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 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.

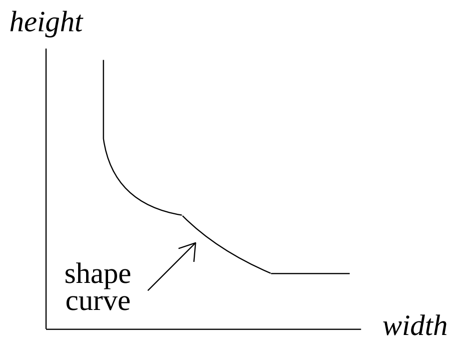


Fig. 2.1a Shape curve for a continuous block.

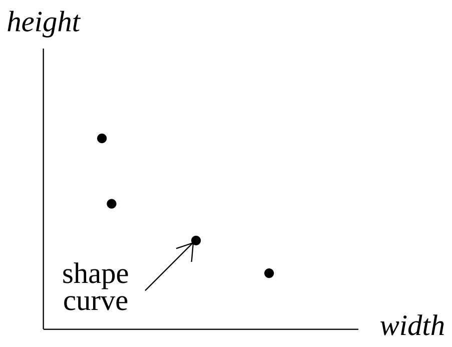


Fig. 2.1b Shape curve for a discrete block.

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.

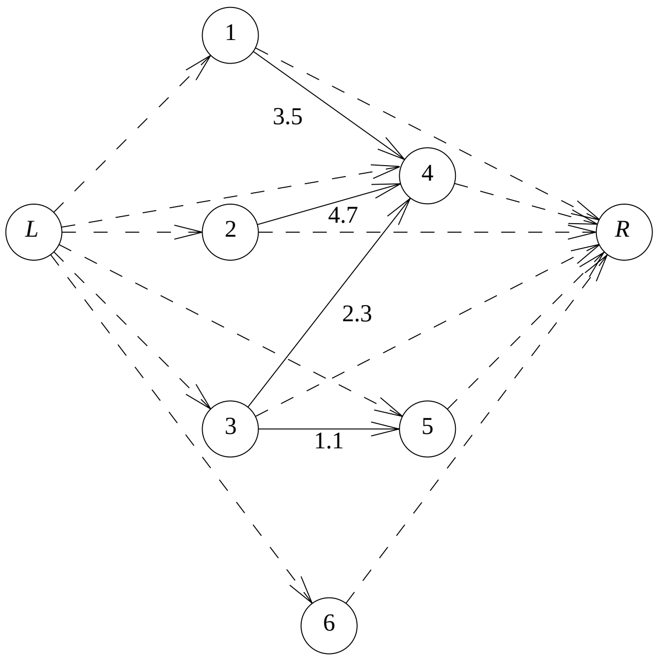


Fig. 2.2a G_H.

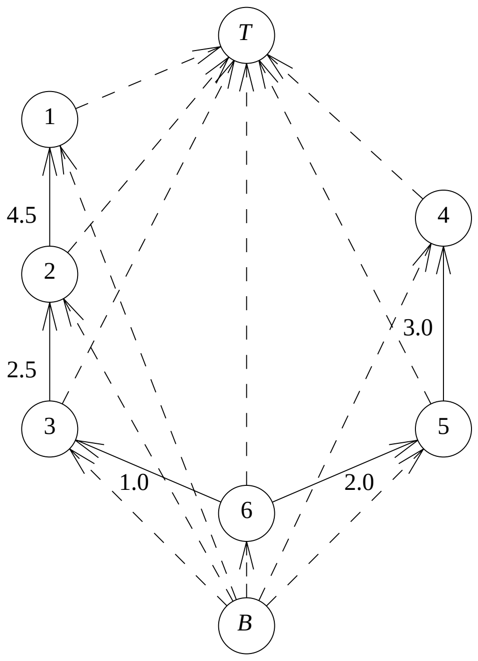


Fig. 2.2b G_V.

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

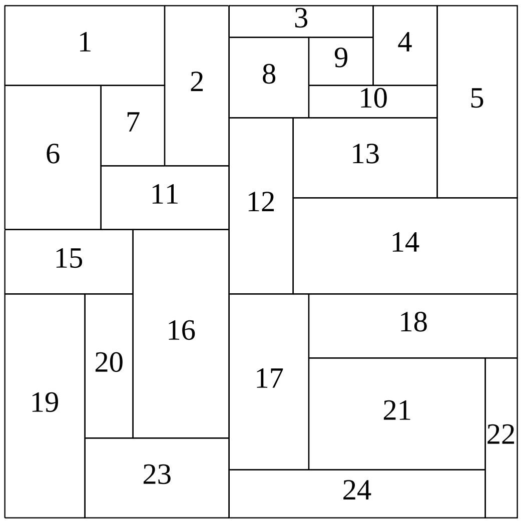


Fig. 2.3a.

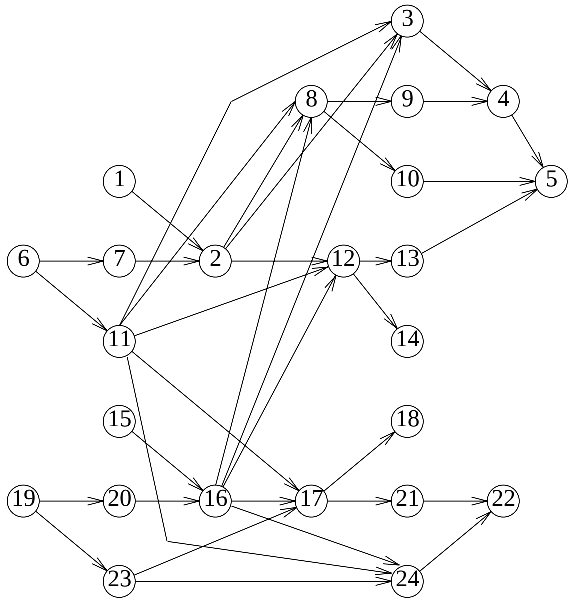


Fig. 2.3b G_H derived from Fig. 2.3a.

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

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

($PRED_V(B_i)$, $SUCC_V(B_i)$ 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 $L \to 2 \to 4 \to R$), whereas the length of the longest path through block 3 is 22.3 (achieved by path $L \to 3 \to 4 \to R$) and the length of the longest path through block 6 is 10 (achieved by path $L \to 6 \to R$).

The aspect ratio for the bounding rectangle, ρ, is used in the computation of s_x^i 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^i and s_y^i 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

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

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 :

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

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 $(a^i_1, b^i_1), \cdots, (a^i_{j_i}, b^i_{j_i})$ where $a^i_1 < \cdots < a^i_{j_i}$ are the possible widths and $b^i_1, \cdots, b^i_{j_i}$ are the corresponding heights. Let the current width of block B_i be a^i_p. Then the new width $w_i{}'$ is $a^i_{p'}$, where

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

f_i is computed according to (1) and ε is a real constant defined by the user. The new height $h_i{}'$ is $b^i_{p'}$. 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
                $s^i_x \leftarrow \rho \cdot l_H(B_i) + \lambda \cdot |STRADDLE_H(B_i)|$;
                $s^i_y \leftarrow l_V(B_i) + \lambda \cdot |STRADDLE_V(B_i)|$;
                $f_i \leftarrow (s^i_y - s^i_x)/(s^i_x + s^i_y)$;
```

```
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 $<P_i, B_i \to B_j, Q_j>$. 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 $<P_j, B_j \to B_i, Q_i>$. 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.

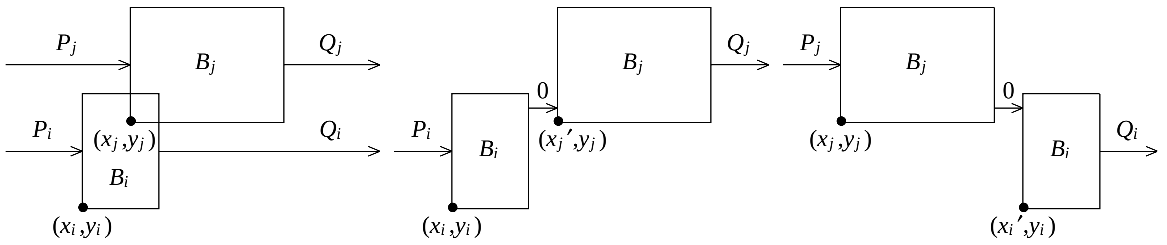


Fig. 2.5a. **Fig. 2.5b**. **Fig. 2.5c**.

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.

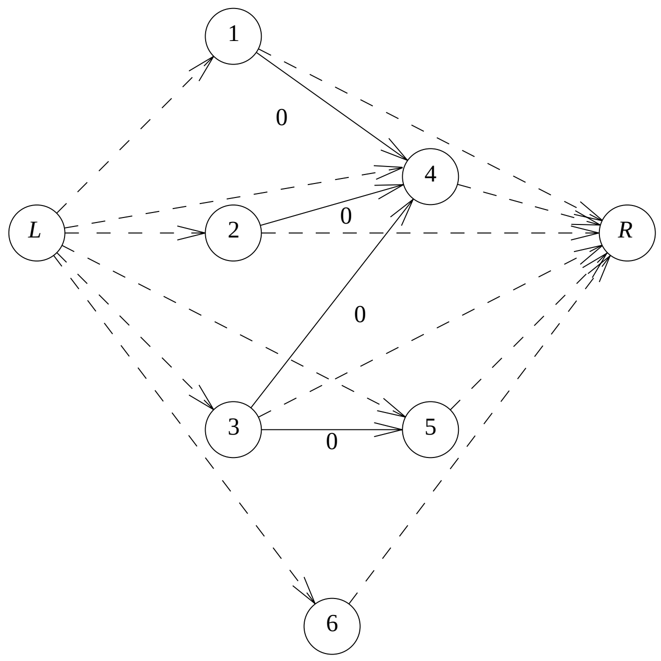


Fig. 2.6a G_H.

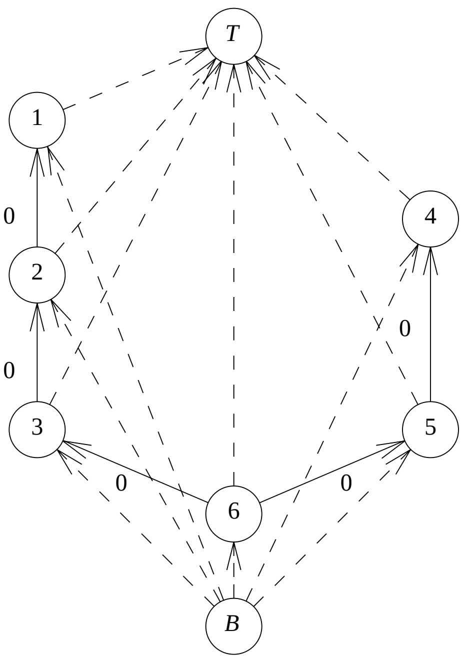


Fig. 2.6b G_V.

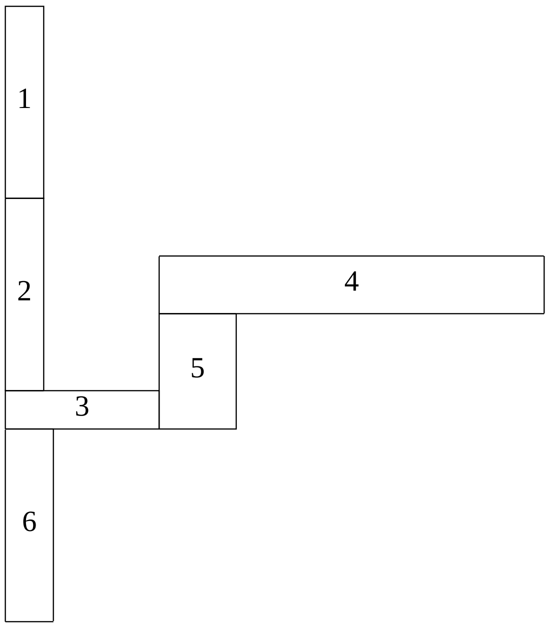


Fig. 2.6c Initial Configuration.

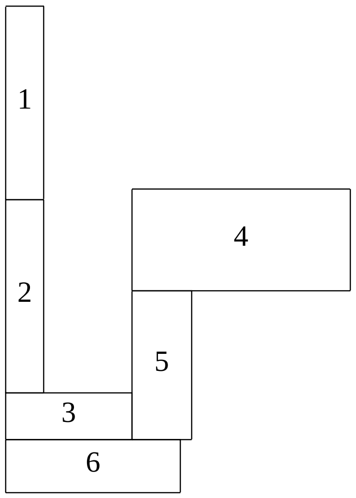


Fig. 2.6d After 10 iterations.

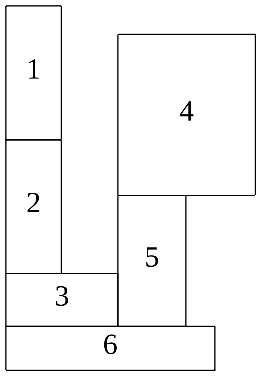


Fig. 2.6e After 20 iterations.

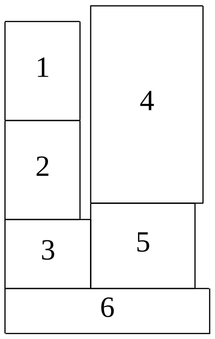


Fig. 2.6f After 50 iterations.

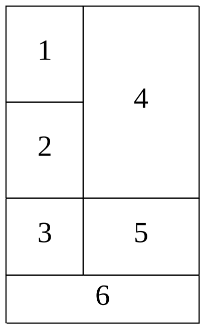


Fig. 2.6g ρ = 1.65, 260 iterations.

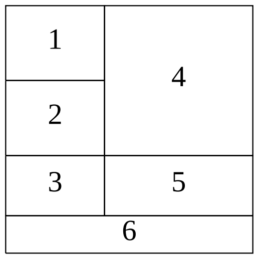


Fig. 2.6h ρ = 1, 330 iterations.

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.

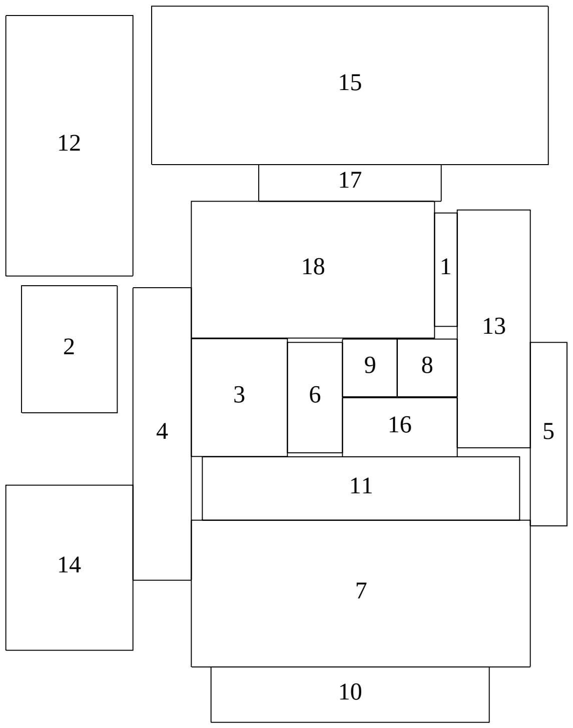


Fig. 2.7a.

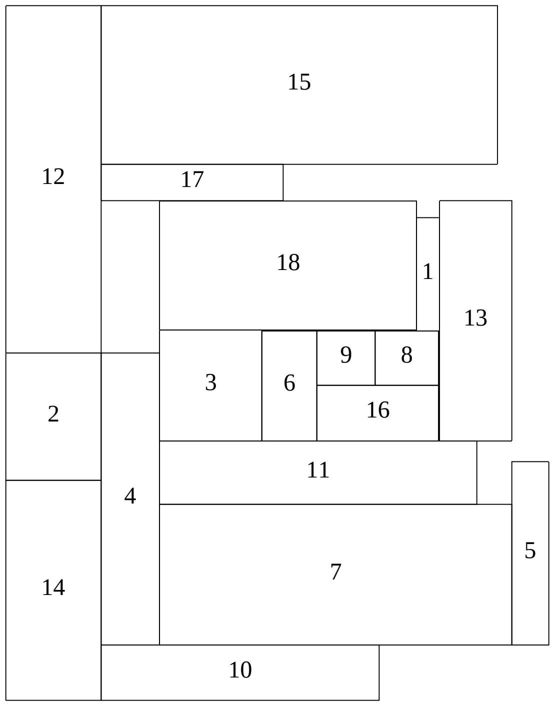


Fig. 2.7b.

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.

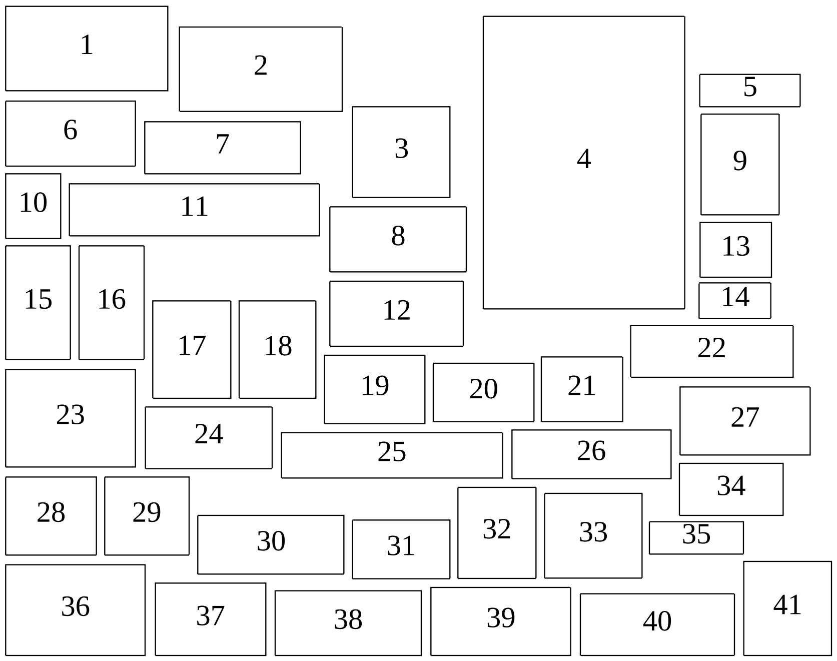


Fig. 2.8a.

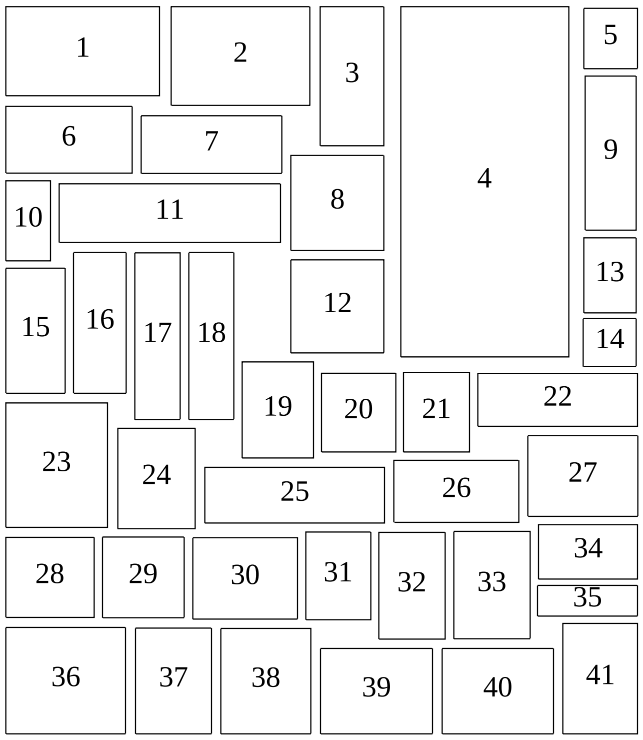


Fig. 2.8b.

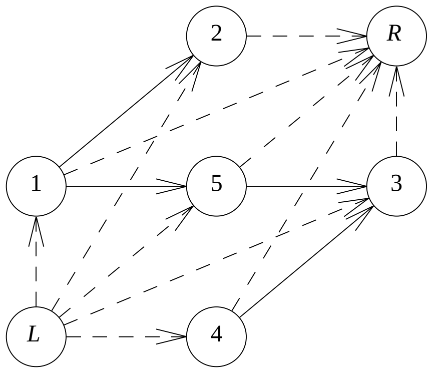


Fig. 2.9a G_H.

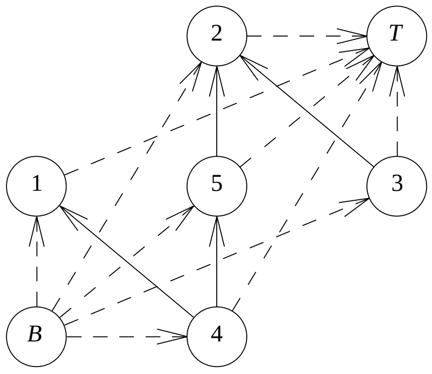


Fig. 2.9b G_V.

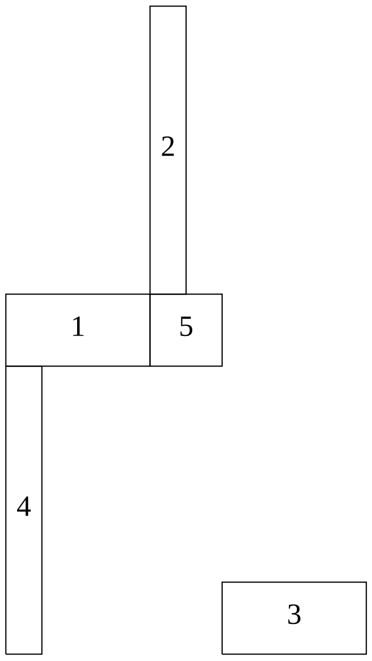


Fig. 2.9c.

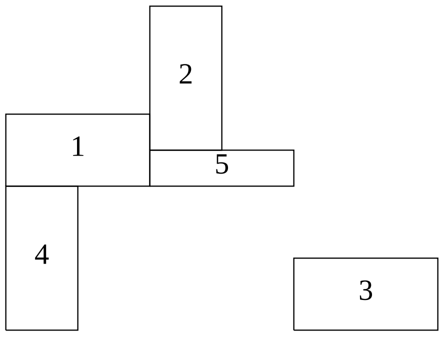


Fig. 2.9d.

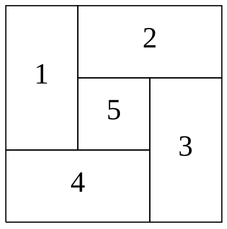


Fig. 2.9e.

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.

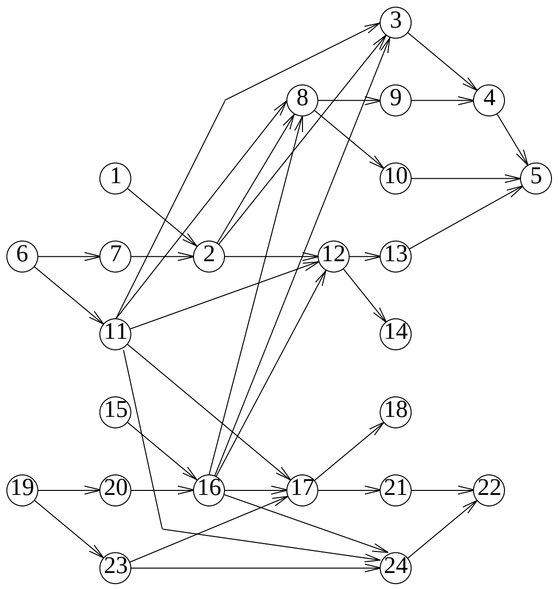


Fig. 2.10a G_H derived from Fig. 2.10c.

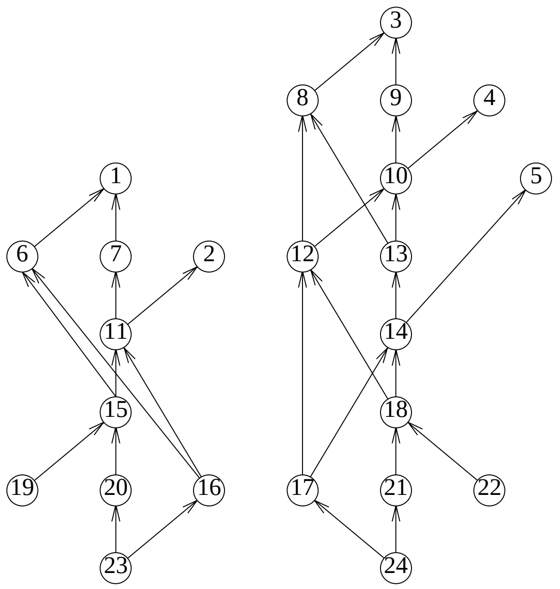


Fig. 2.10b G_V derived from Fig. 2.10c.

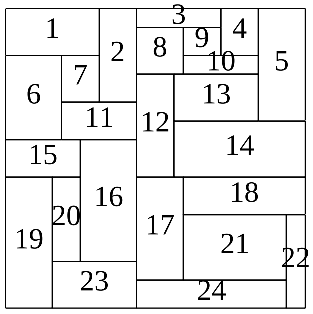


Fig. 2.10c.

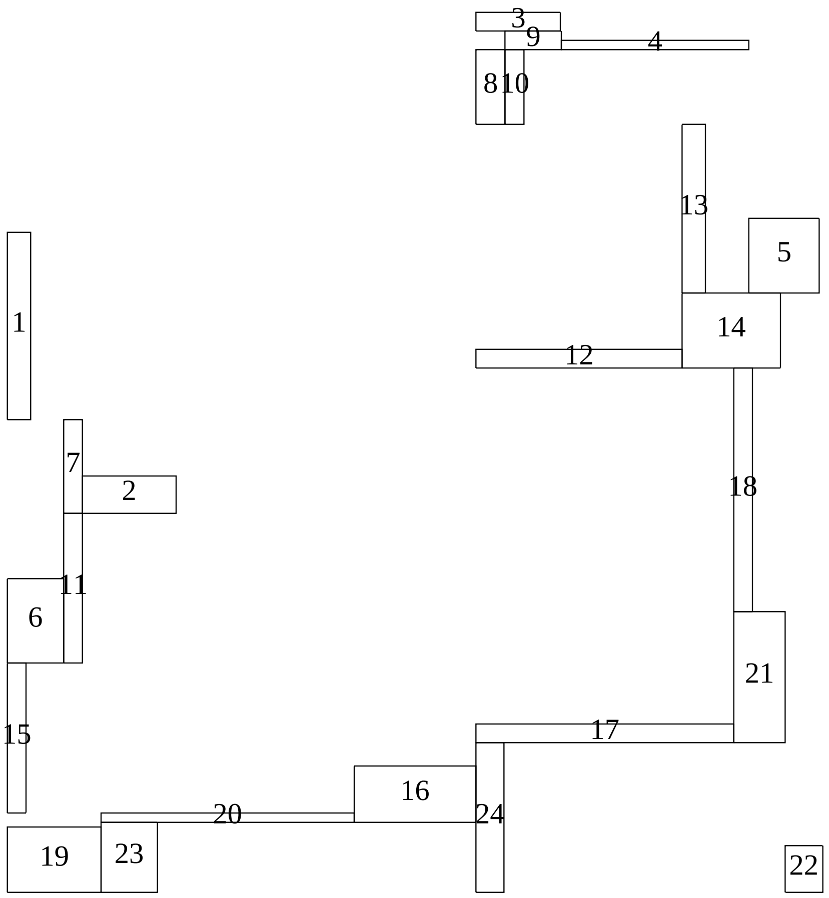


Fig. 2.10d.

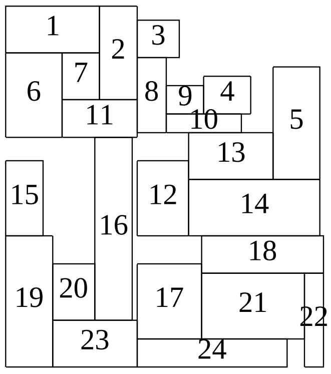


Fig. 2.10e.

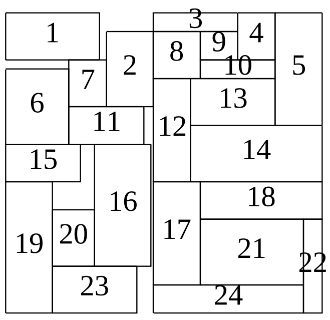


Fig. 2.10f.

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.

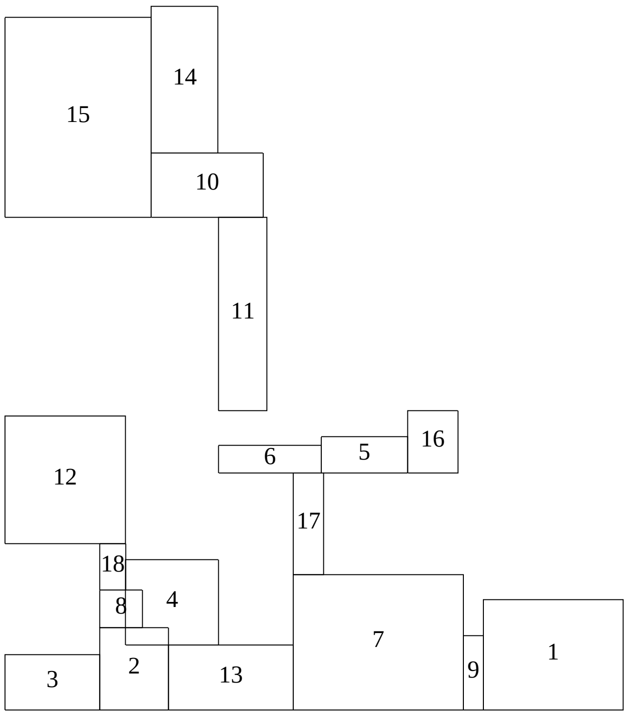


Fig. 2.11a.

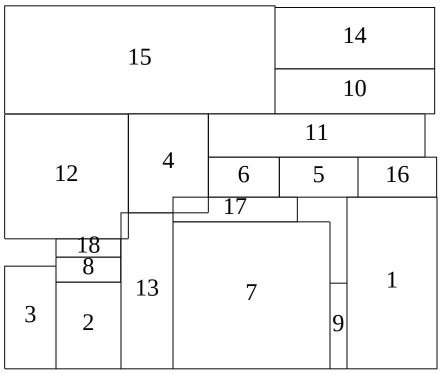


Fig. 2.11b.

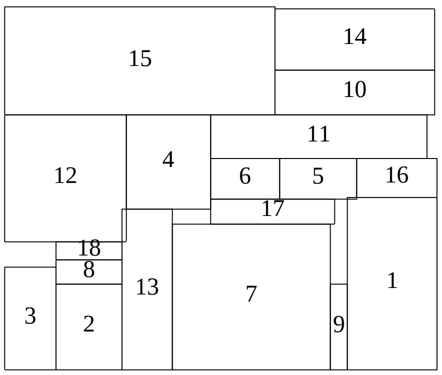


Fig. 2.11c.

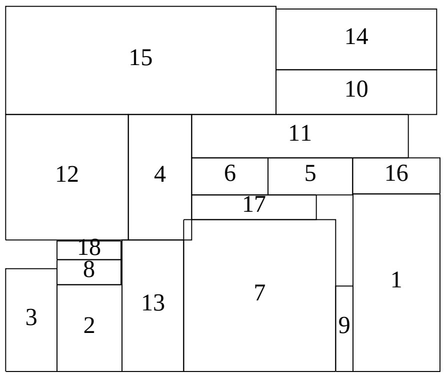


Fig. 2.11d.

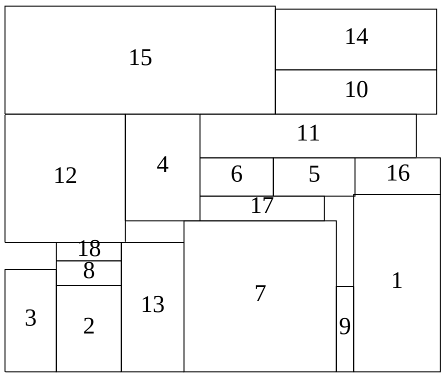


Fig. 2.11e.

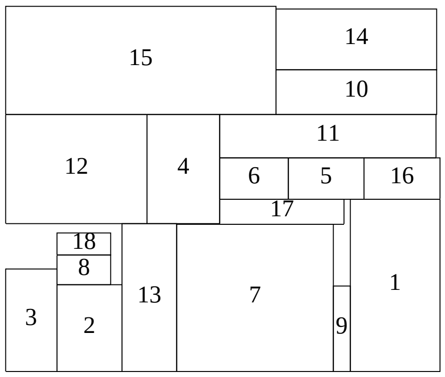


Fig. 2.11f.

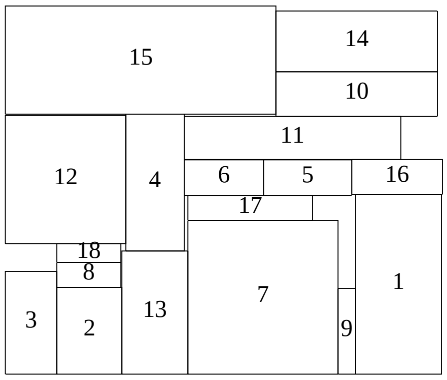


Fig. 2.11g.

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.

Chapter 3

Channel Routing in Quickly Customized Logic

3.1. Introduction

In conventional channel routing, two or more routing (metal) layers are available for interconnections [YoKu82, CoWL88]. While multi-layer architectures provide flexibility in the design of highly integrated circuits, both development time and fabrication time are long compared with those of mask programmable technology. To shorten the turn-around time, one and half layer routing was proposed in some gate array designs [SoCh85, SeSc91] in which a predefined polysilicon layer and a custom metal layer are available for routing. One drawback of this approach is that polysilicon is not an ideal medium for routing because of its poor conductivity. Recently, a new architecture known as Quickly Customized Logic (QCL) [NaES89, YaES89] was proposed. In QCL technology, there is one predefined metal layer and one custom metal layer for routing. The predefined metal layer contains 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 routing, the number of custom-designed masks is reduced from four to one. Experience showed that the turn-around time was shortened to two to five days and that the complexity of the chips (330 to 12K gates) is comparable to those utilizing two-layer customization.

In [SuDS91], the routing problem for a simple channel model in QCL (Figure 3.1) was studied. We propose a new channel model which extends the model 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 compute a set of *ranges* of positions for placing terminals that connect wires crossing from the upper/lower region to the middle region. We show that the ranges can be computed in linear time. We then give a branch-and-bound algorithm to generate a feasible routing solution.

3.2. Channel Model

As in a conventional channel, a channel in QCL technology has *terminal positions* along its top boundary (line TOP) and bottom boundary (line BOTTOM). (See Figure 3.2.)

Terminals placed at terminal positions on lines TOP and BOTTOM will be referred to as the *top* and *bottom* terminals, respectively. There are two metal layers for routing in the channel. They are referred to as *layer I* and *layer II*. The channel is divided into three regions : the *upper region*, *middle region* and *lower region*. The middle region contains *pre-placed vertical wire segments* (or pre-placed segments for short) on layer I as shown in Figure 3.2. Customized wires are allowed only on layer II. (See Figure 3.3.) There are vias at both ends of a pre-placed segment. The locations of the vias along lines P and Q can be viewed as two rows of terminal positions. We assume that there is an underlying grid for the channel. Vertical grid lines are labelled 1, 2, $\cdots$, w from left to right. Horizontal grid lines are labelled 1, 2, $\cdots$, k_u in the upper region; 1, 2, $\cdots$, k_m in the middle region; and 1, 2, $\cdots$, k_l in the lower region. To simplify our discussion, we assume that each net has one top terminal and one bottom terminal. In general, a net may have multiple top and bottom terminals and our algorithm is capable of handling multi-terminal nets.

3.3. Problem Formulation

Let t_i and b_i denote the top and bottom terminal of net i and let $x(t_i)$ and $x(b_i)$ denote the x-coordinate of t_i and b_i, respectively. We assume that the positions of n pairs of terminals $(t_1,b_1), \cdots, (t_n,b_n)$ are given. Without loss of generality, top terminals are labelled $t_1, \cdots, t_n$ from left to right, i.e. $x(t_1) < \cdots < x(t_n)$. Bottom terminals, however, are arranged in an arbitrary order. Let $b_{\pi(1)}, \cdots, b_{\pi(n)}$ be the labels of the bottom terminals where $\pi(i) = j$ if b_j is the ith terminal from left to right. Thus, $x(b_{\pi(1)}) < \cdots < x(b_{\pi(n)})$.

To connect terminals t_i and b_i, we introduce two *intermediate* terminals p_i and q_i. p_i will be placed at some terminal position on line P and q_i will be placed at some terminal position on line Q. (The x-coordinates of terminals p_i and q_i are denoted $x(p_i)$ and $x(q_i)$, respectively.) Terminals t_i and p_i are connected by *river routing* [Ullm84, TuTe91] on layer II in the upper region; so are terminals q_i and b_i in the lower region. (See Figure 3.3.) Terminals p_i and q_i are then connected in the middle region using pre-placed segments on layer I and customized wires on layer II. (See Figure 3.4.) They are connected in one of the three ways shown in Figure 3.5. The three types of wires will be referred to as *trivial*, *-shaped* and *-shaped*. A wire is called *simple* if it is of one of these three types. A *simple routing solution* (for the middle region) is a routing in which the connecting wires are all

simple. By considering only simple routing solutions, the routing problem in the middle region can be transformed into a single-row routing problem with no backward moves and no crossovers [RaSa83] *if* the positions of the intermediate terminals are known. The transformation is done by merging lines P and Q and placing the intermediate terminals in a single row as shown in Figure 3.6.

Our problem can now be formulated as follows : Given the positions of terminals $t_1, \cdots, t_n$, $b_1, \cdots, b_n$ with $x(t_1) < \cdots < x(t_n)$ and $x(b_{\pi(1)}) < \cdots < x(b_{\pi(n)})$,

Fig. 3.4 A 3-dimensional view of part of the middle region.

1. Determine the positions of intermediate terminals $p_1, \cdots, p_n$ on line P and the positions of intermediate terminals $q_1, \cdots, q_n$ on line Q such that

   (i) $x(p_i), x(q_i) \in \{1, \cdots, w\}$.

   (ii) $x(p_1) < \cdots < x(p_n)$ and $x(q_{\pi(1)}) < \cdots < x(q_{\pi(n)})$.

   (iii) $x(p_i) \neq x(q_j)$ if $i \neq j$.

2. Connect t_i and p_i $(1 \le i \le n)$, using no more than k_u horizontal tracks on layer II in the upper region.

3. Connect q_i and b_i $(1 \le i \le n)$, using no more than k_l horizontal tracks on layer II in the lower region [4].

4. Connect p_i and q_i $(1 \le i \le n)$ by simple wires, using no more than k_m horizontal tracks in the middle region.

3.4. The Algorithm

We propose a branch-and-bound algorithm for solving the channel routing problem in the new QCL channel model. Figure 3.7 shows the steps of the algorithm.

Step 1 of the algorithm is to compute the ranges for the intermediate terminals. The *range*, $[\alpha_i^P, \beta_i^P]$, for a terminal p_i $(1 \le i \le n)$ is defined to be the consecutive terminal positions on line P p_i may occupy so that river routing between the n terminal pairs (t_i, p_i) can be completed using no more than k_u horizontal tracks. The range, $[\alpha_i^Q, \beta_i^Q]$, for q_i $(1 \le i \le n)$ is defined analogously. The notion of range is a *new* concept in river routing. In Section 3.4.1, we give an example of the ranges for intermediate terminals and discuss a *linear*-time algorithm for computing the ranges. The correctness of the algorithm is proved in Section 3.4.2.

Step 2 is to determine a lower bound on the number of horizontal tracks needed in the middle region. Given the ranges for intermediate terminals, a *feasible terminal configuration* is an

[4] Since the nature of the routing problem in the upper region is exactly the same as that of the lower region, we shall restrict our discussion to the routing problem in the upper region.

assignment of intermediate terminals to terminal positions such that

1. $\alpha_i^P \le x(p_i) \le \beta_i^P$ and $\alpha_i^Q \le x(q_i) \le \beta_i^Q$ $(1 \le i \le n)$,
2. $x(p_1) < \cdots < x(p_n)$ and $x(q_{\pi(1)}) < \cdots < x(q_{\pi(n)})$.

A lower bound for the number of horizontal tracks needed for routing in the middle region is the minimum of the densities (in the sense of a classical two-layer channel) of all feasible terminal configurations. Although the middle region of a QCL channel is not the same as a classical two-layer channel (because the use of layer I is restricted to the pre-placed segments), the minimum density, $\min_{c \in C} \{\text{density}(c)\}$ where C is the set of all feasible terminal configurations, is clearly still a valid lower bound. If the lower bound exceeds k_m, then there is no routing solution. This enables us to avoid unnecessary computation in some of the problem instances that have no solutions. In [CaWo90], an algorithm that determines the positions of terminals to minimize the density in a two-layer channel is given. We use their algorithm to compute the minimum density in $O(n^2 \cdot w)$ time and use the minimum density as the lower bound for the middle region.

Step 3 is to determine a routing for the middle region. Suppose the positions of terminals p_i and q_i were known, we have seen in Section 3.3 that the routing problem in the middle region is equivalent to the single-row routing problem with no crossovers and no backward moves. However, since only the ranges for these terminals are known, our algorithm employs a branch-and-bound strategy to examine all feasible terminal configurations. The idea here is to find an overlap graph that is two-colorable. The details are given in Section 3.4.3.

Finally, if there is a feasible routing solution for the middle region, routing solutions for the upper and the lower regions can be obtained in $O(n^2)$ time using a standard river routing algorithm [Ullm84] since the positions of all terminals are known. The solutions obtained will not use more than k_u and k_l horizontal tracks, respectively, because terminals p_i and q_i $(1 \le i \le n)$ are all placed within the ranges computed in Step 1.

3.4.1. Ranges for Intermediate Terminals

In the classical river routing problem, terminals on the top and bottom boundaries of a channel are to be connected by disjoint rectilinear wires on a single layer. We shall refer to the number of horizontal tracks and the number of vertical tracks in the channel as its *height* and *width*, respectively. (See Figure 3.8.) The routing problem in the upper region is similar to the river routing problem except that only the positions of the top terminals are known while the positions of the intermediate terminals on line P have not been determined. Given the positions of the top terminals, the width w and the height k_u of the upper region, we compute a range of terminal positions, $[\alpha_i^P, \beta_i^P]$, for terminal p_i $(1 \le i \le n)$ such that if p_i is placed in any position within this range (i.e. $\alpha_i^P \le x(p_i) \le \beta_i^P$), routing can be completed in no more than k_u tracks. (The range $[\alpha_i^Q, \beta_i^Q]$ is defined analogously for terminal q_i in the lower region.) As long as the relative order of the intermediate terminals on line P from left to right is the same as that of the top terminals (so that river routing is possible), the ranges of p_i $(1 \le i \le n)$ are ‘‘independent’’ in the sense that one can choose *any* terminal position for an intermediate terminal from its corresponding range independent of the choices for other intermediate terminals (Theorem 3.1). Figure 3.9 is an example showing the ranges for four terminals in a channel with height 2 and width 10.

The pseudo-code below computes the ranges for p_i’s [5]. (The ranges for q_i’s can be computed analogously.) The call to procedure *find-ranges* $(k_u, w, x[\,], \alpha[\,], \beta[\,])$ computes the ranges for p_i’s. The function *separation* invoked in procedure *find-left-endpoints-of-ranges* is Karplus’ algorithm [Ullm84 p. 449]. It returns the minimum number of tracks required for river routing given the positions of the terminals. The idea behind procedure *find-left-endpoints-of-ranges* is as follows : Try to place $p_1, \cdots, p_n$ (one at a time in this order) at terminal positions on line P as far left as possible such that the number of tracks used for river routing does not exceed $h = k_u$. Thus, $\alpha[i]$ is the leftmost terminal position that p_i can occupy assuming that terminals $p_1, \cdots, p_{i-1}$ are placed at terminal positions $\alpha[1], \cdots, \alpha[i-1]$, respectively. The property of the ranges and the time complexity for computing the ranges in the upper region are stated in Theorem 3.1 and Theorem 3.2.

[5] The pseudo-code for procedure *find-right-endpoints-of-ranges* is analogous to that of procedure *find-left-endpoints-of-ranges* and is omitted.

```
procedure find-ranges ($h$, $w$, $x$[ ], $\alpha$[ ], $\beta$[ ])
(* Finds the range for terminal $p_j$ and stores it in $\alpha[j]$, $\beta[j]$ ($1 \le j \le n$). *)
(* The positions of terminals $t_1, \cdots, t_n$ are given *)
(* and are stored in $x[1], \cdots, x[n]$, respectively. *)
begin
    find-left-endpoints-of-ranges ($h$, $w$, $x$[ ], $\alpha$[ ]);
    find-right-endpoints-of-ranges($h$, $w$, $x$[ ], $\beta$[ ]);
end;        (* find-ranges *)

procedure find-left-endpoints-of-ranges ($h$, $w$, $x$[ ], $\alpha$[ ])
(* Finds the left endpoint of the range for $p_j$ and stores it in $\alpha[j]$ ($1 \le j \le n$). *)
begin
    $i \leftarrow 1$; $column \leftarrow 1$;
    while ($i \le n$ and $column \le w$) do begin
        $\alpha[i] \leftarrow column$;
        (* separation is a function that returns the minimum number *)
        (* of tracks used in connecting $t_1$ with $p_1, \cdots, t_i$ with $p_i$ *)
        if ($separation(x[1], \cdots, x[i], \alpha[1], \cdots, \alpha[i]) \le h$)
            then $i \leftarrow i + 1$;
        $column \leftarrow column + 1$;
    end;
end;        (* find-left-endpoints-of-ranges *)
```

Theorem 3.1 For $\alpha[1], \cdots, \alpha[n]$, $\beta[1], \cdots, \beta[n]$ computed by procedure *find-ranges*, if $\alpha[i] \le x(p_i) \le \beta[i]$ for $1 \le i \le n$ and $x(p_1) < x(p_2) < \cdots < x(p_n)$, then river routing can be completed in no more than h horizontal tracks.

Proof See Section 3.4.2.

Theorem 3.2 Given the positions of n terminals on one side of a river-routing channel of height h and width w, the ranges for the corresponding terminals on the other side of the channel can be determined in $O(n+w)$ time.

Proof See Section 3.4.2.

3.4.2. Proofs of Theorems 3.1 and 3.2

We first introduce the notion of a *left block* and *barrier functions* from the area of river routing [Ullm84] before we present the proof of Theorem 3.1 and Theorem 3.2.

Consider a channel with terminals $u_1, \cdots, u_n$ on its top boundary and terminals $v_1, \cdots, v_n$ on its bottom boundary where the positions of these terminals are all given. u_i and v_i are the two terminals of net i $(1 \le i \le n)$. A *left* (*right*) block is a *maximal* consecutive sequence of nets $i, \cdots, j$ such that $x(u_k) < x(v_k)$ $(x(u_k) > x(v_k))$ for $i \le k \le j$. A net k with $x(u_k) = x(v_k)$ is a *trivial* net and does not belong to any left or right blocks. (See Figure 3.10.) Clearly, the routing of a trivial net does not interfere with the routing of all other nets. Furthermore, the routing of a left block does not interfere with that of other left blocks and right blocks. Thus, without loss of generality, we shall restrict our discussion to a left block.

For a terminal v_i in a left block, one can draw a set of contour lines around it. (See Figure 3.11.) Some of these contour lines are 'barriers' to the routing of other nets in this left block in the sense that the routing for these nets cannot intersect with these barriers. Formally, each contour line can be described by a *barrier function* $g_{v_i}^j$ (or g_i^j for short) such that

$$g_i^j(x) = \begin{cases} 0 & \text{if } x \ge x(v_i) + j \\ j+1 & \text{otherwise.} \end{cases}$$

For example, the barrier function $g_i^1(x) = 0$ if $x \ge x(v_i) + 1$ and 2 otherwise, is a contour line that the routing of net $i+1$ cannot cross for otherwise the spacing between net i and net $i+1$ will be less than 1 which violates the requirement of river routing.

Theorem 3.3 [Ullm84 p. 447] The minimum number of horizontal tracks needed for routing a left block with n nets is $\max_{1 \le i < j \le n} g_i^{j-i}(x(u_j) - x(v_i))$.

Proof of Theorem 3.1 Let $u_1, \cdots, u_n$ be $p_1, \cdots, p_n$ and $v_1, \cdots, v_n$ be $t_1, \cdots, t_n$, respectively. Assume Theorem 3.1 is false, i.e. there exists a set of terminal positions $x_1', \cdots, x_n'$ for terminals $u_1, \cdots, u_n$ with $\alpha[i] \le x_i' \le \beta[i]$ $(1 \le i \le n)$ and $x_1' < \cdots < x_n'$ such that river routing can only be completed using more than h horizontal tracks.

Without loss of generality, assume it is a left block that causes the routing to use more than h tracks. By Theorem 3.3, there exists $i < j$ such that

$$g_i^{j-i}(x_j' - x(v_i)) = j - i + 1 \quad > h$$

$$x_j' - x(v_i) < x(v_i) + (j - i) \quad \text{by definition of } g_i^{j-i}$$

$$\alpha[j] - x(v_i) < x(v_i) + (j - i) \quad \text{since } \alpha[j] \le x_j'$$

$$g_i^{j-i}(\alpha[j] - x(v_i)) = j - i + 1 \qquad > h\,.$$

This implies that placing terminal u_j at position $\alpha[j]$ would require more than h horizontal tracks for routing, contradicting the definition of $\alpha[j]$. □

Proof of Theorem 3.2 In procedure *find-left-endpoints-of-ranges*, the **while** loop is executed at most $n + w$ times. Each time through the while loop, the function *separation* [Ullm84 p.449] is invoked. Function *separation* will apply Theorem 3.3 to the terminals of nets $1, \cdots, i$ passed to it as its argument. The observation here is that it is not necessary to examine the terminals starting from nets $1, 2, \cdots$ each time *separation* is invoked; the terminals of nets $1, \cdots, n$ need only be scanned once among all the invocation of *separation* in the **while** loop. Hence the time complexity of *find-left-endpoints-of-ranges* is $O(n + w + n)$ or $O(n + w)$. □

3.4.3. Simple Routing Solution

In this section, we present a branch-and-bound algorithm for finding a simple routing solution. We shall introduce some definitions, describe our algorithm and give a small example to illustrate the search process.

Let $= \{I_1, \cdots, I_n\}$ be a set of n intervals on a line segment. Each interval I_j is denoted by $I_j = [l_j, r_j]$ $(l_j \le r_j)$ where l_j and r_j are the left and right endpoints of interval I_j, respectively. An *overlap graph* $G(V,E)$ is a simple graph with vertex set $V = \{v_j | \ I_j \in \ \}$ and edge set $E = \{(v_i, v_j) | \ I_i$ overlaps $I_j\}$, i.e. $l_i < l_j < r_i < r_j$ or $l_j < l_i < r_j < r_i$ [Golu80]. A graph G is said to be *two-colorable* if its set of vertices can be partitioned into two disjoint sets (color classes) such that the two vertices each edge is incident with are in different color classes.

Given the positions of $p_1, \cdots, p_n, q_1, \cdots, q_n$, we define n intervals $I_i = [x(p_i), x(q_i)]$ if $x(p_i) \le x(q_i)$ or $I_i = [x(q_i), x(p_i)]$ if $x(q_i) < x(p_i)$ $(1 \le i \le n)$. Note that I_i corresponds to the horizontal wire segment of net i on layer II in the middle region. Let G denote the overlap graph of these n intervals. If G is not two-colorable, then there is no simple routing solution to the given problem instance. Otherwise, the problem of determining whether a -shaped wire or a -shaped wire should be used for each of the nets corresponds to that of determining how the vertices of the corresponding overlap graph should be divided into two color classes. (Each vertex in one color class will correspond to a -shaped wire and each vertex in the other color class will correspond to a -shaped wire.) However, we only know that p_i lies in $[\alpha_i^P, \beta_i^P]$ and q_i lies in $[\alpha_i^Q, \beta_i^Q]$. Thus, we use a branch-and-bound algorithm to examine all feasible terminal configurations. The intermediate terminals are assigned to the terminal positions net by net and the overlap graph is contructed interval by interval correspondingly. For each interval created by the assignment of the two terminals of a net to some terminal positions, the overlap graph is updated to reflect the overlaps among the already existing intervals. If at any time, the overlap graph is not two-colorable (i.e. an odd cycle exists [BoMu76]) or the density of the existing intervals exceed k_m, the current assignment is not feasible. The algorithm will backtrack and proceed to explore another branch of the search tree. If the overlap graph obtained from the n intervals is two-colorable, an $O(n^3)$ algorithm [RaSa83, SuDS91] is used to determine a two-coloring of the graph such that, k, the number of horizontal tracks used, is the minimum. If $k \le k_m$, a solution is found, otherwise the algorithm backtracks to another assignment of the intermediate terminals and iterates.

A small example with five nets, $w = 12$, $k_u = k_l = 2$, $k_m = 1$; and part of the search tree of the branch-and-bound algorithm is shown in Figure 3.12.

3.5. Experimental Results

We applied our algorithm to an industrial example. The example have three channels between four rows of cells. The simple routing solution obtained is shown in Figure 3.13. Note that some of the nets in the examples are multi-terminal nets.

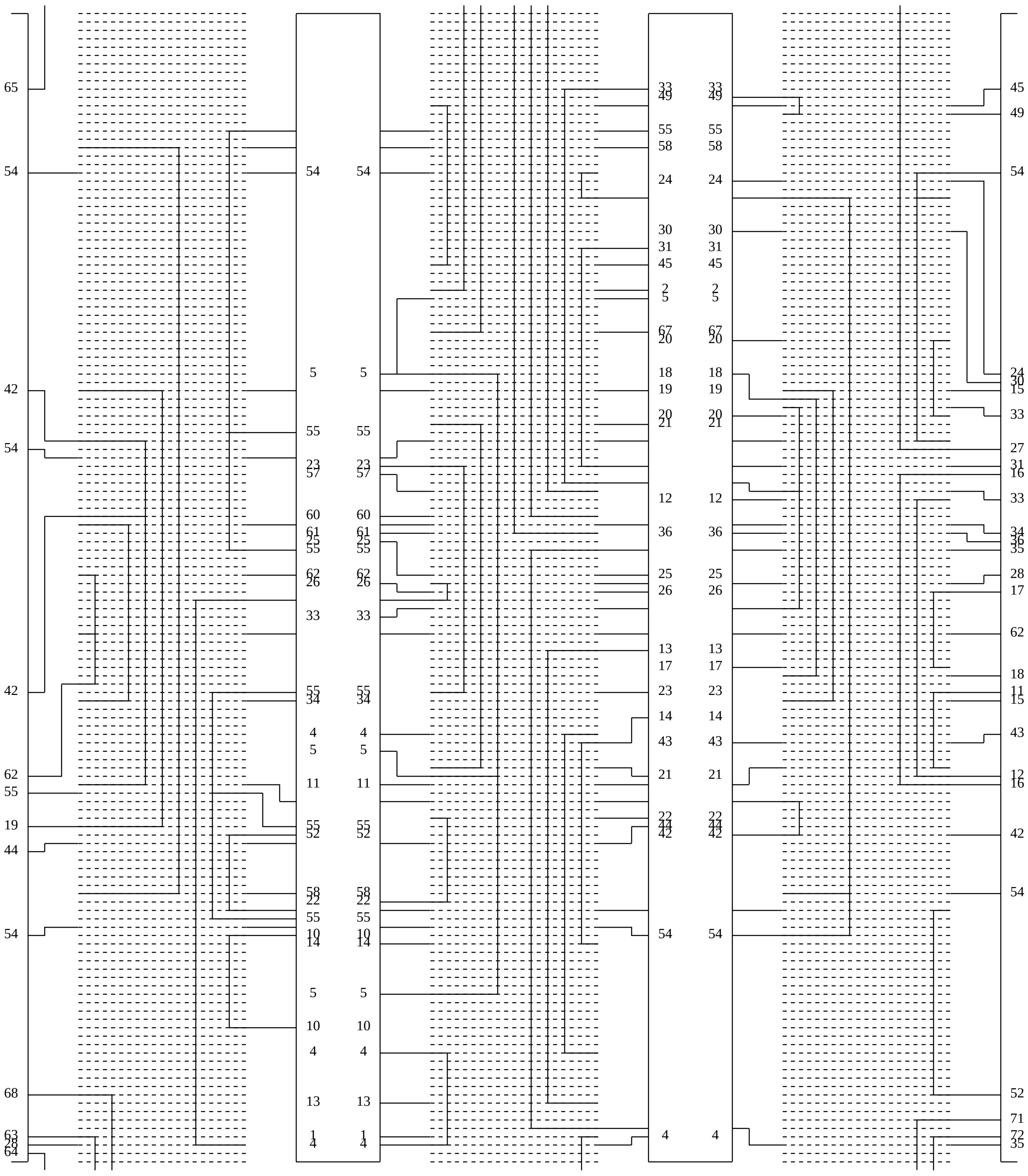


Figure 3.13.

3.6. Summary

Quickly Customized Logic uses one predefined metal layer and one custom metal layer for interconnections. We propose a new channel model for QCL which has three regions, namely, the upper, middle and the lower regions. In the upper and lower regions, river routing techniques are used while in the middle region, single-row routing techniques are used. We give a branch-and-bound algorithm to examine all feasible terminal configurations. If there is a feasible terminal configuration that admits a simple routing solution in the middle region, then it is guaranteed that river routing in the upper and the lower regions can be completed within the prespecified number of horizontal tracks.

Chapter 4

Constraint Relaxation in Graph-based Compaction

4.1. Introduction

A common approach to the solution of the symbolic layout compaction problem is to express the constraints on the positions of circuit elements in terms of a system of linear inequalities S; and then find a feasible solution for S that minimizes a certain linear objective function, e.g. the width of a cell [Boye88, Marp90, BaVa92, LeTa92, YaCD93]. S may contain various types of constraints : hierachical, pitchmatching, separation and connectivity (design rules) and user-defined constraints. Consequently, we often find that S is an over-constrained system of linear inequalities, i.e. there is no feasible solution for S. In the presence of over-constraints, a compactor can either :

(i) identify *some* or *all* of the inequalities that are too restrictive and ask the circuit designer to manually modify the layout; or

(ii) use some heuristics to relax *some* of the inequalities;

so as to remove the over-constraints in S incrementally. These two strategies, however, are ‘‘local’’ in nature since the over-constraints are not resolved all at the same time. Both strategies have been used by different compactors, for example :

- in hierarchical compaction, where over-constraints occur because of interaction among cells at the same as well as different levels of the hierarchy, a graph-theoretic technique was proposed to identify some of the inequalities that cause over-constraints. These inequalities are then recorded in a database to provide feedback for the circuit designer [BaVa93].

- in leaf cell compaction, the system of linear inequalities, S, is modeled by a weighted, directed graph G_S. It is well-known that the system S has a solution if and only if the constraint graph G_S has no positive (directed) cycles [LiWo83]. In other words, over-constraints among circuit elements within the same cell appear in the form of positive cycles in G_S. In [LiWo83], for those edges with negative weights and whose corresponding inequalities cannot be satisfied, *all* the positive cycles that contain these edges are exhaustively enumerated. Such information is presented to the circuit designer who will decide the inequalities to be relaxed. In [King84], edges in a constraint graph are prioritized according to the kind of constraints they represent : user-defined constraints are less important than design rule constraints which are less important than abutment constraints, etc. When a positive cycle is detected, the weights of those edges in the cycle with the lowest priority will be modified. In [Schi88], jog generation and edge weight modification are used to remove positive cycles. The latter selects the edge with the smallest weight in the cycle and decreases its weight by the weight of the cycle.

We consider the problem of removing over-constraints in a constraint graph. The ‘‘local’’ strategy of removing one positive cycle at a time which we mentioned earlier [King84, Schi88] has two drawbacks :

(*i*) the removal of a positive cycle is carried out independent of the removal of other positive cycles. Since positive cycles might have edges in common, this might lead to ‘‘suboptimal’’ results and require computation that might otherwise be unnecessary.

(*ii*) there could be an exponential number of positive cycles in a constraint graph. This limits the size of the problem that can be handled by a compactor using this strategy.

A possible ‘‘global’’ strategy is to first find a subgraph of the constraint graph which has the most number of edges and contains no positive cycles; and then modify the weights of those edges not in the subgraph so as to remove the positive cycles. Unfortunately, finding such a largest subgraph is *NP*-complete (Section 4.3). We propose an alternative strategy and consider the *Constraint Relaxation Problem* : the problem on how to change the edge weights of a constraint graph minimally such that all positive cycles are removed and the length of the longest path is minimized. We show that this problem can be solved in polynomial time by the method of linear programming.

In Section 4.2, we discuss two situations in layout compaction which motivate this work. In Section 4.3, we show that finding a largest subgraph that contains no positive cycles is *NP*-complete.

In Section 4.4, we state the Constraint Relaxation Problem and solve it by the method of linear programming. We give three linear programming formulations for this problem. The three formulations differ in the objectives they try to achieve. In particular, the dual linear program of the third formulation is that of a minimum cost flow problem and hence can be solved quite efficiently. Section 4.5 gives the experimental results.

4.2. Motivation

One way to eliminate positive cycles in a constraint graph is to change the weights of some of its edges. We discuss two situations in layout compaction where a minimal change in edge weights is meaningful. We then give an example of a constraint graph in which different changes in the individual edge weight have different effects on the length of the longest path. This example demonstrates the difficulty in deciding how edge weights should be changed.

For submicron technology with high source to drain resistance, it is important that spacings between circuit elements be tightly controlled. For example, consider Figure 4.1 where parts of three transistors are shown : C_1, C_2, C_3 are contact cuts and G_1, G_2, G_3 are gates. Suppose compaction is carried out in the x-direction. We are interested in minimizing the x-dimension of the diffusion region. Let $G_i.x$ and $C_j.x$ be the unknown x-coordinate of gate G_i $(i = 1, 2, 3)$ and contact C_j $(j = 1,3)$, respectively. It is desirable that the spacing between C_1 and G_1, G_1 and G_2, G_2 and G_3, and G_3 and C_3 be as small as possible. These requirements can be expressed as :

$$G_1.x - C_1.x = a$$

$$G_2.x - G_1.x = 2a + b$$

$$G_3.x - G_2.x = c$$

$$C_3.x - G_3.x = a$$

where a is the minimum spacing between a contact cut and polysilicon, b is the minimum width of a contact cut, and c is the minimum spacing between polysilicon and polysilicon, respectively. These four equations (or eight inequalities), together with other compaction constraints, might form an over-constrained system of inequalities. In this case, some of these equality constraints need to be relaxed and be satisfied as strict inequality constraints. Since we want to minimize the ‘‘stretching’’ of the diffusion region, the problem becomes that of :

$$\begin{aligned} &\min\ \alpha+\beta+\gamma+\alpha \\ &G_1.x - C_1.x = \alpha \\ &G_2.x - G_1.x = \beta \\ &G_3.x - G_2.x = \gamma \\ &C_3.x - G_3.x = \alpha \end{aligned}$$

where $\alpha \geq a$, $\beta \geq 2a + b$ and $\gamma \geq c$. A minimum adjustment in edge weights in the constraint graph will correspond to a minimum increase in the x-dimension of the diffusion region.

Cell compaction with abutment constraints is another situation in which a change in edge weights may be necessary for the removal of positive cycles. Figure 4.2 shows two cells A and B where pin P_i is to be connected to pin Q_i, $i = 1$, 2, 3, by abutment. These requirements can be expressed as :

$$\begin{aligned} P_1.x - Q_1.x &= 0 \\ P_2.x - Q_2.x &= 0 \\ P_3.x - Q_3.x &= 0 \end{aligned}$$

where $P_i.x$ $(Q_i.x)$ is the x-coordinate of pin P_i (Q_i), $i = 1$, 2, 3. These abutment constraints together with the intra-cell constraints of cell A and cell B might create positive cycles in the constraint graph. In this case, it is necessary to relax some of the abutment constraints. For example, the circuit designer might decide to keep the abutment constraints between the pairs of pins (P_1,Q_1) and (P_2,Q_2) but connect P_3 and Q_3 by river routing instead of by abutment [LiCS93]. To minimize the length of the connecting wire, it is desirable to have $|P_3.x - Q_3.x|$ be as small as possible. This corresponds to :

$$\begin{aligned} &\min\ \alpha \\ &P_1.x - Q_1.x = 0 \\ &P_2.x - Q_2.x = 0 \\ &P_3.x - Q_3.x \leq \alpha \end{aligned}$$

$$Q_3.x - P_3.x \leq \alpha$$

A minimum adjustment in edge weights will minimize the length of the connecting wire used in river routing.

The two situations above show the need for an algorithm that can remove positive cycles from a constraint graph with as little change to the edge weights as possible. However, deciding the set of edges and by how much their weights should be changed is a non-trivial problem. Consider the constraint graph G shown in Figure 4.3 where the edge

weights are shown next to the edges. G has a positive cycle $v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_2$. To remove this cycle, the minimum amount of change in edge weights is 4, the weight of the cycle. In the mean time, we want to minimize the length of the longest path from v_1 to v_6. Of course, one could choose to change the edge weights by a large amount. In Figure 4.4, the total amount of change in edge weights is 7; and the length of the longest path from v_1 to v_6 is 0 (edges on the longest path are shown in bold arrows).

Figure 4.5 gives four different ways to remove the positive cycle in G. They all introduce the same total amount of change in edge weights, namely 4, but with different effects on the length of the longest path from v_1 to v_6, $l(v_1, v_6)$, and on the lengths of the longest paths between other pairs of vertices.

4.3. *NP*-completeness Results

In this section, we show that given a constraint graph, the problem of finding a largest subgraph which has the most number of edges and contains no positive cycles is *NP*-complete. Thus, it is not likely that the approach of retaining the largest *number* of edges intact and modifying the weights of the remaining edges will lead to an efficient algorithm for the removal of over-constraints. The *NP*-completeness result can be extended to the problem of finding a largest *consistent* [6] subset of inequalities in a given system of inequalities. This shows that in the case where a system of *general* linear inequalities is used, e.g. hierarchical compaction, finding a largest subset of inequalities that are not over-constrained is a difficult problem.

Let $G = (V, E, w)$ be a constraint graph where $V = \{v_1, \ldots, v_n\}$ is the set of vertices, E is the set of (directed) edges and $w : E \rightarrow$ is the edge weight function, respectively. We shall adopt the convention that v_1 and v_n represent the left and the right cell boundary, respectively (assuming the direction of compaction is along the x-axis) and that w_{ij} denotes the weight of edge (v_i, v_j). We define the *Largest Feasible Subgraph* problem as follows :

LARGEST FEASIBLE SUBGRAPH

INSTANCE : Constraint graph $G = (V, E, w)$, positive integer $M \leq |E|$.

QUESTION : Is there a subset $E' \subseteq E$ with $|E'| \geq M$ such that $G'(V, E', w')$, where w' is w restricted to E', has no positive cycles ?

We shall show that the Largest Feasible Subgraph problem is NP-complete by reduction from the *Feedback Arc Set* problem. The latter is defined as :

FEEDBACK ARC SET [GaJo79, p. 192]

INSTANCE : Directed graph $H = (V, A)$, positive integer $K \leq |A|$.

QUESTION : Is there a subset $A' \subseteq A$ with $|A'| \leq K$ such that A' contains at least one edge from every directed cycle in H ?

Theorem 4.1 LARGEST FEASIBLE SUBGRAPH is NP-complete .

Proof By reduction from FEEDBACK ARC SET. Given an instance, $H = (V, A)$ and integer K, of FEEDBACK ARC SET, consider the instance of LARGEST FEASIBLE SUBGRAPH $G = (V, A, w)$ where $w_{ij} = 1$ for every edge $(v_i, v_j) \in A$ and $M = |A| - K$. Then H has a subset of edges with size at most K and contains at least one edge from every directed cycle in H if and only if G has a subgraph with at least M edges and which does not have any positive cycles. $\square$

Corollary 4.2 Given a system S of linear inequalities, the problem of finding a largest (in terms of

[6] A consistent set of linear inequalities is one in which the solution set it defines is non-empty.

the number of inequalities) consistent subset of S is NP-complete.

Proof By reduction from LARGEST FEASIBLE SUBGRAPH. □

4.4. Problem Formulations and Solutions

Since finding a largest subgraph that has no positive cycles is NP-complete, we propose an alternative approach for removing over-constraints in graph-based compaction. Instead of minimizing the *number* of edges whose weights will be changed, the alternative approach tries to minimize the total *amount* of change in edge weights used in the removal of positive cycles.

We define the *Constraint Relaxation Problem* as follows : given a constraint graph $G = (V, E, w)$ with positive cycles, modify the edge weights of G such that all positive cycles are removed, the change in edge weights is minimal and the length of the longest path from v_1 to v_n is minimized.

We shall give three linear programming formulations for the Constraint Relaxation Problem. The first formulation requires the solution of a sequence of two linear programs (LPs) (Section 4.4.1). We then argue that some of the generality in the first formulation is not necessary and derive a formulation that can be solved by a single LP (Section 4.4.2). Finally, we consider a special case of the second formulation and show that its dual corresponds to that of a minimum cost flow problem and hence can be solved efficiently (Section 4.4.3).

4.4.1. First Formulation

For a given constraint graph $G = (V, E, w)$, its corresponding LP is [PaDL93] :

$$\min x_n - x_1$$
$$x_j - x_i \geq w_{ij}, \ (v_i, v_j) \in E$$

where variable x_i is the unknown x-coordinate of vertex $v_i \in V$. In the presence of positive cycles, some of the edge weights in the constraint graph need to be modified. For each edge (v_i, v_j) of weight w_{ij}, a variable ε_{ij} is introduced and the new edge weight will be $w_{ij} + \varepsilon_{ij}$, where $l_{ij} \leq \varepsilon_{ij} \leq u_{ij}$. l_{ij} and u_{ij} are constants given by the circuit designer to limit the amount of change in the weight of edge (v_i, v_j). For example, if $u_{ij} \leq 0$, it means that the weight of edge (v_i, v_j) may decrease but cannot increase. If an edge weight w_{ij} should not be changed because it corresponds to some important compaction constraint, one can set $l_{ij} = u_{ij} = 0$. (In this case, a simpler way is not to create the variable ε_{ij} for this particular edge at all.)

To determine, Δ, the minimum amount of change in edge weights that is necessary to remove all positive cycles, consider the following mathematical program P1 :

If P1 has a solution, the mathematical program P2 below will determine the minimum longest path length from v_1 to v_n and how much the weight of each edge should be changed subject to the condition that the total change in edge weights is Δ.

P1 and P2, in its current form, are not linear programs because of the absolute value operator. To overcome this limitation, we shall introduce two *non-negative* variables ε_{ij}^{+} and ε_{ij}^{-} for each variable ε_{ij}. The intention is to replace every occurrence of ε_{ij} by $\varepsilon_{ij}^{+} - \varepsilon_{ij}^{-}$ and replace $|\varepsilon_{ij}|$ by $\varepsilon_{ij}^{+} + \varepsilon_{ij}^{-}$. In particular, the new edge weight for edge (v_i, v_j) will be $w_{ij} + \varepsilon_{ij}^{+} - \varepsilon_{ij}^{-}$. For variable ε_{ij}^{+}, there will be a new lower bound λ_{ij}^{+} and a new upper bound μ_{ij}^{+}; and for variable ε_{ij}^{-}, a new lower bound λ_{ij}^{-} and a new upper bound μ_{ij}^{-}. The new lower and upper bounds are defined as :

$$\lambda_{ij}^{+} = \max\{0, \; l_{ij}\}, \qquad \mu_{ij}^{+} = \max\{0, \; u_{ij}\}$$

$$\lambda_{ij}^{-} = \max\{0, -u_{ij}\}, \qquad \mu_{ij}^{-} = \max\{0, -l_{ij}\}.$$

The validity of these new lower and upper bounds can be seen by analyzing the following three cases (assuming $l_{ij} \leq u_{ij}$) :

Case 1 $0 \leq l_{ij}$.

$$l_{ij} = \lambda_{ij}^{+} \leq \varepsilon_{ij}^{+} \leq \mu_{ij}^{+} = u_{ij} \text{ and } 0 = \lambda_{ij}^{-} \leq \varepsilon_{ij}^{-} \leq \mu_{ij}^{-} = 0.$$

Case 2 $u_{ij} \leq 0$.

$$0 = \lambda_{ij}^{+} \leq \varepsilon_{ij}^{+} \leq \mu_{ij}^{+} = 0 \text{ and } -u_{ij} = \lambda_{ij}^{-} \leq \varepsilon_{ij}^{-} \leq \mu_{ij}^{-} = -l_{ij}.$$

Case 3 $l_{ij} < 0$ and $0 < u_{ij}$.

$$0 = \lambda_{ij}^{+} \leq \varepsilon_{ij}^{+} \leq \mu_{ij}^{+} = u_{ij} \text{ and } 0 = \lambda_{ij}^{-} \leq \varepsilon_{ij}^{-} \leq \mu_{ij}^{-} = -l_{ij}.$$

P1 and P2 can now be formulated as two bona fide linear programs :

By solving the linear programs P1′ followed by P2′, we can determine the least amount of change in edge weights that will remove all positive cycles from G and the resultant minimum longest path length from v_1 to v_n.

4.4.2. Second Formulation

In P1 and P2, u_{ij}, the upper bound on ε_{ij} can be some positive constant. Thus, the new edge weight for edge (v_i, v_j), $w_{ij} + \varepsilon_{ij}$, can increase in value. However, this is not necessary since increasing the weight of an edge can never help in removing positive cycles. It is the decrease in weight of some positive-weight edges and/or some negative-weight edges that remove the positive cycles in a constraint graph. Thus, we can assume u_{ij} is non-positive. According to Case 2 of the analysis in Section 4.4.1, $\varepsilon_{ij}^{+} = 0$. Hence, we can remove all occurrences of the variables ε_{ij}^{+} and their corresponding lower bound and upper bound constraints from P1′ and P2′. Furthermore, we consider a new objective function which is a weighted sum of the two objective functions of P1′ and P2′. The result is the linear program P3 shown below (A and B are positive integers defined by the user) :

Intuitively, the more changes made to the individual edge weights, the shorter the longest path from v_1 to v_n. Thus, the objective function of P3 captures the tradeoff between the amount of change in edge weights and the length of the longest path from v_1 to v_n. By using this formulation, only one instead of two LPs need to be solved.

4.4.3. Third Formulation

In this section, we show that a special case of P3 has a dual LP that corresponds to that of a minimum cost flow problem. The minimum cost flow problem [Tarj83] is a network flow problem in which there is a cost c_{ij} associated with an edge (v_i, v_j) in the network and such that a unit of flow along that edge will have a cost of c_{ij}. The objective is to obtain a flow of a pre-specified quantity from some supply vertices to some demand vertices that has the minimum total cost. The minimum cost flow problem can be solved in low order polynomial time and it has been used in the solution of other problems. For example, the problem of determining the minimum number of delay elements inserted into a pipelined system for achieving synchronization has been formulated as a linear program whose dual corresponds to that of a minimum cost flow problem [WoDF89, HuHB91, BoHS92].

We consider the case when the constraints $-\varepsilon_{\bar{i}\bar{j}} \geq l_{ij}$, $(v_i, v_j) \in E$, are absent from P3. In other words, the linear program looks like :

From P4, the new weight of an edge satisfies $w_{ij} - \varepsilon_{\bar{i}\bar{j}} \leq w_{ij} + u_{ij}$, i.e. the new edge weight, $w_{ij} - \varepsilon_{\bar{i}\bar{j}}$, has an upper bound but not a lower bound. The difference between P3 and P4 is that the latter does not have control on how small the weight of an edge will become.

Consider D4, the dual LP [PaSt82] of P4 (the dual variables y_{ij} and z_{ij} used in D4 are shown in P4 next to the inequalities to which they correspond) :

In D4, the variables y_{ij}'s and z_{ij}'s are related by the equations $y_{ij} + z_{ij} = A$. By substituting $z_{ij} = A - y_{ij}$ into the objective function of D4; and replacing the equation $y_{ij} + z_{ij} = A$ by the inequality $y_{ij} \leq A$ (since z_{ij} is non-negative), we can eliminate, for each $(v_i, v_j) \in E$, the variable z_{ij} from D4. The resultant LP is D4′ shown below :

In D4′, the second term in the objective function $-\Sigma A \cdot u_{ij}$ can be dropped since it is a constant. Since maximizing a linear function f is the same as minimizing $-f$, D4′ is a minimum cost flow problem where the cost of each edge is $-(w_{ij} + u_{ij})$ and the flow on each edge is the unknown y_{ij}. The first equation in D4′ is the flow conservation constraint for vertices $v_2, \ldots, v_{n-1}$; the second and the third equations specify that the outgoing flow at v_1 is B and the incoming flow at v_n is B, respectively; and the fourth and the fifth inequalities specify that the flow on each edge must be no more than A and non-negative, respectively.

4.5. Experimental Results

We first give an example similar to the one shown in Figure 4.1 and its solution produced by our first formulation. Consider the following constraint graph G (Figure 4.6) which contains more than one positive cycles :

Suppose we only allow the weights of edges (C_1,G_1), (G_3,C_3), (G_1,C_1), (G_2,G_1), (G_3,G_1) and (C_3,G_3) be changed and we require that the distance between contact C_1 and gate G_1 is the same as that between gate G_3 and contact C_3. By introducing the ε's required by our first formulation, we obtain a new constraint graph G' (Figure 4.7).

The corresponding LPs are :
Solving the linear program Q1, we obtain an optimal solution with $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0$ and $\varepsilon_3 = 0.4$ which minimizes $\Delta(= 0.6)$. In this case, the length of the longest path from C_1 to C_3 is 1.5. If we set $\Delta = 0.6$ and solve the linear program Q2, we obtain $\varepsilon_1 = 0$, $\varepsilon_2 = 0.1$ and $\varepsilon_3 = 0.5$. In this solution, the total amount of change in edge weights remains unchanged, namely 0.6. However, the length of the longest path from C_1 to C_3 becomes 1.4 which is the shortest possible given that $\Delta = 0.6$.

To demonstrate the quality of the solutions produced by our linear programming formulations, we apply the second formulation on four test examples. Each example is a constraint graph with some positive cycles. The number of vertices and the number of edges in each constraint graph are listed in the second and the third column of Table 4.1, respectively. We compare our approach with a heuristic. The heuristic examines each positive cycle in a constraint graph one by one and decreases the smallest edge weight in the cycle by the weight of the cycle. In Table 4.1, Δ_H and Δ_{LP} is the total amount of change in edge weights introduced by the heuristic and by LP, respectively; l_H and l_{LP} is the length of the longest path from v_1 to v_n achieved by the heuristic and LP, respectively. From the four test examples, we see that there is more than 40% difference in the amount of change in edge weights and more than 30% difference in the length of the longest path from v_1 to v_n between the two approaches. This shows that our linear programming formulations indeed produce very good results.

4.6. Summary

We study the problem of removing positive cycles in a constraint graph by modification of edge weights. Previous attempts to solve this problem examine the positive cycles one at a time and use heuristics to determine which edge weight to be modified. We show that the problem can be solved in polynomial time by the method of linear programming. We give three linear program formulations for the problem. In particular, the third formulation has a dual whose linear program is that of a minimum cost flow problem and hence can be solved efficiently.

Chapter 5

Reduction Problem on Inequalities with At Most 2 Variables Per Inequality

5.1. Introduction

Given a consistent system of linear inequalities S, we define the *reduction problem* to be that of finding a *smallest* system (i.e. a system with the least number of linear inequalities) S' such that S and S' have the same set of solutions. The reduction problem comes in two flavors :

(i) the *subsystem reduction problem* in which S' is required to be a subsystem (i.e. subset) of S, and

(ii) the *optimum reduction problem* in which S' is *not* required to be a subsystem of S.

The decision version of the subsystem reduction problem is NP-complete (K-relevancy problem [GaJo79]) while the optimum reduction problem is known to be in P.

We are interested in the reduction problem where the inequalities are of some special forms. In particular, we are interested in *graph-systems* and *2VPI-systems*. A graph-system is a system of *graph-inequalities* (inequalities of the form $x_i - x_j \geq c$ where x_i, x_j are variables and c is a constant) and a 2VPI-system is 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 at least one of the constants a and b is non-zero). We shall review what is known about the *graph reduction problem* and give new results on the *2VPI reduction problem*. The former is a reduction problem in which both S and S' are restricted to be graph-systems and the latter is one in which both S and S' are restricted to be 2VPI-systems. We summarize the known results on the graph reduction problem [PaDL93] and the new results on the 2VPI reduction problem below (see Table 5.1):

(i) The decision version of the subsystem graph reduction problem, and hence that of the subsystem 2VPI reduction problem, is NP-hard. In fact, both decision problems are NP-complete.

(ii) The optimum graph reduction problem can be solved in $O(n^2 \log n + nm)$ time where n is the number of variables and m is the number of graph-inequalities in S.

(iii) The optimum 2VPI reduction problem is in P.

	graph-systems	2VPI-systems
subsystem reduction (decision) problem	NP-complete	NP-complete
optimum reduction problem	$O(n^2 \log n + nm)$	P

Table 5.1.

Graph-systems and 2VPI-systems have been studied in the past because of their special structures. Certain linear programming problems involving these two kinds of systems can be solved efficiently. For example, there are strongly polynomial time algorithms for finding a solution for a graph-system [Leng84] and for a 2VPI-system [Megi83, CoMe91]. For a graph-system in which some of the real variables are restricted to be integer variables, the problem of finding a solution can also be solved in polynomial time [LeSa88]. We are interested in the representation rather than the solution of graph-systems and 2VPI-systems. Our problem is motivated by applications in the area of VLSI design. In the VLSI layout compaction problem, graph-systems are frequently used to express geometric constraints between circuit components [Leng82]. Hence, for a given graph-system, it is important to find a smallest graph-system that has the same solutions [PaDL93]. Incidentally, the optimum graph reduction problem is equivalent to the problem of finding a transitive reduction in a *weighted* directed graph which is a generalization of the unweighted case [AhGU72]. We generalize the graph reduction problem to the 2VPI reduction problem and show that the optimum 2VPI reduction problem can be solved in polynomial time by linear programming techniques.

5.2. Graph Reduction Problem

Let S be a system of linear inequalities on variables $x_1, \cdots, x_n$. A *solution* of S is a vector $(\sigma_1, \cdots, \sigma_n)$ in ^{n} such that setting $x_i = \sigma_i$ for $i = 1, \cdots, n$ will satisfy all inequalities in S. If S' is a system of linear inequalities on variables $x_1, \cdots, x_n$, S' is said to be *equivalent* to S if S' and S have the same set of solutions. For a graph-system S, S' is a *minimum graph-subsytem* of S if S' is a smallest subsystem of S that is equivalent to S; S' is an *optimum graph-reduction* of S if S' is a smallest graph-system that is equivalent to S. A *redundant inequality* is an inequality that can be removed from S such that the resultant system is equivalent to S.

Although it is easy to determine the redundancy of a single inequality in a graph-system, determining a largest set of redundant inequalities to be removed in order to obtain a minimum graph-subsystem is (NP-)hard. This is due to the fact that the removal of a redundant inequality may cause another redundant inequality to become non-redundant. In other words, the order in which redundant inequalities are removed from S determines the number of inequalities in the resultant system. Another concern is that the the number of inequalities in a minimum graph-subsystem can be larger than that of an optimum graph-reduction since the latter is allowed to contain inequalities not in S. In [PaDL93], it was shown that an optimum graph-reduction could be found in $O(n^2 \log n + nm)$ time where n is the number of variables and m is the number of inequalities in S. This result was obtained through a natural correspondence between a graph-system and a ‘‘constraint graph’’. In Sections 5.2.1 and 5.2.2, we shall describe briefly the results in [PaDL93] to prepare for the solution of the optimum 2VPI reduction problem.

5.2.1. Graph-systems and Constraint Graphs

Given a graph-system S, one can construct a weighted, directed graph G^S as follows : for each variable x_i that appears in S, there is a vertex v_i in the vertex set $V(G^S)$; for each inequality $x_i - x_j \geq c$ in S, there is an arc (v_j, v_i) with weight c in the arc set $A(G^S)$. G^S is called the *constraint graph* of S. It is well-known that S has a solution if and only if G^S has no (directed) positive cycles [CoLR90]. Henceforth, we only consider weighted, directed graphs that do not have positive cycles. Figure 5.1 shows a graph-system and its corresponding constraint graph.

$x_2 - x_1 \geq 2$	$x_1 - x_4 \geq -1$
$x_3 - x_1 \geq 2$	$x_2 - x_4 \geq 1$
$x_4 - x_1 \geq 1$	$x_3 - x_4 \geq 1$
$x_5 - x_1 \geq 2$	$x_5 - x_4 \geq 1$
$x_3 - x_2 \geq 0$	$x_3 - x_5 \geq 0$
$x_4 - x_3 \geq -1$	

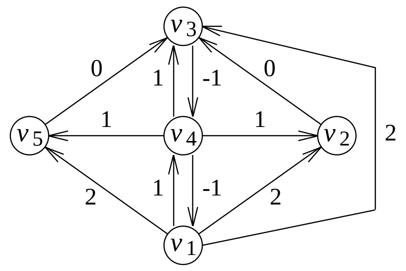
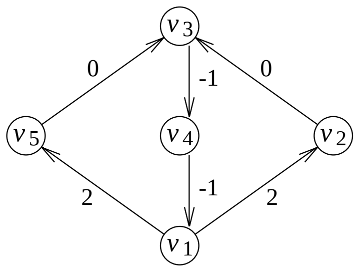


Fig. 5.1 A graph-system and its constraint graph.



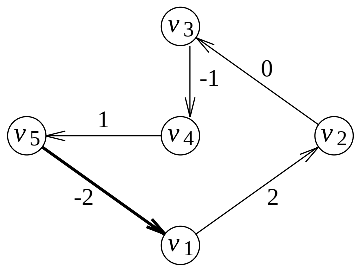


Fig. 5.2 A minimal graph. **Fig. 5.3** An optimum graph.

Let $l_G(u,v)$ denote the length of a longest (directed) path from u to v in G. (The length of a path is equal to the sum of the weights of the arcs that lie on the path.) As a convention, we let $l_G(u,u) = 0$ for any $u \in V(G)$ and $l_G(u,v) = -\infty$ if there is no path from u to v in G.

Theorem 5.1 [PaDL93] Let S_1 and S_2 be two graph-systems on the same set of variables and G^{S_1} and G^{S_2} be their constraint graphs, respectively. S_1 is equivalent to S_2 if and only if the longest distance from v_i to v_j in G^{S_1} is the same as that in G^{S_2} for any vertices v_i and v_j, i.e. $l_{G^{S_1}}(v_i,v_j) = l_{G^{S_2}}(v_i,v_j)$.

In light of Theorem 5.1, the following observation can be made :

(i) An inequality $x_i - x_j \geq c$ is redundant in S if and only if the corresponding arc (v_j,v_i) can be deleted from G^S without affecting the longest distance from v_j to v_i in the resultant graph. We call such an arc a *redundant arc*.

(ii) The subsystem graph reduction problem is the same as the problem of finding a smallest (in terms of the number of arcs) subgraph G' of a constraint graph G that preserves the longest distance between any two vertices in G. G' will be referred to as a *minimal graph*.

(iii) The optimum graph reduction problem is the same as the problem of finding a smallest graph G' that preserves the longest distance between any two vertices of a constraint graph G. G' will be referred to as an *optimum graph*.

For the constraint graph shown in Figure 5.1 (11 arcs), a minimal graph is shown in Figure 5.2 (6 arcs) and an optimum graph is shown in Figure 5.3 (5 arcs). We also have the following results :

Lemma 5.2 The subsystem graph reduction (decision) problem is NP-hard.

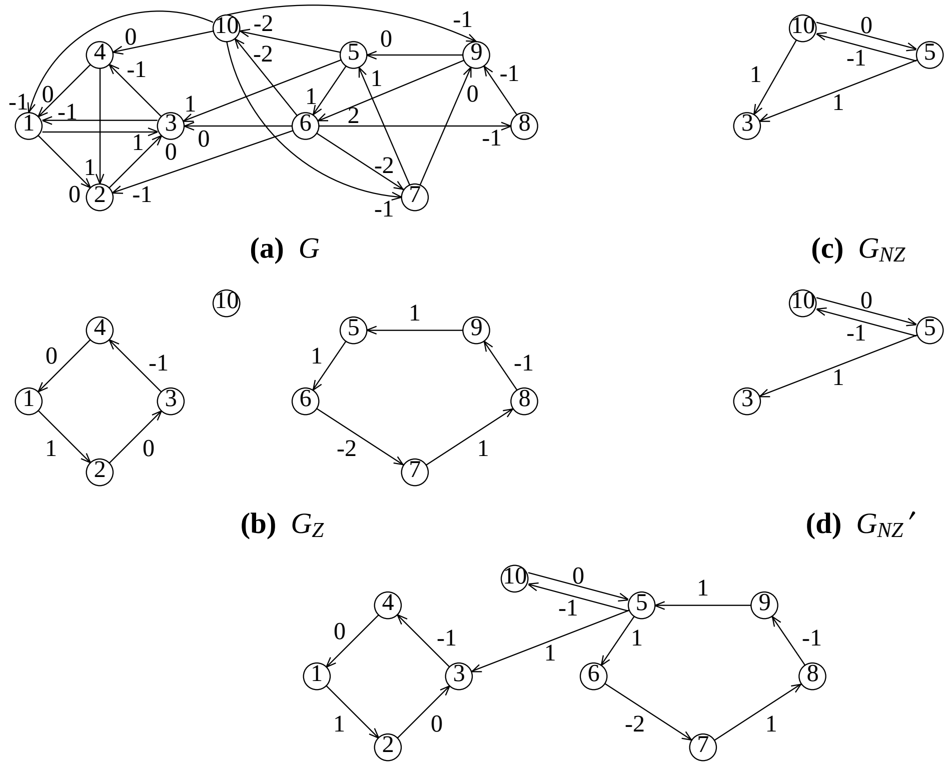
Proof An instance of this problem is a graph-system S and an integer k. The question is to decide if there is a subsystem of S with no more than k inequalities that is equivalent to S. The proof is by reduction from the directed Hamiltonian cycle problem [GaJo79, p. 199]. Given a strongly-connected directed graph G, G has a directed Hamiltonian cycle if and only if G has a minimal graph with no more than $|V(G)|$ arcs when each arc of G is assigned weight 0. □

Theorem 5.3 The subsystem graph reduction (decision) problem and the subsystem 2VPI reduction (decision) problem are NP-complete.

5.2.2. An Algorithm for Optimum Graph Reduction Problem

In this section, we discuss the polynomial-time algorithm in [PaDL93] for computing an optimum graph of a given constraint graph G^S. The graph-system corresponding to the optimum graph will be an optimum graph-reduction of S. We shall use an example to illustrate the ideas of the algorithm.

A zero closed walk is a sequence of arcs $(u_{i_1},u_{i_2}), (u_{i_2},u_{i_3}), \cdots, (u_{i_{k-1}},u_{i_k}), (u_{i_k},u_{i_1})$ such that the sum of the weights of these arcs equals to 0. If G contains *no* zero closed walks, it can be shown that the *unique* optimum graph of G is $G-R_G$ where $G-R_G$ is the graph obtained by deleting from G the arcs in R_G, the set of all redundant arcs in G. An arc (u,v) in R_G can be characterized as one in which there exists a vertex $x \neq u, v$ that satisfies $l_G(u,x) + l_G(x,v) \geq w(u,v)$ where $w(u,v)$ is the weight of the arc (u,v). If l_G, the longest distance between any two vertices, is given, then it is clear that R_G, and thus the optimum graph $G-R_G$, can be determined in $O(mn)$ time where m is the number of arcs and n is the number of vertices in G.



(a) G **(c)** G_{NZ} **(b)** G_Z **(d)** G_{NZ}' **(e)** G_{OPT}

Fig. 5.4 An example for the optimum graph reduction problem.

If G contains zero closed walks (e.g. Figure 5.4(a)), then the optimum graph of G is not unique. We show how an optimum graph can be computed. Two vertices u and v are said to be *dependent* if either $u = v$ or there is a zero closed walk that contains both u and v. The dependency relationship is an equivalence relation and partitions the vertex set $V(G)$ into equivalence classes $V_1, \cdots, V_k$. It can be shown that if u and v are in the same class, then $l_G(u,v) + l_G(v,u) = 0$.

We introduce two weighted, directed graphs G_Z and G_{NZ}. The vertex set of G_Z is the same as that of G; the arc set of G_Z is formed by introducing a directed cycle that contains all vertices in an equivalence class V_i with $|V_i| \geq 2$ $(1 \leq i \leq k)$ (e.g. Figure 5.4(b)). The vertex set of G_{NZ} has one arbitrarily chosen vertex from each equivalence class V_i; the arc set of G_{NZ} has an arc from u to v where $u \in V_i$ and $v \in V_j$ $(i \neq j)$ if and only if there is an arc in G from some vertex in V_i to some vertex in V_j (e.g. Figure 5.4(c)). The weight of an arc (u,v) in G_Z or G_{NZ} is equal to $l_G(u,v)$.

By construction, G_{NZ} has no zero closed walks and hence its optimum graph, G_{NZ}', is $G_{NZ} - R_{G_{NZ}}$ (e.g. Figure 5.4(d)). Putting together G_Z and G_{NZ}', an optimum graph of G, G_{OPT}, is obtained (e.g. Figure 5.4(e)). We summarize the steps of the algorithm below :

Input : Weighted, directed graph G.
Output : Optimum graph G_{OPT}.

```
1. Compute $l_G$.
2. Determine equivalence classes $V_1, \cdots, V_k$.
3. Construct $G_Z$ and $G_{NZ}$.
4. Compute $G_{NZ}'$.
5. $G_{OPT} \leftarrow G_Z \cup G_{NZ}'$.
```

The time complexity of the algorithm is dominated by the computation of l_G in Step 1. This can be achieved in $O(n^2 \log n + nm)$ time by an all-pair longest path algorithm [CoLR90].

5.3. Optimum 2VPI Reduction Problem

Based on the observation that a graph-inequality is also a 2VPI-inequality, we extend the results on graph-systems to 2VPI-systems. In this section, we introduce the formal counterpart of the notions of ‘‘longest distance’’, ‘‘redundant arc’’, ‘‘zero closed walk’’ and ‘‘dependent’’. With the help of these extended notions, a polynomial-time algorithm that resembles the one for the optimum graph reduction problem can be devised to solve the optimum 2VPI reduction problem.

Let S be a system of m linear inequalities on n variables $x_1, \cdots, x_n$. S can be written as $\mathrm{A}x \geq b$ or $\{aiTx \geq b_i \mid i = 1, \cdots, m\}$ where A is an $m \times n$ matrix, $x = (x_1, \cdots, x_n)TR$, $b = (b_1, \cdots, b_m)TR$ and aiT is the ith row of A. A *positive* vector is a vector in which all of the entries are non-negative and at least one of the entry is positive. We assume that S is *consistent*, i.e.

$$\exists\, y \ (m \times 1 \text{ positive vector) such that } yTR\,\mathrm{A} = 0V \text{ and } yTRb > 0$$

and *simple*, i.e.

$$\text{for } i \neq j,\ \exists\, c > 0 \text{ such that } caiT = ajT.$$

The first assumption guarantees that the set of solutions of S is non-empty. The second assumption ensures that no inequality in S is trivially implied by another inequality. For example, only the second inequality in the system $\{x_1 - x_2 \geq 1,\ x_1 - x_2 \geq 3,\ 3x_1 - 3x_2 \geq 6\}$ should be retained. Henceforth, we only consider systems of linear inequalities that are consistent and simple.

Definition 5.4 S' is a *minimum subsystem* of S if S' is a smallest subsystem of S that is equivalent to S; S' is an *optimum reduction* of S if it is a smallest system that is equivalent to S.

Definition 5.5 For a 2VPI-system S, S' is a *minimum 2VPI-subsystem* of S if S' is a smallest subsystem of S that is equivalent to S; S' is an *optimum 2VPI-reduction* of S if it is a smallest 2VPI-system that is equivalent to S.

> **Remark** : In general, an optimum 2VPI-reduction of S may contain more inequalities than an optimum reduction of S. However, the optimum reduction may contain inequalities with more than 2 variables per inequality.

Definition 5.6 Let u be an $n \times 1$ vector and P be a system of linear inequalities on variables $x_1, \cdots, x_n$. The *weight* of u with respect to P is defined as

$$w_P(u) = \min uTRx \text{ subject to } P.$$

(The weight w_P corresponds to the longest distance l_G.)

For a system S and a vector a_i such that aiT is one of the rows in A, let $S \backslash a_i \equiv S - \{aiTx \geq b_i\}$ be the system obtained by removing from S the inequality $aiTx \geq b_i$.

Definition 5.7 An inequality $aiTx \geq b_i$ is *redundant* in S if $w_{S \backslash a_i}(a_i) \geq b_i$.

(The notion of a redundant inequality corresponds to that of a redundant arc.)

Definition 5.8 If $\exists$ positive vector $y = (y_1, \cdots, y_m)TR$ such that $yTR\,\mathrm{A} = 0V$ and $yTRb = 0$, then the set of inequalities $\{ajTx \geq b_j \mid y_j > 0\}$ is called a *0-combination* in S.

(The notion of a 0-combination corresponds to that of a zero closed walk.)

Remark : Every inequality $ajTx \geq b_j$ in a 0-combination in S is *tight*, i.e. $ajTx = b_j$. To see this, let $z = 1/y_j\,(y_1, \cdots, y_{j-1}, 0, y_{j+1}, \cdots, y_m)TR$. $zTR\,\mathrm{A} = -ajT$ and $zTRb = -b_j$. Since $zTR\,\mathrm{A}x \geq zTRb$, we have $-ajTx \geq -b_j$. Together with the inequality $ajTx \geq b_j$, we get $ajTx = b_j$.

For a 2VPI-system S, let R_1 be a binary relation on variables $x_1, \cdots, x_n$ such that $x_i R_1 x_j$ if either $x_j = d$ in all the solutions of S for some constant d, or x_i and x_j appear together in some tight inequality. Let R be the binary relation such that $x_i R x_j$ if and only if $x_i R_1^* x_j$ and $x_j R_1^* x_i$ where R_1^* is the transitive closure of R_1.

Definition 5.9 x_i is said to be *dependent* on x_j if $x_i R x_j$. Since *dependency*, i.e. R, is an equivalence relation, the variables $x_1, \cdots, x_n$ are partitioned into equivalence classes $X_1, \cdots, X_k$. (The notion of two variables being dependent corresponds to that of two vertices being dependent.) A *representative* of an equivalence class is an arbitrarily chosen variable in that class. We denote the representatives of $X_1, \cdots, X_k$ by $x_{i_1}, \cdots, x_{i_k}$.

In Section 5.3.1, we give an algorithm for the optimum 2VPI reduction problem. The steps of the algorithm are explained with the aid of a small example. In Section 5.3.2, we state some lemmas and theorems which prove the correctness of the algorithm. The proofs of these lemmas and theorems are given in Section 5.3.3.

5.3.1. An Algorithm for Optimum 2VPI Reduction Problem

We present a polynomial-time algorithm for computing an optimum 2VPI-reduction. Since each step of the algorithm can be completed in polynomial time, the optimum 2VPI reduction problem is in P. We shall use an example to illustrate the ideas of the algorithm. The steps of our algorithm are as follows :

```
Input  : 2VPI-system $S$ : $Ax \geq b$.
Output : Optimum 2VPI-reduction $S_{OPT}$.
1. Find all tight inequalities $Bx = d$ in $Ax \geq b$ and solve $Bx = d$.
2. Compute $R_1$ and $R$ to determine equivalence classes $X_1, \cdots, X_k$.
3. Construct $S_Z$ and $S_{NZ}$.
4. Compute $S_{NZ}'$.
5. $S_{OPT} \leftarrow S_Z \cup S_{NZ}'$.
```

Given a 2VPI-system S (e.g. Figure 5.5(a)), Step 1 is to find all the tight inequalities in S. An inequality $aiTx \geq b_i$ in S is tight if the value of max $aiTx$ subject to $Ax \geq b$ equals b_i. Since there are m inequalities in S, Step 1 requires the solution of m linear programs. We shall refer to the set of tight inequalities as the system of equations $\mathrm{B}x = d$. (To simplify our discussion, if a variable x_j does not appear in any of the tight inequalities found, a trivial tight inequality $0\,x_j = 0$ is added to $\mathrm{B}x = d$ (e.g. x_{15} in Figure 5.5(b)). The method of Gaussian elimination can then be used to compute the solution x which, in general, will contain some free variables (e.g. Figure 5.5(c)).

Step 2 is to partition the set of variables into equivalence classes. The relation R_1 can be obtained from the system $\mathrm{B}x = d$ and its solution. In Figure 5.5(d), a ''1'' in entry (i,j) means that $x_i R_1 x_j$. Notice that for a variable that has a fixed value as its solution, e.g x_1, $x_i R_1 x_1$ for all i. The relation R can be obtained by computing the transitive closure of R_1. R partitions the set of variables into equivalence classes $X_1, \cdots, X_k$. In the example, $X_1 = \{x_1, \cdots, x_7\}$, $X_2 = \{x_8, x_9, x_{10}, x_{11}\}$, $X_3 = \{x_{12}, x_{13}, x_{14}\}$ and $X_4 = \{x_{15}\}$. The representatives of the equivalence classes are x_1, x_8, x_{12} and x_{15}.

For the variables in the equivalence classes, a new system of 2VPI-inequalities S_Z is introduced to represent the tight inequalities that they satisfy (e.g. Figure 5.5(f)). The

$$
\begin{bmatrix}
1 & 1 \\
& -2 & -1 \\
-2 & & 1 \\
3 & 1 \\
& -1 & & 2 \\
-3 & & & -2 \\
& & & & 2 & 3 \\
& & & & -2 & -3 \\
& & & & -1 & -4 \\
& & & & & 2 & 1 \\
& & & & 1 & & -2 \\
& & & & & & & & -1 & 3 \\
& & & & & & & -2 & & -1 \\
& & & & & & & 6 & 1 \\
& & & & & & & & -1 & & -2 \\
& & & & & & & -3 & & & 1 \\
& & & & & & & & & 3 & 2 \\
& & & & & & & & & & & 2 & 1 \\
& & & & & & & & & & & -2 & -1 \\
& & & & & & & & & & & 1 & & -3 \\
& & & & & & & & & & & -1 & & 3 \\
& & & & & & & & & & & & 1 & 6 \\
& & & & & & & & & & & & & & 1 \\
& & 2 & & & & & & & & & & & & 1 \\
& -6 & & & & & & & -1 \\
& & & & -2 & & & & & & & & -1 \\
& & & & & & & & & 1 & & & -1 \\
& & & & & & & & & & -1 & & & 9 \\
& & & & & & & 1 & & & & -1
\end{bmatrix}
x \ge
\begin{bmatrix}
-5 \\ 5 \\ 5 \\ -11 \\ 2 \\ 9 \\ 8 \\ -8 \\ -9 \\ 7 \\ -5 \\ 5 \\ -2 \\ 1 \\ -7 \\ 3 \\ 10 \\ 13 \\ -13 \\ -14 \\ 14 \\ 27 \\ 2 \\ 1 \\ 11 \\ -13 \\ -5 \\ 51 \\ -8
\end{bmatrix}
$$

(a) $S : \mathrm{A}x \ge b$

$$
\begin{bmatrix}
1 & 1 \\
& -2 & -1 \\
-2 & & 1 \\
3 & 1 \\
& -1 & & 2 \\
-3 & & & -2 \\
& & & & 2 & 3 \\
& & & & -2 & -3 \\
& & & & -1 & -4 \\
& & & & & 2 & 1 \\
& & & & 1 & & -2 \\
& & & & & & & & -1 & 3 \\
& & & & & & & -2 & & -1 \\
& & & & & & & 6 & 1 \\
& & & & & & & & -1 & & -2 \\
& & & & & & & -3 & & & 1 \\
& & & & & & & & & & & 2 & 1 \\
& & & & & & & & & & & -2 & -1 \\
& & & & & & & & & & & 1 & & -3 \\
& & & & & & & & & & & -1 & & 3 \\
& & & & & & & & & & & & & & 0
\end{bmatrix}
x =
\begin{bmatrix}
-5 \\ 5 \\ 5 \\ -11 \\ 2 \\ 9 \\ 8 \\ -8 \\ -9 \\ 7 \\ -5 \\ 5 \\ -2 \\ 1 \\ -7 \\ 3 \\ 13 \\ -13 \\ -14 \\ 14 \\ 0
\end{bmatrix}
$$

The set of tight inequalities in S.

(b) $\mathrm{B}x = d$

$$
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix}
=
\begin{bmatrix} -3 \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \\ 3 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 0 \end{bmatrix}
+
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -6 \\ -2 \\ 3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} x_8
+
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -2 \\ 1/3 \\ 0 \end{bmatrix} x_{12}
+
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} x_{15}
$$

(c) The solution of $\mathrm{B}x = d$.

								x_j							
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
x_i	1	1	1	1	1	1	1	1	1	1	1				
	1	1	1	1	1	1	1	1	1	1	1				
	1	1	1	1	1	1	1	1	1	1	1				
	1	1	1	1	1	1	1	1	1	1	1				
	1	1	1	1	1	1	1					1	1	1	
	1	1	1	1	1	1	1					1	1	1	
	1	1	1	1	1	1	1					1	1	1	
	1	1	1	1	1	1	1								1

(d) R_1

								x_j							
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
	1	1	1	1	1	1	1								
x_i								1	1	1	1				
								1	1	1	1				
								1	1	1	1				
								1	1	1	1				
												1	1	1	
												1	1	1	
												1	1	1	
															1

(e) R

$$
\begin{aligned}
x_1 & \ge -3 \\
-x_1 + x_2 & \ge 1 \\
- x_2 + x_3 & \ge 1 \\
- x_3 + x_4 & \ge 1 \\
- x_4 + x_5 & \ge 1 \\
- x_5 + x_6 & \ge 1 \\
- x_6 + x_7 & \ge 1 \\
- x_7 & \ge -3
\end{aligned}
\qquad
\begin{aligned}
6x_8 + x_9 & \ge 1 \\
- x_9 + 3x_{10} & \ge 5 \\
- 3x_{10} - 2x_{11} & \ge -12 \\
-6x_8 + 2x_{11} & \ge 6 \\
\\
2x_{12} + x_{13} & \ge 5 \\
- x_{13} - 6x_{14} & \ge -41 \\
- 2x_{12} + 6x_{14} & \ge 36
\end{aligned}
$$

(f) S_Z

$$
\begin{aligned}
x_{15} & \ge 2 \\
2x_3 + x_{15} & \ge 1 \\
-6x_2 - x_9 & \ge 11 \\
-2x_5 - x_{13} & \ge -13 \\
x_{10} - x_{13} & \ge -5 \\
-x_{11} + 9x_{14} & \ge 51 \\
x_8 - x_{12} & \ge -8
\end{aligned}
$$

(g) T : Inequalities from S with variables in different equivalent classes.

$$
\begin{aligned}
x_{15} & \ge 2 \\
x_{15} & \ge 3 \\
x_8 & \ge 0 \\
x_{12} & \ge -3 \\
-x_8 + x_{12} & \ge -1 \\
-x_8 + x_{12} & \ge 0 \\
x_8 - x_{12} & \ge -8
\end{aligned}
$$

(h) S_{NZ} : Inequalities expressed in terms of representatives.

$$
\begin{aligned}
x_{15} & \ge 3 \\
x_8 & \ge 0 \\
\\
-x_8 + x_{12} & \ge 0 \\
x_8 - x_{12} & \ge -8
\end{aligned}
$$

(i) S_{NZ}'

Fig. 5.5 An example for the optimum 2VPI reduction problem.

inequalities in S_Z are obtained from the solution of $Bx = d$. Next, let T be the set of inequalities in S that are not tight (e.g. Figure 5.5(g)). From the solution x of $Bx = d$, T can be transformed into a system S_{NZ} where all the variables are representatives of equivalence classes (e.g. Figure 5.5(h)). By construction, S_{NZ} has no 0-combinations and its optimum 2VPI-reduction, S_{NZ}' can be found by discarding all the redundant inequalities in S_{NZ} (e.g. Figure 5.5(i)). Let S_{OPT} be the system consisting of inequalities in S_Z and S_{NZ}'. In Section 5.3.2, we shall see that S_{OPT} is an optimum 2VPI-reduction of S.

5.3.2. Statements of Lemmas and Theorems

Lemma 5.10 Let S be a (general) system of linear inequalities and RI be the set of redundant inequalities in S. If there are no 0-combinations in S, then $S - RI$ is equivalent to S.

Theorem 5.11 If there are no 0-combinations in S, then $S - RI$ is the unique (up to multiplication by positive constants) optimum reduction of S.

By Theorem 5.11, if S is a 2VPI-system with no 0-combinations, then $S - RI$ is the optimum 2VPI-reduction of S. If S has 0-combinations, the following results are needed to establish the correctness of the algorithm.

Lemma 5.12 For a system of tight inequalities $Bx = d$ where the rank of the matrix B is r, $r+1$ inequalities are necessary and sufficient to represent the solutions of the system.

Lemma 5.13 Let S_1 and S_2 be two equivalent systems of linear inequalities with no redundant inequalities. S_1 and S_2 can be written as

$$S_1: \quad A_1 x = b_1 \qquad\qquad S_2: \quad A_2 x = b_2$$
$$\qquad C_1 x \geq d_1 \qquad\qquad\qquad C_2 x \geq d_2$$

where $A_i x = b_i$ is the set of tight inequalities and $C_i x \geq d_i$ is the set of non-tight inequalities in S_i, $i = 1, 2$ and

(i) rank of A_1 = rank of A_2 and

(ii) $A_1 x = b_1$ and $A_2 x = b_2$ have the same set of free variables.

Theorem 5.14 Given a 2VPI-system S, S_{OPT} produced by the algorithm in Section 5.3.1 is an optimum 2VPI-reduction of S.

Proof Clearly, S_{OPT} is equivalent to S. Let S' be an optimum 2VPI-reduction of S. Claim S_{OPT} and S' have the same number of inequalities and hence S_{OPT} is an optimum 2VPI-reduction of S. By construction, the set of tight inequalities and non-tight inequalities in S_{OPT} are S_Z and S_{NZ}', respectively. Let T and NT be the set of tight and non-tight inequalities of S'. By Lemma 5.13, S_Z and T have the same set of free variables. Hence, S_{NZ}' and NT can be expressed as two systems of inequalities involving only these free variables. Since the latter two have no 0-combinations, by Theorem 5.11 they are the same set of inequalities and hence have the same number of inequalities. The minumum number of inequalities to represent S_Z and that to represent T are the same because they have the same rank (Lemma 5.13) and hence the same number of inequalities (Lemma 5.12). □

5.3.3. Proofs of Lemmas and Theorems

Lemma 5.10 If there are no 0-combinations in S, then $S - RI$ is equivalent to S.

Proof Without loss of generality, assume that the first p inequalities in S are redundant, i.e. $RI = \{L_1 \geq 0, \cdots, L_p \geq 0\}$. For $i \in \{1, \cdots, p\}$, consider the following two linear programs :

$$\begin{array}{cc}
\text{min } aiTx \text{ subject to} & \text{max } \alpha_i TRb \text{ subject to} \\
a_1 TRx \geq b_1 & \alpha_i TR\, \mathrm{A} = aiT \\
\vdots & \alpha_i TR \geq 0V \\
\cancel{aiTx \geq b_i} & \alpha_{ii} = 0 \\
\vdots & \text{where } \alpha_i TR = (\alpha_{i1}, \cdots, \alpha_{im}). \\
a_m TRx \geq b_m &
\end{array}$$

The one on the left is the linear program used in the definition of $w_{S \backslash a_i}(a_i)$ (recall Definition 5.6); the one on the right is its dual linear program. By the Duality Theorem and the fact that $w_{S \backslash a_i}(a_i) = b_i$, there exists a positive vector $\alpha_i TR = (\alpha_{i1}, \cdots, \alpha_{i,i-1}, 0, \alpha_{i,i+1}, \cdots, \alpha_{im})$ that satisfies the dual linear program and such that $\alpha_i TRb = b_i$. In other words, each redundant inequality can be expressed as a combination of other inequalities in S as follows :

$$\begin{array}{lclclclclcll}
L_1 & = & 0\,L_1 & + & \alpha_{12}L_2 & + \cdots + & \alpha_{1p}L_p & + \cdots + & \alpha_{1m}L_m & & (1.1)\\
L_2 & = & \alpha_{21}L_1 & + & 0\,L_2 & + \cdots + & \alpha_{2p}L_p & + \cdots + & \alpha_{2m}L_m & & (1.2)\\
\vdots & & & & & & & & \vdots & & \vdots\\
L_p & = & \alpha_{p1}L_1 & + & \alpha_{p2}L_2 & + \cdots + & 0\,L_p & + \cdots + & \alpha_{pm}L_m & & (1.p)
\end{array}$$

We shall show that $L_2, \cdots, L_p$ can be expressed as a combination of inequalities without L_1 as follows :

$$\begin{array}{lclclclclc}
L_2 & = & 0\,L_1 + & 0\,L_2 & + & \beta_{23}L_3 & + \cdots + & \beta_{2p}L_p & + \cdots + & \beta_{2m}L_m\\
L_3 & = & 0\,L_1 + & \beta_{32}L_2 & + & 0\,L_3 & + \cdots + & \beta_{3p}L_p & + \cdots + & \beta_{3m}L_m\\
\vdots & & & & & & & & & \vdots\\
L_p & = & 0\,L_1 + & \beta_{p2}L_2 & + & \beta_{p3}L_3 & + \cdots + & 0\,L_p & + \cdots + & \beta_{pm}L_m
\end{array} \qquad (2)$$

where $\beta_{ij} \geq 0,\ \beta_{ii} = 0$ and at least one $\beta_{ij} > 0$ in each row of (2). The β_{ij}'s are obtained from (1.1), $\cdots$, (1.p) in the following way :

Case 1. $\alpha_{i1} = 0,\ i \in \{2, \cdots, p\}$.

Set $\beta_{ij} = \alpha_{ij}$ for $j = 2, \cdots, m$.

Case 2. $\alpha_{i1} > 0,\ i \in \{2, \cdots, p\}$.

Substituting equation (1.1) into equation (1.i), we get

$$\begin{array}{lcl}
L_i & = & \alpha_{i1}\,(\alpha_{12}L_2 + \cdots + \alpha_{1i}L_i + \cdots + \alpha_{1m}L_m) \quad + \\
 & & (\alpha_{i2}L_2 + \cdots + 0\,L_i + \cdots + \alpha_{im}L_m) \qquad \text{or}\\
(1-\alpha_{i1}\alpha_{1i})L_i & = & \sum_{j=2}^{i-1}(\alpha_{i1}\alpha_{1j} + \alpha_{ij})L_j + \sum_{j=i+1}^{m}(\alpha_{i1}\alpha_{1j} + \alpha_{ij})L_j
\end{array} \qquad (3)$$

If $\alpha_{1j} = \alpha_{ij} = 0$ for $j = 2, \cdots, i-1, i+1, \cdots, m$, then equation (1.1) gives $L_1 = \alpha_{1i}L_i$ and equation (1.i) gives $L_i = \alpha_{i1}L_1$, contradicting the fact that S is simple. Hence, some coefficient of L_j on the right-hand side of equation (3) is positive. If $1-\alpha_{i1}\alpha_{1i} \leq 0$, then equation (3) gives

$$-(1-\alpha_{i1}\alpha_{1i})L_i + \sum_{j=2}^{i-1}(\alpha_{i1}\alpha_{1j} + \alpha_{ij})L_j + \sum_{j=i+1}^{m}(\alpha_{i1}\alpha_{1j} + \alpha_{ij})L_j = 0Vx + 0,$$

contradicting that S has no 0-combinations. Hence, $1-\alpha_{i1}\alpha_{1i} > 0$ and

$$L_i = \frac{1}{1-\alpha_{i1}\alpha_{1i}}\left(\sum_{j=2}^{i-1}(\alpha_{i1}\alpha_{1j} + \alpha_{ij})L_j + \sum_{j=i+1}^{m}(\alpha_{i1}\alpha_{1j} + \alpha_{ij})L_j\right).$$

Set $\beta_{ij} = \dfrac{\alpha_{i1}\alpha_{1j} + \alpha_{ij}}{1-\alpha_{i1}\alpha_{1i}}$ for $j = 2, \cdots, i-1, i+1, \cdots, m$.

The existence of (2) shows that for $i = 2, \cdots, p$, L_i can be expressed as a combination of inequalities without L_1 and L_i. Hence, S is equivalent to $S - \{L_1 \geq 0\}$. The same argument can be applied to (2) to show that $S - \{L_1 \geq 0\}$ is equivalent to $S - \{L_1 \geq 0, L_2 \geq 0\}$. Applying this argument p times shows that S is equivalent to $S - \{L_1 \geq 0, \cdots, L_p \geq 0\}$. □

Theorem 5.11 If there are no 0-combinations in S, then $S-RI$ is the unique (up to multiplication by positive constants) optimum reduction of S.

Proof By Lemma 5.10, $S-RI$ is equivalent to S. Let $S-RI = \{L_1 \geq 0, \cdots, L_q \geq 0\}$ be the set of non-redundant inequalities in S and let $S' = \{L_1' \geq 0, \cdots, L_p' \geq 0\}$ be an optimum reduction of S. We shall show that each inequality in $S-RI$ is a positive multiple of some inequality in S' and hence $S-RI$ is the unique (up to multiplication by positive constants) optimum reduction of S.

Choose S' to be an optimum reduction of S that has the largest number of inequalities in common with $S-RI$. Let $|S' \cap (S-RI)| = t-1$ $(t \geq 1)$ and $L_1' = L_1, \cdots, L_{t-1}' = L_{t-1}$. If $t-1 = q$, $S-RI \subseteq S'$. This implies $S-RI = S'$ since S' is an optimum reduction of S. If $t-1 = p$, $S' \subseteq S-RI$. This implies $S-RI = S'$ for otherwise $L_t \geq 0, \cdots, L_q \geq 0$ are redundant inequalities in $S-RI$, a contradiction.

Thus, $t-1 < q$ and $t-1 < p$. Since $S-RI$ is equivalent to S', $w_{S'}(a_i) = w_{S-RI}(a_i) = b_i$ and $w_{S-RI}(a_j') = w_{S'}(a_j') = b_j'$ where $L_j' = a_j' TRx - b_j'$. By the Duality Theorem, there exist positive vectors $(\alpha_{i1}, \cdots, \alpha_{ip})TR$, $(\beta_{j1}, \cdots, \beta_{jq})TR$ such that

$$L_i = \sum_{j=1}^{p} \alpha_{ij} L_j' \qquad i = t, \cdots, q \qquad (4)$$

and

$$L_j' = \sum_{k=1}^{q} \beta_{jk} L_k \qquad j = t, \cdots, p \qquad (5)$$

From equations (4) and (5),

$$\begin{aligned} L_t &= \sum_{j=1}^{t-1} \alpha_{tj} L_j' + \sum_{j=t}^{p} \alpha_{tj} L_j' \\ &= \sum_{j=1}^{t-1} \alpha_{tj} L_j + \sum_{j=t}^{p} \alpha_{tj} \sum_{k=1}^{q} \beta_{jk} L_k \\ &= \sum_{k=1}^{t-1} \left(\alpha_{tk} + \sum_{j=t}^{p} \alpha_{tj} \beta_{jk} \right) L_k + \sum_{k=t}^{q} \left(\sum_{j=t}^{p} \alpha_{tj} \beta_{jk} \right) L_k \end{aligned}$$

$$= \sum_{k=1}^{q} \gamma_k L_k$$

where $\gamma_k = \begin{cases} \alpha_{tk} + \sum_{j=t}^{p} \alpha_{tj}\beta_{jk} & k \in \{1, \cdots, t-1\} \\ \sum_{j=t}^{p} \alpha_{tj}\beta_{jk} & k \in \{t, \cdots, q\}\ . \end{cases}$

Rearranging terms, we get $(1-\gamma_t)L_t = \sum_{k=1}^{t-1}\gamma_k L_k + \sum_{k=t+1}^{q}\gamma_k L_k$.

Case 1. $\exists\ k \in \{1, \cdots, t-1, t+1, \cdots, q\}, \gamma_k > 0$.

If $(1-\gamma_t) \le 0$, then $-(1-\gamma_t)L_t + \sum_{k=1}^{t-1}\gamma_k L_k + \sum_{k=t+1}^{q}\gamma_k L_k = 0Vx + 0$, a 0-combination in S which is a contradiction.

If $(1-\gamma_t) > 0$, then $L_t = \frac{1}{1-\gamma_t}\left(\sum_{k=1}^{t-1}\gamma_k L_k + \sum_{k=t+1}^{q}\gamma_k L_k\right)$,

contradicting that L_t is a non-redundant inequality. Thus, Case 1 is impossible.

Case 2. $\forall\ k \in \{1, \cdots, t-1, t+1, \cdots, q\}, \gamma_k = 0$.

Since $\gamma_1 = \gamma_2 = \cdots = \gamma_{t-1} = 0$, $\alpha_{t1} = \alpha_{t2} = \cdots = \alpha_{t,t-1} = 0$. If $\alpha_{tt} = \alpha_{t,t+1} = \cdots = \alpha_{tp} = 0$, equation (4) gives $L_t = 0Vx + 0$, a contradiction. Thus, at least one of $\alpha_{tt}, \alpha_{t,t+1}, \cdots, \alpha_{tp}$ is positive. In fact, we claim that exactly one of them is positive because if $\alpha_{tr} > 0$ and $\alpha_{ts} > 0$ for some $r \neq s$, $r, s \in \{t, t+1, \cdots, p\}$, $\gamma_k = 0$ implies $\beta_{rk} = \beta_{sk} = 0$ for $k = 1, \cdots, t-1, t+1, \cdots, q$. Equation (5) gives

$$L_r' = \sum_{k=1}^{q}\beta_{rk}L_k = \beta_{rt}L_t \quad \text{and} \quad L_s' = \sum_{k=1}^{q}\beta_{sk}L_k = \beta_{st}L_t\ .$$

Hence $L_r' = (\beta_{rt}/\beta_{st})\, L_s'$, contradicting S' is simple. (Note that β_{rt} and β_{st} are positive because $(\beta_{r1}, \cdots, \beta_{rq})TR$ and $(\beta_{s1}, \cdots, \beta_{sq})TR$ are positive vectors.)

By claim, $\alpha_{tr} > 0$ and $\alpha_{tj} = 0$ for $j \neq r$. Thus $L_t = \sum_{j=1}^{p}\alpha_{tj}L_j' = \alpha_{tr}L_r'$. But then $S'' = S' - \{L_r' \ge 0\} \cup \{L_t \ge 0\}$ is an optimum reduction of S with $|\,S'' \cap (S-RI)\,| = t > |\,S' \cap (S-RI)\,|$, this contradicts the choice of S'. Hence $S-RI = S'$. □

Lemma 5.12 For a system of tight inequalities $\mathrm{B}x = d$ where the rank of the matrix B is r, $r+1$

inequalities are necessary and sufficient to represent the solutions of the system.

Proof (Sufficiency) Let the system of equations $Bx = d$ be $\{b_1 x = d_1, \ldots, b_m x = d_m\}$. Without loss of generality, let $b_1 x = d_1, \ldots, b_r x = d_r$ be r linearly independent rows. The following $r+1$ inequalities represent the same solution space:

$$\begin{array}{rcl} b_1 x & \geq & d_1 \\ & . & \\ & . & \\ & . & \\ b_r x & \geq & d_r \\ -(b_1 + \cdots + b_r) x & \geq & -(d_1 + \cdots + d_r) \end{array}$$

(Necessity) Let

$$\begin{array}{rcl} \beta_1 x & \geq & \delta_1 \\ & . & \\ & . & \\ & . & \\ \beta_k x & \geq & \delta_k \end{array}$$

be a minimal system of linear inequalities that has the same solution space as $Bx = d$. Claim $k > r$. Assume otherwise, consider

$$\begin{array}{rcl} \beta_1 x & = & \delta_1 \\ & . & \\ & . & \\ & . & \\ \beta_{k-1} x & = & \delta_{k-1} \\ \beta_k x & \geq & \delta_k \end{array}$$

Let Σ_1 denote the set of solutions satisfying the first $k-1$ equations and Σ_2 denote the set of solutions satisfying the last inequality in the above system. We have

$$\dim(\Sigma_1) \geq n - (k - 1) \geq n - r + 1$$

since the rank of a system of $k-1$ equations is at most $k-1$. Consequently

$$\dim(\Sigma_1 \cap \Sigma_2) \geq n - r + 1.$$

Now $(\Sigma_1 \cap \Sigma_2)$ is a subset of the solution set of $Bx = d$ and the latter only has dimension $n - r$, this implies

$$n - r + 1 \leq n - r, \text{ a contradiction.} \quad \square$$

5.4. Summary

We study the 2VPI reduction problem, i.e. the problem of finding a smallest 2VPI-system S' that has the same solution space as a given 2VPI-system S. We show that the optimum 2VPI reduction problem (where S' can contain inequalities not in S) can be solved in polynomial time through $O(m)$ linear programs where m is the number of inequalities in S. On the other hand, the subsystem 2VPI reduction problem (where S' must be a subset of S) is NP-complete.

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.