. 482-485, Nov. 1986.

[Rose89] E. Rosenberg, Optimal Module Sizing in VLSI Floorplanning by Nonlinear Programming, *Methods and Models of Operations Research*, Vol. 33, pp. 131-143, 1989.

[Schi88] W. L. Schiele, Compaction with Incremental Over-Constraint Resolution, *Proceedings of 25th Design Automation Conference*, pp. 390-395, June 1988.

[SeSc91] M. Servit and J. Schmidt, Strategy of One and Half Layer Routing, *Microprocessing and Microprogramming*, Vol. 32, No. 1-5, pp. 417-424, Aug. 1991.

[SoCh85] J. N. Song and Y. K. Chen, An Algorithm for One and Half Layer Channel Routing, *Proceedings of 22nd Design Automation Conference*, pp. 131-136, June 1985.

[Stoc83] L. Stockmeyer, Optimal Orientations of Cells in Slicing Floorplan Designs, *Information and Control*, Vol. 57, pp. 91-101, 1983.

[SuDS91] Y. Sun, S. K. Dong, S. Sato and C. L. Liu, A Channel Router for Single Layer Customization Technology, *Digest of Technical Papers, International Conference on Computer-Aided Design*, pp. 436-439, Nov. 1991.

[Tarj83] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.

[TuTe91] T. C. Tuan and K. H. Teo, On River Routing with Minimum Number of Jogs, *IEEE Transactions on CAD*, Vol. 10, No. 2, pp. 270-273, Feb. 1991.

[Ullm84] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, Md., 1984.

[WaWo90] T. C. Wang and D. F. Wong, An Optimal Algorithm for Floorplan Area Optimization, *Proceedings of 27th Design Automation Conference*, pp. 180-186, June 1990.

[WiKC88] S. Wimer, I. Koren and I. Cederbaum, Optimal Aspect Ratios of Building Blocks in VLSI, *Proceedings of 25th Design Automation*, pp. 66-72, June 1988.

[WoDF89] D. Wong, G. De Micheli and M. Flynn, Inserting Active Delay Elements to Achieve Wave Pipelining, *Digest of Technical Papers, International Conference on Computer-Aided Design*, pp. 270-273, Nov. 1989.

[WoLi86] D. F. Wong and C. L. Liu, New Algorithm for Floorplan Design, *Proceedings of 23rd Design Automation Conference*, pp. 101-107, June 1986.

[WoLL88] D. F. Wong, H. W. Leong and C. L. Liu, *Solution of Design Automation Problems by the Method of Simulated Annealing*, Kluwer Academic Publishers, Chapter 3, 1988.

[YaCD93] S. Z. Yao, C. K. Cheng, D. Dutt, S. Nahar and C. Y. Lo, Cell-based Hierarchical Pitchmatching Compaction Using Minimal LP, *Proceedings of 30th Design Automation Conference*, pp. 395-400, June 1993.

[YaES89] K. Yamashita, Y. Enomoto, T. Sasaki, S. Kawahara, A. Kumagai, T. Sendo and A. Okada, A Quick Turnaround Time ASIC 'QCL series', *Institute of Electronics, Information and Communication Engineers Technical Report*, Vol. 89, No. 93, pp. 65-69, June 1989. (In Japanese)

[YoKu82] T. Yoshimura and E. S. Kuh, Efficient Algorithms for Channel Routing, *IEEE Transactions on CAD*, Vol. CAD-1, pp. 25-35, Jan. 1982.

[ZhDa92] Q. Zhu and W. M. Dai, Perfect-balance Planar Clock Routing with Minimal Path-length, *Digest of Technical Papers, IEEE International Conference on Computer-Aided Design*, pp. 473-476, Nov. 1992.

Vita

Sai-keung Dong was born on June 23, 1961 in Hong Kong. In 1984, he received his B.S. degree *magna cum laude* in Mathematics and Computer Science from the University of Illinois at Urbana-Champaign. In 1986, he received his M.S. degree in Computer Science with minor in Mathematics from the University of Minnesota, Minneapolis. He is currently with Quickturn Design Systems, Inc., Mountain View, California.