

Copyright © 1984, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

PLACEMENT ALGORITHMS AND APPLICATIONS  
TO VLSI DESIGN

by

C-K. Cheng

Memorandum No. UCB/ERL M84/40

16 May 1984

(cover)

PLACEMENT ALGORITHMS AND APPLICATIONS TO VLSI DESIGN

by

C-K. Cheng

Memorandum No. UCB/ERL M84/40

16 May 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering  
University of California, Berkeley  
94720



# Placement Algorithms and Applications to VLSI Design

Ph.D.

*Chung-Kuan Cheng*

EECS Dept.

Signature: 

Committee Chairman

## ABSTRACT

With the advent of VLSI technology, the number of components in a chip becomes very large. In view of the growing complexity of the chip and the need for fast turnaround time, the importance of automatic layout is quite obvious.

While there are many effective and efficient wiring algorithms currently available which have been used extensively, more work need to be done in partitioning and placement. This dissertation deals with theoretical study of partitioning and placement and the implementation of algorithms for chip design.

There currently exist three basic strategies for automatic chip layout, namely: the standard-cell, the gate-array, and the building-block custom chip.

We will demonstrate that the placement problem can be transformed to a network optimization problem. Based on the sparsity of the network, an efficient algorithm has been developed

which is capable of solving the constructive placement problem.

The algorithm has been extended to assign components of different width into a row based on the row-type chip design such as the standard-cell and gate-array systems.

While the building-block custom chip system includes irregular sized and shaped components, a reduction process has been developed to tackle the overlapping problem. This process minimizes the chip area, but still maintains the original relative positions among the components. However, it allows rotation and flipping of modules.

Finally, an investigation of the placement problem in terms of different objective functions is explored. In linear placement, the sum of the wiring lengths, the sum of the squared lengths and the number of tracks required are used for comparison. Based on the max-flow min-cut method, an efficient algorithm for linear placement has been developed. It is shown that the method generates the best results in terms of the sum of the wire lengths, whereas the network optimization method achieves the best results in terms of the sum of squared lengths.

## Acknowledgement

I am deeply grateful to my thesis advisor, Prof. Ernest Kuh, for his patience, guidance and support. Working under him has been a most rewarding experience. In addition, I would like to thank Prof. T.C. Hu for his encouragement and advice on linear placement. Prof. A.R. Newton and Prof. W.G. Bade were also very helpful to me.

I am indebted to Dr. Jerry Lee for his encouragement and advice. Dr. B.S. Ting and Dr. B.N Tien of Hughes Aircraft strongly supported this project.

I would also like to thank C.C. Chen, and most particularly Dr. M. Marek-Sadowska, for many interesting and enlightening discussions. Also to be credited on this account are my colleagues T. Tarng, H. Chen, W.M. Dai, X.M Xiong and S.H. Lin. In providing an environment for implementation, J.T. Li and M. Turner worked diligently and with great skill on the BAGEL system. My friend C.L. DeMarco gave me much encouragement and advice throughout.

For their constant encouragement and faith, I am very grateful to my parents. Finally, I am most indebted to my wife, Chuan-Ying, for her continued faith and love.

## Table of Contents

Abstract .....	i
Acknowledgement .....	1
Table of Contents .....	2
1 Introduction .....	5
2 Placement Based on Resistive Network Optimization .....	13
2.1 Introduction .....	13
2.2 Formulation of the approach .....	14
2.2.1 Objective function .....	14
2.2.2 Network analogy .....	16
2.2.3 Boundary constraints .....	17
2.2.4 Slot constraints .....	19
2.3 Proposed method .....	20
2.3.1 Optimization .....	21
2.3.2 Scaling .....	23
2.3.3 Relaxation .....	25
2.3.4 Partitioning and assignment .....	27
2.4 Discussion .....	28
2.4.1 Multi-module nets .....	28
2.4.2 Computation complexity .....	28
2.4.3 Experimental results .....	29
2.5 Conclusion .....	37

	3
<b>3 Gate-Array and Standard-Cell placements .....</b>	<b>39</b>
<b>3.1 Introduction .....</b>	<b>39</b>
<b>3.2 Formulation of the approach .....</b>	<b>42</b>
<b>3.2.1 Objective function .....</b>	<b>42</b>
<b>3.2.2 Network analogy .....</b>	<b>44</b>
<b>3.2.3 Slot constraints .....</b>	<b>47</b>
<b>3.3 Proposed method .....</b>	<b>47</b>
<b>3.3.1 Optimization .....</b>	<b>47</b>
<b>3.3.2 Scaling .....</b>	<b>49</b>
<b>3.3.3 Relaxation .....</b>	<b>50</b>
<b>3.3.4 Partitioning and merging .....</b>	<b>52</b>
<b>3.3.5 Assignment .....</b>	<b>53</b>
<b>3.4 Experimental results .....</b>	<b>56</b>
<b>3.5 Conclusion .....</b>	<b>60</b>
<b>4 Building Block Placement .....</b>	<b>61</b>
<b>4.1 Introduction .....</b>	<b>61</b>
<b>4.2 Formulation .....</b>	<b>62</b>
<b>4.3 Preliminary locations of the modules .....</b>	<b>65</b>
<b>4.4 Basic operations .....</b>	<b>65</b>
<b>4.4.1 Compaction .....</b>	<b>66</b>
<b>4.4.2 Decompaction .....</b>	<b>67</b>
<b>4.4.3 Rotation .....</b>	<b>73</b>
<b>4.4.4 Selection of preferable direction .....</b>	<b>74</b>
<b>4.5 Algorithms .....</b>	<b>75</b>

	4
4.5.1 Spacing .....	75
4.5.2 Reduction .....	76
4.6 Assignment .....	78
4.7 Experiments .....	79
4.8 Conclusion .....	82
5 Linear Placement .....	84
5.1 Introduction .....	84
5.2 Formulation .....	85
5.3 Theory .....	87
5.4 Parallel graph .....	96
5.4.1 Algorithm .....	97
5.4.2 Example .....	98
5.5 Arbitrary graph .....	101
5.5.1 Graph modification .....	101
5.5.2 Algorithm .....	107
5.5.3 Example .....	107
5.5.4 Theorems .....	109
5.6 Application .....	112
5.6.1 Multi-pin nets .....	113
5.6.2 Algorithm .....	116
5.6.3 Discussion .....	117
5.6.4 Experiments .....	117
5.7 Conclusion .....	119
References .....	121

# Chapter 1

## Introduction

### 1.1. IC Layout

With the rapid evolution of VLSI technology, the number of components in a chip becomes large. In view of the growing density and complexity of the chip, physical layout becomes critical to turnaround time of the design and the performance of the circuit. Not only an extra long design period causes the cost of the product unreasonably high, but also an ill-designed layout might be fatal to the whole chip. Thus, an effective and efficient automatic layout is in need for the advancement of the integrated circuit technology.

Due to the tremendous complexity of the problem, layout system is usually decomposed into two phases, namely, placement and routing. The placement assigns the locations of the components on the chip. Based on the result of placement, the routing tries to complete the connections among the components under the constraint of specified design rules. Different placement results generate different connection patterns. Thus, the result of layout is much dependent on the phase of placement.

While there are now many effective and efficient routing algorithms available which have been used extensively, more work must be done in partitioning and placement. This thesis deals with theoretical study of placement and the implementation of algorithms for chip design.

### 1.2. Placement problems

There are currently three basic structures for automatic chip layout.

namely: the standard-cell, the gate-array, and the building block custom chip.

Gate-array and standard-cell approaches have a regular structured chip. The IO pads are placed on the boundary of the chip, while the modules are assigned to an array of rows inside the chip. Modules are assumed to have same height but varying width. The routing area is embedded between the rows and around the periphery of the array. This style significantly simplifies the design process. In gate-array approach, the chips are preprocessed to establish the transistor sites and, often, power and ground metallization as well. The chip size and routing areas are therefore fixed. Thus, achieving 100% routability is one objective for the layout. On the other hand, the standard-cell layout involves the customization of all mask layers. This additional freedom permits variable chip size and adjustable routing space between rows of active areas. Completing the routing with smallest chip area becomes important.

Building block design style is sometimes referred to as the general cell or macrocell approach. It is a popular approach for developing high density, high production- volume integrated circuit chips. Thus the sizes and shapes of the modules are irregular. No geometry or size limitation apart from yield considerations is put on the cell layout. The building block layout is to achieve 100% routability while keeping the chip size as small as possible.

### **1.3. Review of placement**

#### **1.3.1. Objective function**

The placement results are input to the routing phase where all the nets are routed. The size of routing areas are finally determined in the stage of routing. Because of the tremendous complexity of the combined problem, placement and routing are usually separated as independent procedures. In placement,

however, some of this complexity remains: the estimation of the exact sizes of routing areas can become very complicated. Therefore, the goal related to minimizing routing spaces has to be simplified so that the objective function is easy to enumerate on the computer. In the literature, the sum of wire lengths is commonly used as the objective function for placement.

On the other hand we prefer strongly-connected modules to be near each other to improve circuit performance. Otherwise, a long connection wire might cause signal to delay more than a tolerable range. We choose the sum of squared wire lengths as the objective function, since a small sum leads to both the chip size minimization and delay time reduction. With this objective function, the placement problem can be transformed into a network optimization problem.

### **1.3.2. Placement algorithms**

The algorithms for placement can be divided into two categories: constructive placement and iterative improvement of placement.

#### **A. Constructive placement**

One constructive algorithm is the clustering method. Initially, some modules are placed on the chip as seeds. The unplaced modules are then put on the chip sequentially. In each step, the module with the maximal connection to the placed modules is taken from unplaced modules. It is assigned to the position so that the lengths of connections is minimal. This process is repeated until all modules are placed.

Another algorithm is the top-down approach based on bipartitioning. The algorithm used is a minimal cut method such as the Kernighan and Lin's. Given

a cut-line which partitions the chip into two, the modules are iteratively exchanged between the two sides to minimize the number of crossing wires on the cut-line. Then each subregion is partitioned again into smaller subregions. The procedure continues until each subregion contains only one of the prescribed modules.

Both clustering and bipartitioning algorithms are greedy in the sense that they try to achieve the local optimal in each step, but does not view the problem globally. Quinn and Breuer introduced a force-directed method. In their formulation, point modules are assumed, and a force-model is used to determined the state of equilibrium. Hook's law gives the forces of attraction for modules connected by signal nets, and repulsive forces are used to keep modules apart for those which are not connected. This method leads to a good initial placement. However, the algorithm amounts to solving a large set of nonlinear equations , which is time consuming.

### **B. Iterative improvement of placement**

The purpose of this phase is to improve the placement by applying small local changes, such as the pair-wise exchange of modules. Goto, who used the concept of the medium of a module, suggested a multi-way exchange iterative improvement method. This method has better performance than the traditional pair-wise exchange method. But, it often leads to a local minimum solution.

Kirkpatrick introduced a thermal annealing concept to avoid the problem of trapping in the local minimum. The method simulates the physical thermal annealing process, which starts with an appropriately chosen high temperature. Pair of modules are randomly chosen to be exchanged. The difference of objective function is checked with respect to the exchange. According to the tem-

perature, a probabilistic model is set to determine a threshold. The exchange of the pair is accepted if the difference of objective function is smaller than the threshold. The exchanges are repeated and the user gradually decreases the temperature until no more improvements can be obtained. During the process, a large number of different pairs of modules is tested for exchange. Therefore, it consumes a great amount of CPU time.

#### 1.4. Resistive network optimization

As the number of modules becomes larger, a global view of the problem becomes more important. At the same time, the computational complexity of the algorithm should be kept as low as possible. Otherwise, a large amount of CPU running time would prohibit the implementation of the system.

In this thesis, we propose a constructive placement method. First, we simplify the problem by assuming that all modules are of same size and shape. Thus, the model of point module is assumed. Let slots be the fixed locations for modules to be assigned. The objective function is the sum of squared wire lengths. The placement problem is transformed into the problem of minimizing the power dissipation of a resistive network. The constraint of the slots is formulated in terms of a number of polynomial equations. In optimization, the first order constraint is chosen to simplify the problem. This amounts to keeping the center of gravity of modules at the center of the region. With network theory and optimization techniques, the optimal locations of modules are easily obtained.

Since only the linear constraint equation is used in optimization, the placement is more or less confined to the center of the region. Therefore, a scaling and relaxation scheme is proposed to spread the modules so obtained to the

entire region. Then, a partitioning process is used to put the modules into different subregions. The partitioning process continues until each subregion contains only one module.

From several experiments conducted, the method is shown to generate excellent placement results. Since the corresponding network inherits the sparsity of the placement problem, the method is very time efficient with the aid of sparse matrix techniques.

### 1.5. Standard-cell and gate-array placements

The network optimization method is first implemented in standard-cell and gate-array placement. In the partitioning, the sums of the sizes of the modules are checked on the two sides of a dividing line to even out the distribution of the modules over the chip area. After modules are partitioned into rows, a decompaction process is used to separate the overlapping modules.

### 1.6. Building block placement

The network optimization is extended to find the relative locations of modules in building block placement. Modules of rectangular circuit blocks are categorized into two types. The modules which have width or height greater than a certain threshold value are classified as **critical module**. After network optimization, critical modules are placed first. Then, other modules are relocated in the free spaces left by critical modules. A **spacing** algorithm is used to separate the overlapping modules by the operations of compaction, decompaction and rotation. In each iteration, the decompaction operation slides modules back toward the original locations to avoid overlap. Thus, the original relative locations are maintained during the process. The **reduction** algorithm is then applied to reduce the chip size. The algorithm reduces the breadth of chip in

each iteration and calls for repeated use of the spacing algorithm until no more reductions can be made.

### **1.7. Linear placement**

In order to have a fully automatic layout system, the optimum placement ultimately means ease in routing. We need to compare different criteria used in placement. Thus, an investigation of the placement problem in terms of various objective functions is explored. In linear placement, the sum of wiring lengths, the sum of squared lengths and the number of tracks required are used for comparison.

Algorithms for linear placement are developed to minimize the sum of lengths. Two criteria, max-flow min-cut, and a cost ratio, are used to make partitions of the optimal order of the modules. The algorithm decomposes the modules into smaller subregions. This process is repeated until each subregion contains only one module.

It is shown that the linear placement method generates the best results in terms of the sum of the wire lengths, whereas the network optimization method achieves the best results in terms of the sum of squared lengths.

### **1.8. Thesis overview**

The network optimization method is described in Chapter 2. Its extensions to standard-cell and gate-array are discussed in Chapter 3. Chapter 4 deals with the building block placement. The spacing and reduction algorithms which separate the modules and reduce the chip size are described. Finally, Chapter 5 describes the properties of linear placement and introduces two new algorithms. The proposed method and network optimization method are compared with

published results in terms of the sum of wiring lengths, the sum of squared lengths, and the number of tracks required.

All algorithms described have been implemented in C programming language and several examples obtained from industry have been used to test the algorithms. The results are far superior than the manual placements.

## Chapter 2

### Placement Based on Resistive Network Optimization

#### 2.1. Introduction

The force-directed method introduced by Quinn and Breuer is a good constructive placement method which leads to initial placement[1]. In their formulation, point modules are assumed, and a force-model is used to determine the state of equilibrium. Hook's Law gives the forces of attraction for modules connected by signal nets, and repulsive forces are used to keep modules apart for those which are not connected. The algorithm amounts to solving a large set of nonlinear equations, which is time consuming. An improvement has been proposed by Antreich, Johnnes and Kirsch using the same force-directed method but with a more systematic formulation of equations[2].

In this Chapter we propose a more efficient method based on resistive network analogy of the placement problem. The idea of using network analogy to attack layout problems was first introduced by Charney and Plato[3]. They proposed a method of module clustering according to the sensitivity of a network analogy for the purpose of partitioning. In the Chapter, we first solve the optimum placement problem in a systematic way by network analogy. The general formulation of the problem of placing modules on slots involves optimization with nonlinear constraints. However, if only the linear constraints are considered, the problem amounts to solving a linear sparse resistive network. Thus, sparse matrix techniques can be used. Because of its computational efficiency, the procedure is repeated in the overall algorithm of partitioning and module assignment. In the formulation, a key feature is that we allow some modules to be fixed in position. Fixed modules could represent I-O pads; but they also play an important role in solving each optimization problem in succession in the overall algorithm.

In section 2 we give a detailed formulation of our approach to the problem. Section 3 is divided into subsections of optimization, scaling, relaxation, and partitioning and assignment. Section 4 briefly discusses the problem of multi-module nets, the computation complexity and experimental results.

## 2.2. Formulation of the approach

Consider the module placement problem in chip layout. With reference to Fig. 1 where movable modules together with fixed modules represent I-O pads are shown. The movable modules are to be placed on slots where horizontal and vertical lines intersect. The net interconnection specification is given by a net list relating nets and modules. We assume first that all nets are 2-module nets and multi-module nets have been preprocessed and replaced with 2-module nets[4]. Furthermore, all modules are assumed to have zero dimension, thus their shape, size and pin locations are initially ignored.

### 2.2.1. Objective function

Let the two dimensions on the chip be specified by the  $x$  and  $y$  coordinates. Let there be a total of  $n$  modules located at  $(x_i, y_i)$ ,  $i=1,2,\dots,n$ . Let  $c_{ij}$  denote the connectivity between module  $i$  and module  $j$ , i.e., the number of wires between them. Thus  $c_{ii}=0$ . In the literature, the objective function used for placement is usually the sum of wire lengths. However, because of network analogy, we choose an objective function which is the sum of squared wire lengths. Let the objective function be given by:

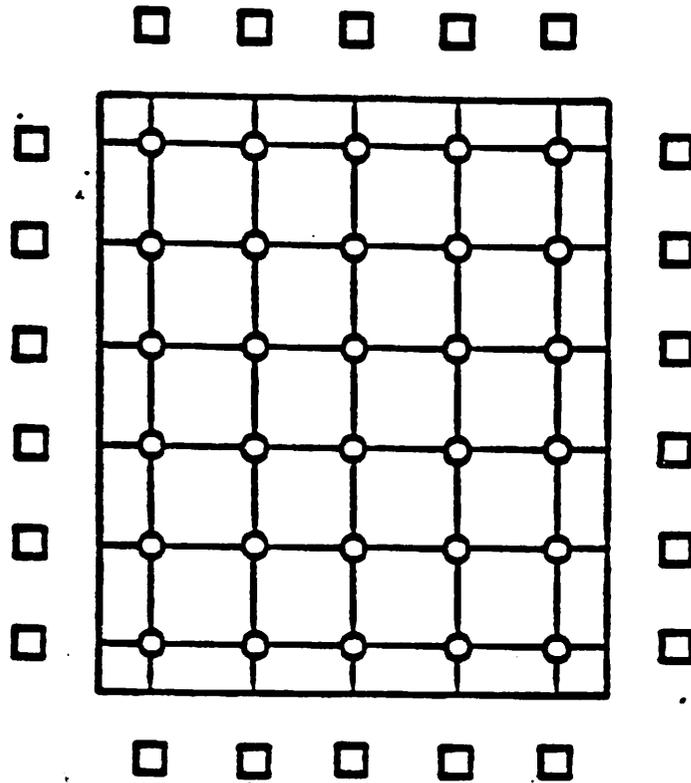


Fig. 2.1. An example with movable modules to be placed on slots within the square and fixed modules on the boundary representing I-O pads.

$$\Phi(x, y) = \frac{1}{2} \sum_{i,j=1}^n c_{ij} L_{ij}^2 = \frac{1}{2} \sum_{i,j=1}^n c_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2] \quad (1)$$

where  $L_{ij}$  is the Euclidean distance between module  $i$  and module  $j$ . Equation (1) can be written as follows[5]:

$$\begin{aligned} \Phi(x, y) &= \frac{1}{2} \sum_{i,j=1}^n c_{ij} [x_i^2 + x_j^2 - 2x_i x_j + y_i^2 + y_j^2 - 2y_i y_j] \\ &= \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_i^2 - \sum_{i=1, i \neq j}^n \sum_{j=1}^n c_{ij} x_i x_j + \sum_{i=1}^n \sum_{j=1}^n c_{ij} y_i^2 - \sum_{i=1, i \neq j}^n \sum_{j=1}^n c_{ij} y_i y_j \\ &= x^T Bx + y^T By \end{aligned} \quad (2)$$

where

$$B = D - C \quad (3)$$

is an  $n \times n$  symmetric matrix,  $C = [c_{ij}]$  is the connectivity matrix and  $D$  is a diagonal matrix whose  $i$ -th element  $d_{ii}$  is equal to  $\sum_{j=1}^n c_{ij}$ .

With the symmetry between  $x$  and  $y$  in Eq. (1), we need to consider only the one-dimension problem insofar as optimization is concerned. Thus we dispense with the  $y$  coordinate until the end of Sec. 3 where we discuss partitioning and assignment.

### 2.2.2. Network analogy

For those who are familiar with circuit theory,  $B$  in Eq. (3) is seen to be of the same form as the indefinite admittance matrix of an  $n$ -terminal linear passive resistive network. We will model the coordinate of module  $i$ ,  $x_i$  with a node voltage  $v_i$  at node  $i$ . The reference coordinate  $x=0$  is thus the datum voltage. The term  $-c_{ij}$  in Eq. (3) is then the mutual admittance between node  $i$  and node  $j$ , and  $d_{ii} = \sum_{j=1}^n c_{ij}$  is the self admittance at node  $i$ .

The power dissipation in the resistive network is

$$P = v^T Y_n v \quad (4)$$

where  $v$  is an  $n$ -vector representing the node voltage vector and  $Y_n$  is the indefinite admittance matrix which is symmetric. Thus the objective function of the placement problem becomes the power dissipation in the linear passive resistive network. It is well-known that in a passive resistive network the current distributes itself in such a way that the power is minimum[6]. That is, any other current distributions which are not the solution of the network would have a larger power dissipation. In other words, the problem of solving network equations is equivalent to that of minimizing a well-selected function which

represents power.

### 2.2.3. Boundary constraints

Consider the  $n$ -terminal resistive network shown in Fig. 2.2. The first  $m$  nodes are floating and their voltages are denoted by an  $m$ -vector  $v_1$ . The remaining  $n-m$  nodes are connected to voltage sources denoted by an  $(n-m)$ -vector  $v_2$ . Thus the coordinates of the  $n$  modules are represented by an  $n$ -vector  $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$  where the coordinates of the fixed modules are specified by  $v_2$  and the coordinates of the movable modules which are to be determined are represented by  $v_1$ .

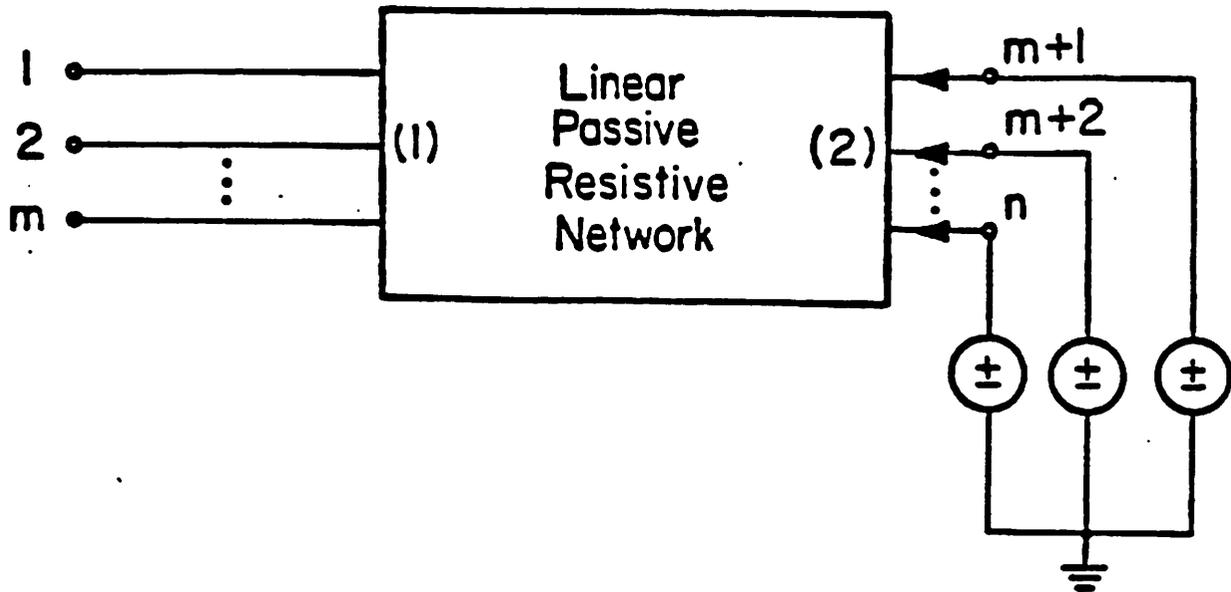


Fig. 2.2. An  $n$ -terminal linear, passive resistive network whose first  $m$  nodes are floating and the remaining  $n-m$  nodes are connected to voltage sources.

The network equations are:

$$0 = y_{11}v_1 + y_{12}v_2 \quad (5a)$$

$$i_2 = y_{21}v_1 + y_{22}v_2 \quad (5b)$$

where  $y_{11}$ ,  $y_{12}=y_{21}^T$  and  $y_{22}$  are the familiar short-circuit admittance submatrices of the indefinite admittance matrix,  $Y_n$ . From (4b), we obtain

$$v_1 = -y_{11}^{-1}y_{12}v_2 \quad (6)$$

which gives the solution of the movable modules in terms of the fixed modules and the admittance submatrices.

### Remarks

- (1)  $y_{11}$  is the short circuit driving-point admittance submatrix of a passive resistive network and is positive definite; thus  $y_{11}^{-1}$  always exists.
- (2) The solution of Eq. (6) must fall inside the region defined by the smallest and largest voltages of the voltage sources. This is because in a passive resistive network, node voltage can not lie outside the range of voltage sources[6].
- (3) The dissipated power obtained from the solution in Eq. (6) is the minimum among all possible  $v_1$ . Any deviation from the solution will result in an increase in power.

### 2.2.4. Slot constraints

Up to now we have not imposed the constraint that the movable modules must be located on slots. This means that the voltage vector  $v_1$  when finally determined must represent a set of prescribed discrete voltages called the **legal values**. Let us designate the prescribed slots in terms of the permutation vector  $p = [p_1, p_2, \dots, p_m]^T$  where  $p_i$  is the  $i$ -th legal value and  $m$  is the total number of the movable modules. Thus the permutation of the  $m$  legal values must be assigned to the  $m$  modules of  $v_2$ . To express this in terms of our optimization problem, let  $v_1 = [x_1, x_2, \dots, x_m]^T$ , i.e.,  $x_i$  denotes the coordinate of module  $i$  or the voltage at node  $i$ . We claim that the following set of equations represents the constraints on the modules which are required to be on slots:

$$\begin{aligned}
 \sum_{i=1}^m x_i &= \sum_{i=1}^m p_i \\
 \sum_{i=1}^m x_i^2 &= \sum_{i=1}^m p_i^2 \\
 &\vdots \\
 \sum_{i=1}^m x_i^m &= \sum_{i=1}^m p_i^m
 \end{aligned} \tag{7}$$

The first equation can be written as

$$1^T v_1 = 1^T p \equiv d \quad (8)$$

where  $1$  is a unit vector and  $d$  is a constant which is equal to the sum of the  $m$  legal values.

**Proof:**

$\Rightarrow$  Let  $[x_1, x_2, \dots, x_m]$  equal to any permutation of  $[p_1, p_2, \dots, p_m]$ . Eq. (7) is automatically satisfied.

$\Leftarrow$  Let us define

$$f(x) = \prod_{i=1}^m (x + x_i)$$

Then the coefficients of the variable  $x$  are multi-variable polynomials of  $[x_1, x_2, \dots, x_m]$ . Through simple algebraic operations[7] and by using Eq. (7), we can show that

$$f(x) = \prod_{i=1}^m (x + p_i)$$

which implies that all modules are on slots.

**Q.E.D.**

### 2.3. Proposed method

The proposed method can be divided into subproblems of optimization, scaling, relaxation, and partitioning and assignment. The main idea is to solve a simple optimization problem using linear resistive network analogy repeatedly and, in the process, the movable modules are assigned to slots. We shall use node voltages and module coordinates interchangeably in the ensuing discussion, for sometimes it is more intuitive to make statements in terms of voltages; while in dealing with the actual placement problem it is more convenient to use the coordinates.

### 2.3.1. Optimization

From Eqs. (4) and (5), we wish to minimize the power dissipation

$$P = \mathbf{v}^T \mathbf{Y}_n \mathbf{v} = \begin{bmatrix} v_1^T & v_2^T \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = v_1^T y_{11} v_1 + 2v_1^T y_{12} v_2 + v_2^T y_{22} v_2 \quad (9)$$

subject to the complete set of constraint equations in Eq. (7). This is clearly not feasible. Therefore, we propose to use only the first equation in Eq. (7), which is a linear constraint expressed by Eq. (8).

The solution to the optimization problem of minimizing  $P$  in Eq. (9) subject to the linear constraint in Eq. (8) is derived from the well-known Kuhn-Tucker conditions. The first order necessary conditions, in addition to the constraints are

$$2y_{11}v_1 + 2y_{12}v_2 + \lambda \mathbf{1} = 0 \quad (10a)$$

$$\mathbf{1}^T v_1 = d \quad (10b)$$

where  $\lambda$  is the so called Lagrange multiplier. The solution of the above simultaneous equations is

$$v_1 = y_{11}^{-1} \left[ -y_{22}v_2 + i_1 \right] \quad (11a)$$

where

$$i_1 \equiv \frac{d + \mathbf{1}^T y_{11}^{-1} y_{12} v_2}{\mathbf{1}^T y_{11}^{-1} \mathbf{1}} \quad (11b)$$

With respect to the second order conditions, we find the Hessian matrix is equal to  $2y_{11}$  which is positive definite. Thus the solution is optimal.

It is seen that the first term in Eq. (11a) is precisely that given by Eq. (6) for which there is no constraint on slots. The second term of Eq. (11a) can be viewed as a correction term which attempts to put the solution on slots. In terms of electric network, we may use current sources to interpret the effect as shown in Fig. 2.3. Thus we have a linear resistive network with both voltage and current sources. In addition, we know that the network is sparse because of the

inherent nature of the placement problem. Using well-known sparse matrix algorithms, we can greatly reduce the computation time in comparison with those that use attraction and repulsion forces[1,2].

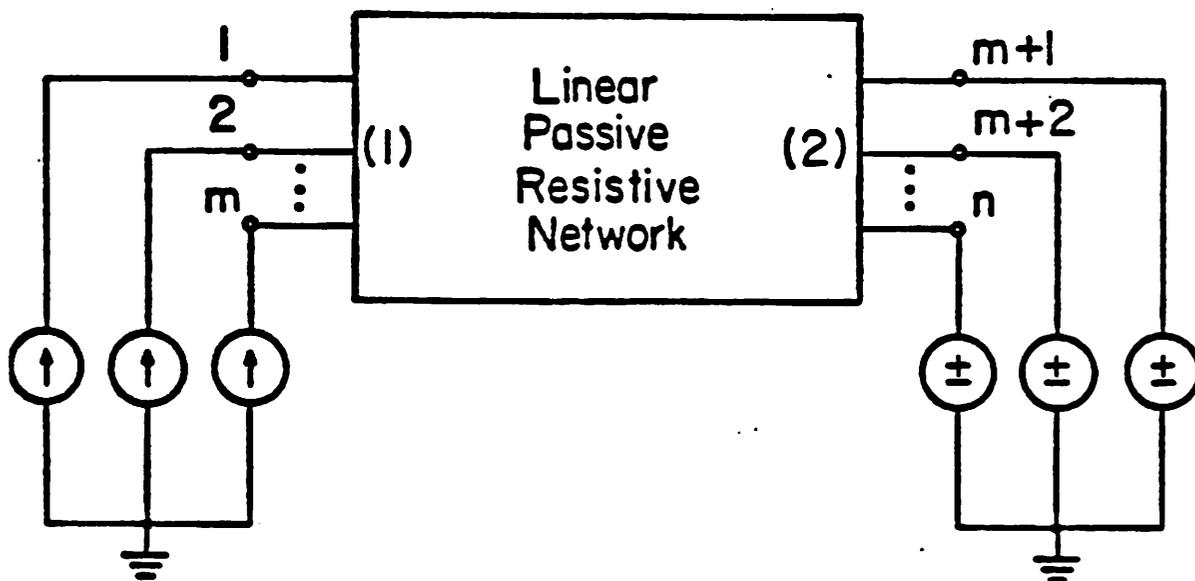


Fig. 2.3. Network interpretation of the optimization problem with linear constraints.

As mentioned previously, because only the linear constraint equation is used, the solution will not put modules on slots. As a matter of fact the result will lead to modules more or less confined to the center of the region. Therefore we must introduce ways to spread the modules so obtained apart and then to bring them to the legal positions. Thus the next step in our overall method is scaling which will distribute the solution more evenly over the entire region. However, let us first analyze the effect of module movements to changes in power dissipation. Let us assume that we deviate away from the solution  $v_1$  of

Eq. (11) by  $\delta v_1$  under the constraint of Eq. (8), i.e.

$$1^T \delta v_1 = 0 \quad (12)$$

We claim that the power dissipation is increased by

$$\delta v^T y_{11} \delta v_1.$$

**Proof:**

From equation (9), we have

$$\Delta P = P(v_1 + \delta v_1) - P(v_1) = 2\delta v^T y_{11} v_1 + \delta v^T y_{11} \delta v_1 + 2\delta v^T y_{12} v_2$$

From equation (11),

$$y_{12} v_2 = -y_{11} v_1 + i_1$$

and using Eq. (12), we obtain

$$\Delta P = \delta v^T y_{11} \delta v_1$$

**Q.E.D.**

Furthermore, it is possible to derive an upper bound on the increase in power dissipation in terms of  $y_M$ , the largest diagonal element in  $y_{11}$ . From the Theorem of Gerschgorin[8], we know that the eigenvalues of  $y_{11}$  are not larger than  $2y_M$ , then

$$\Delta P = \delta v^T y_{11} \delta v_1 \leq |y_{11}| |\delta v_1|^2 \leq 2y_M \sum_{i=1}^m \delta v_i^2 \quad (13)$$

Therefore the increase in power dissipation has an upper bound which is proportional to the norm of the deviation  $\delta v_1$ .

### 2.3.2. Scaling

The result of the optimization with linear constraint leads to solutions which have movable modules concentrated at the center of gravity of all modules. The linear constraint dictates the mean position of the modules. The only forces which attempt to scatter the modules are the fixed modules at the boundary. Therefore, in order to be able to partition the modules we will introduce scaling

to redistribute the modules at the expense of increasing the power dissipation. The method used here is to minimize the increase of power  $\Delta P$  under the constraints which include both the first order and second order equations in Eq. (7). Fortunately, by using the norm of  $\delta v_1$  in Eq. (13), we again can resort to the well-known Kuhn-Tucker conditions.

Let us assume that in the region where there are  $k$  modules the legal values are given by the permutation vector  $[p_1, p_2, \dots, p_k]$ . Let  $x_o = [x_{o1}, x_{o2}, \dots, x_{ok}]$  denote the solution obtained from optimization and let  $x_n = [x_{n1}, x_{n2}, \dots, x_{nk}]$  denote the new solution after scaling. Thus our problem is to minimize

$$\sum_{i=1}^k (x_{ni} - x_{oi})^2 \quad (14)$$

under the constraints

$$\sum_{i=1}^k x_{ni} = \sum_{i=1}^k p_i \quad (15)$$

and

$$\sum_{i=1}^k x_{ni}^2 = \sum_{i=1}^k p_i^2 \quad (16)$$

The solution is derived from the Kuhn-Tucker conditions. The first order necessary condition is

$$2(x_n - x_o) + \lambda_1 1 + 2\lambda_2 x_n = 0 \quad (17)$$

In addition to the constraints of Eq. (15) and Eq. (16), the solution is

For  $i=1, 2, \dots, k$

$$x_{ni} = \frac{x_{oi} - c_o}{a_o} a_n + c_n \quad (18)$$

where

$$c_n = \frac{1}{k} \sum_{i=1}^k p_i \quad (19)$$

$$a_n = \left[ \frac{1}{k} \sum_{i=1}^k (p_i - c_n)^2 \right]^{\frac{1}{2}} \quad (20)$$

$$c_o = \frac{1}{k} \sum_{i=1}^k x_{oi} \quad (21)$$

and

$$a_o = \left[ \frac{1}{k} \sum_{i=1}^k (x_{oi} - c_o)^2 \right]^{\frac{1}{2}} \quad (22)$$

Thus  $c_o$  is the mean position of the computed module positions and  $a_o$  is the root mean square amplitude measured from  $c_o$ .

With respect to the Kuhn-Tucker second order conditions, we find the Hessian matrix is equal to  $2 \frac{a_o}{a_n} I$  where  $I$  is an identity matrix. Where  $a_o > 0$ , the Hessian matrix is positive definite. The solution is optimal. If  $a_o$  turns out to be very small which approaches zero, so does  $x_{oi} - c_o$  in Eq. (18); then Eq. (18) must be replaced by

$$x_{ni} = c_n \quad (23)$$

After scaling the norm of deviation becomes

$$\sum_{i=1}^k (x_{ni} - x_{oi})^2 = k \left[ (a_n - a_o)^2 + (c_n - c_o)^2 \right] \quad (24)$$

The result of scaling represents an improvement from the result of optimization as far as module location is concerned, but it gives an increase in power dissipation.

### 2.3.3. Relaxation

Before undertaking partitioning and assigning of modules to slots, we need to perform relaxation to be described below. This will greatly improve the preliminary results from optimization and scaling. The method calls for repeated use of scaling and optimization over subregions to be specified by designers. This tends to spread the modules out more evenly over the entire region. It is important to note that when a pertinent subregion is considered, modules outside are always kept fixed.

We propose to choose subregions in the following way: First we start from one end of the region, then the other end and, finally, the middle. After the initial optimization over the entire region, three such steps of scaling and optimization over subregions are carried out. The result tends to settle down and is ready for partitioning. Thus we have as

Input:

A one-dimensional region with coordinates of movable modules  $x_i$ ,  $i=1,2,\dots,m$  obtained from initial optimization in the entire region with specified fixed modules  $x_i$ ,  $i=m+1,m+2,\dots,n$  on the boundary. A parameter  $\beta$  is to be chosen by the designer with  $0 < \beta < 50\%$ .

Relaxation:

- (1) Order the modules from left to right according to coordinates with the smallest one first.
- (2) Choose  $\lceil \beta m \rceil$  modules from the left, let other modules be fixed and do scaling in the left  $\beta$  region.
- (3) Fix the modules so determined in the left  $\beta$  region and release the modules in the right  $(1-\beta)$  region. Do optimization.
- (4) Choose  $\lfloor \beta m \rfloor$  modules from the right, let other modules be fixed and do scaling in the right  $\beta$  region.
- (5) Fix the modules in the right  $\beta$  region and release the modules in the left  $(1-\beta)$  region. Do optimization.
- (6) Choose  $\lceil \beta m \rceil$  modules from the left, let other modules be fixed and do scaling in the left  $\beta$  region.
- (7) Set modules in both the left  $\beta$  region and the right  $\beta$  region fixed and release the modules in the center subregion. Do optimization.

$\lceil k \rceil$  means the smallest integer which is larger than  $k$ .

Output:

A one-dimensional region with  $m$  modules and new coordinates  $x_i$ ,  
 $i=1,2,\dots,m$ .

#### 2.3.4. Partitioning and Assignment

We next partition the region into two. The ratio of the left subregion to the right subregion is  $\lfloor m/2 \rfloor / \lceil m/2 \rceil$  where  $\lfloor k \rfloor$  denotes the largest integer which is smaller than  $k$ . We do scaling once more for the left subregion and then for the right subregion. As before, in scaling for a subregion we always keep those modules outside fixed. The result of this gives two partitioned subregions together with their associated modules. We next repeat the process for each subregion, i.e., perform independently for each subregion, optimization, relaxation and partitioning.

In the following we will reinstate the  $y$  coordinate to summarize the 2-dimensional partitioning and assignment problem.

Input:

A 2-dimensional region to be partitioned into rectangles each containing a module, a set of  $m$  movable modules together with their coordinates and a set of  $n-m$  fixed modules.

Assignment:

- (1) Do optimization on both the  $x$  coordinate and the  $y$  coordinate of the movable modules.
- (2) While each region contains more than one module

Do

Choose the direction of the cut-line.

Cut the longer side of region.

List all current regions.

For each region do partitioning.

- (3) For each region, assign the module to the legal value.

## 2.4. Discussion

### 2.4.1. Multi-module nets

As mentioned in the introduction section, we assume that all nets are 2-module nets in our treatment. Since multi-module nets are always present, we use the following two models to deal with them:

- (1) At the beginning we use a clique to simulate a multi-module net. If there are  $r$  modules in a net, the weight of each edge on the clique is  $2/r$ .
- (2) After the relative module position is determined, we use a chain to connect the modules. Consider the  $x$  direction, we order the modules according to their coordinates; we then link the modules by a chain in this order.

In the experiments to be discussed in 4.3, the lengths of wires are measured according to the model in (2) in  $x$  and  $y$  directions, respectively.

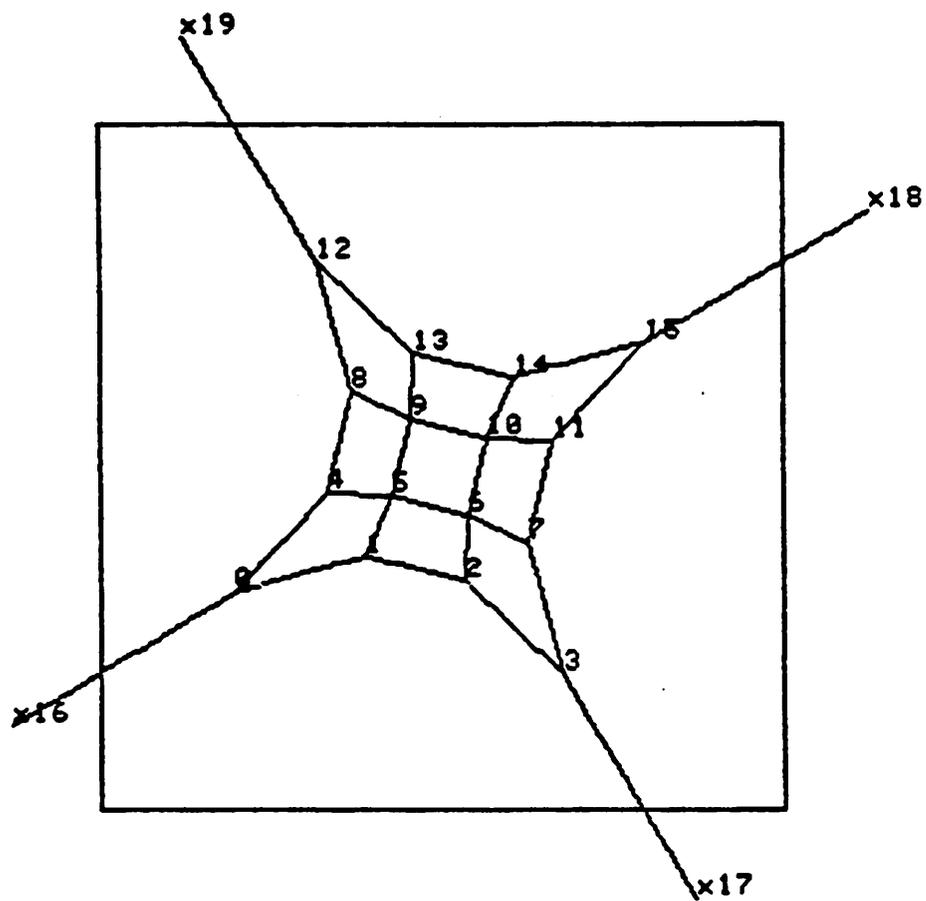
### 2.4.2. Computation complexity

The optimization algorithm amounts to a linear resistive network computation. Using sparse matrix technique, we have the computation complexity  $O(m^{1.4})$  where  $m$  is the number of movable modules. The scaling operation is linear with  $k$  where  $k$  is the number of modules in a subregion.

As to partitioning and assignment, in each iteration, all current regions are divided into two subregions. It takes  $\log_2 n$  iterations to make all the necessary divisions. Thus the total complexity is  $O(n^{1.4} \log_2 n)$ .

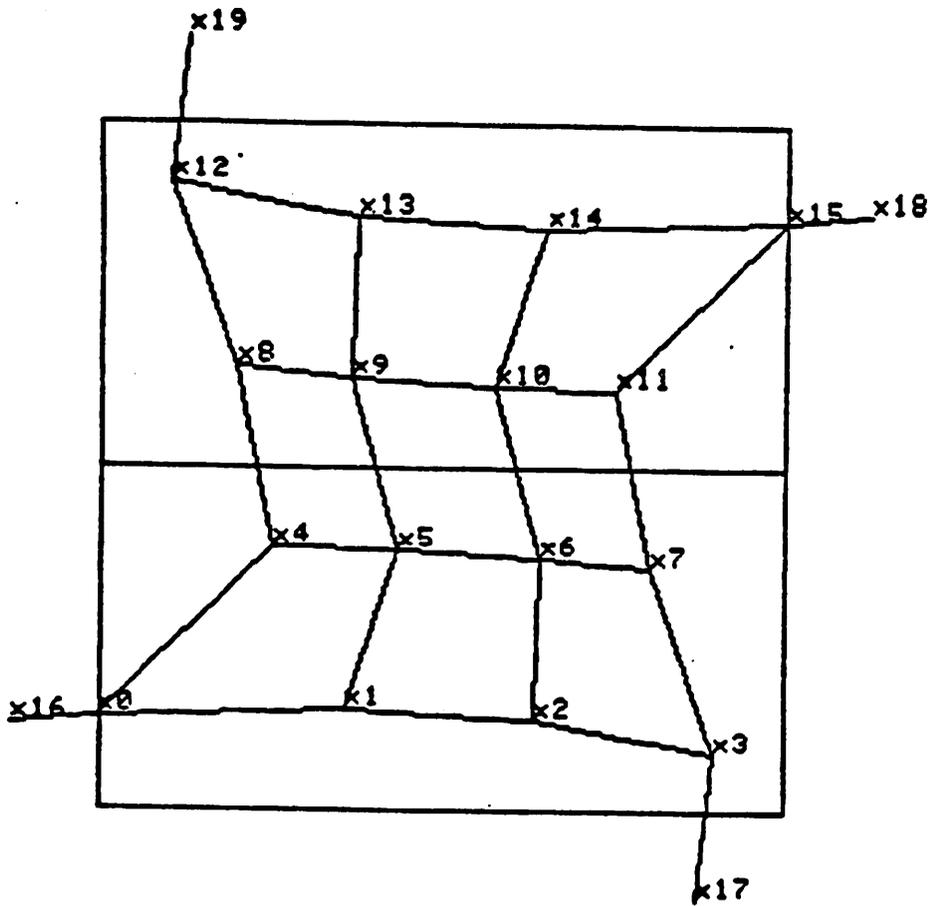
### 2.4.3. Experimental results

A 20 module example is designed to illustrate the procedure of our algorithm. As shown in Fig. 2.7, four modules are fixed on the four corners of the chip, and in the optimal placement every module is connected to the neighboring modules only. Figs. 2.4~2.7 illustrate how module locations evolve from the initial placement onto slots. In the figures, module positions are indicated by points with module numbers. The connectivity among modules is represented by linking lines.



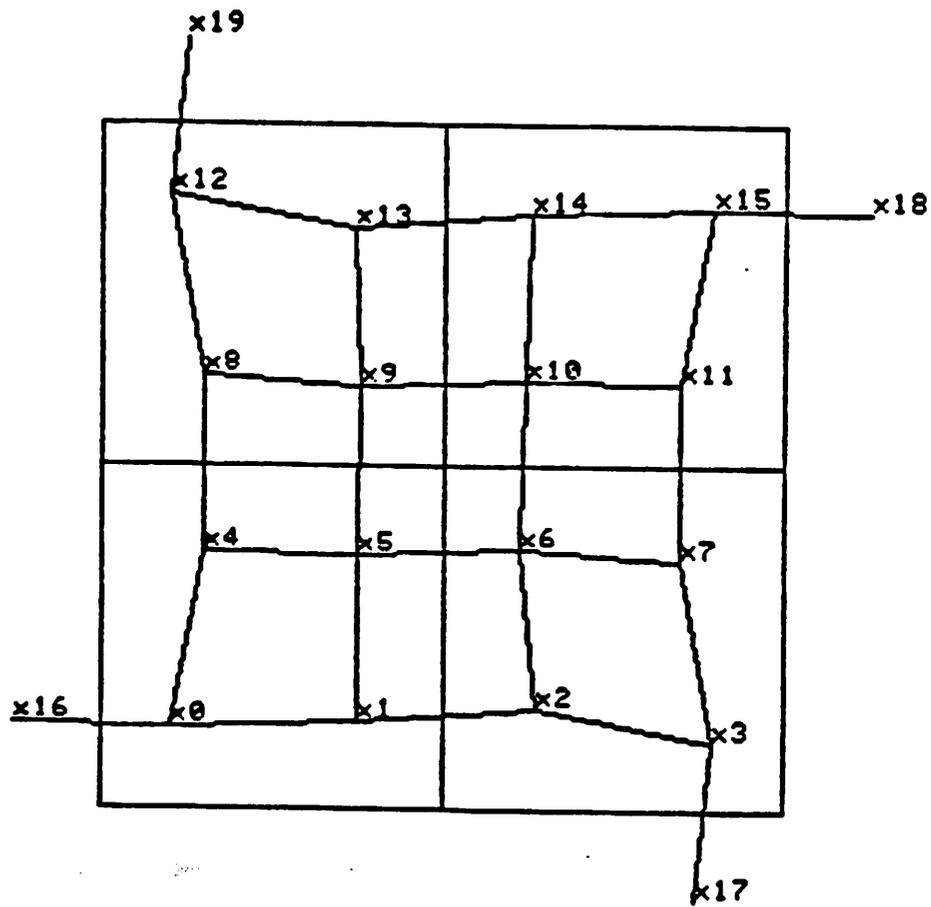
squared length = 17.629627  
manhattan length = 25.4815

Fig. 2.4. Result of assignment step (1) of the 20 module example. The module positions are optimal under the constraint that the center of gravity of the modules is at the center of the chip.



squared length = 28.721426  
manhattan length = 31.8012

Fig. 2.5. Result of first level partitioning and scaling after relaxation is carried out in the vertical direction.



squared length = 27.540583  
manhattan length = 29.6487

Fig. 2.6. Result of second level partitioning and scaling.

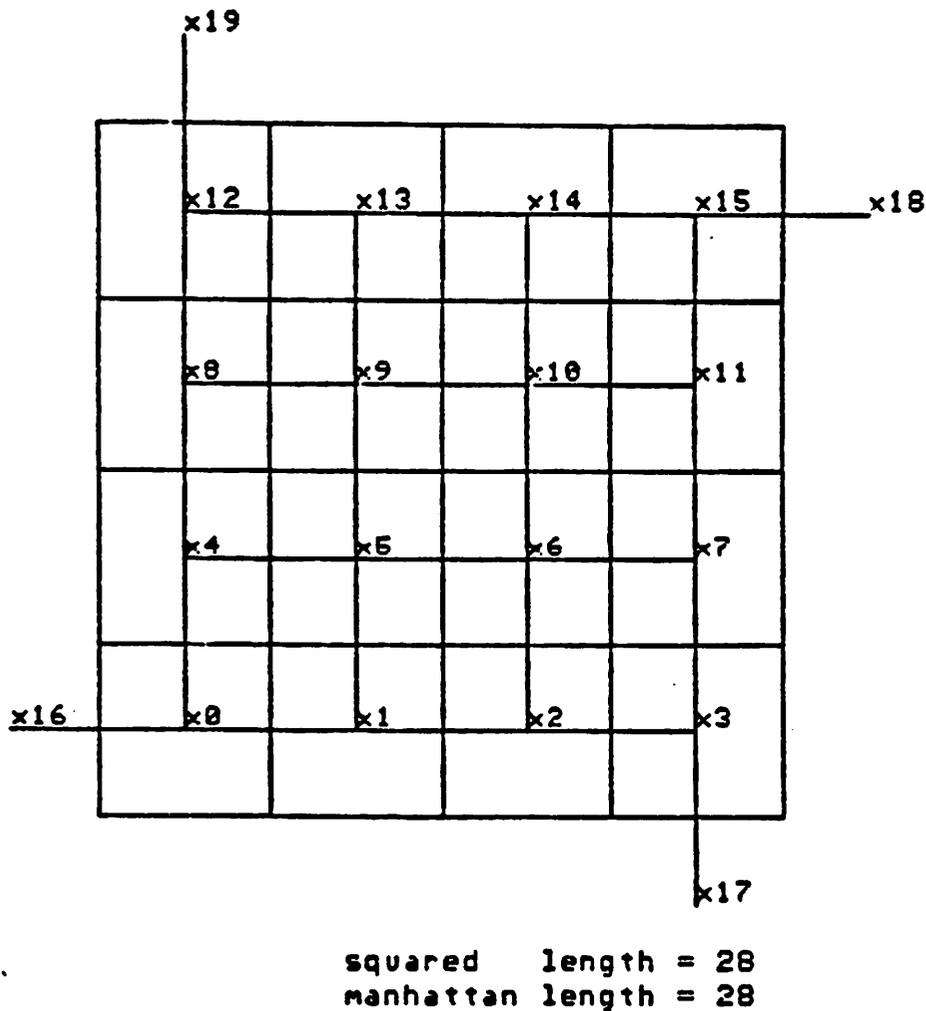


Fig. 2.7. A 20-module placement problem with 4 fixed modules specified.

Fig. 2.4 is the result of initial optimization. The module positions are optimal under the constraint that the center of gravity of the modules is at the center of the chip. Relaxation is next carried out and the modules spread out over the entire region in the vertical direction. Next partitioning and scaling are used to relocate the modules into two subregions as shown in Fig. 2.5. Fig. 2.6 is the result of second level of optimization, relaxation and partitioning using a vertical cut-line; hence, module spread out in the horizontal direction. Fig. 2.7

is the solution of the final assignment. It is seen that all modules are located on slots.

To evaluate the effectiveness of our method, we use the example given by Steinberg[5,9]. However, because we always assume that there exist fixed modules in our formulation, we modified Steinberg's example by fixing the position of the two modules (34 and 26 shown in Fig. 2.9) in the bottom row. In relaxation, we tried different values of  $\beta$  to compare the results. These are shown in Fig. 2.8 where we plot the sum of the squared length for different values of  $\beta$ . It is clear that  $\beta=0$  implies no relaxation. The placement for  $\beta=0.125$  which leads to the smallest squared wire length is shown in Fig. 2.9. This 34 modules, 172 nets example took 13.1 seconds of cpu time and 169K memory on VAX 11/780 machine. For comparison with Steinberg and Hall, we also calculated the sum of the Manhattan length and the sum of Euclidean length. The Table 2.1 summarizes the comparison.

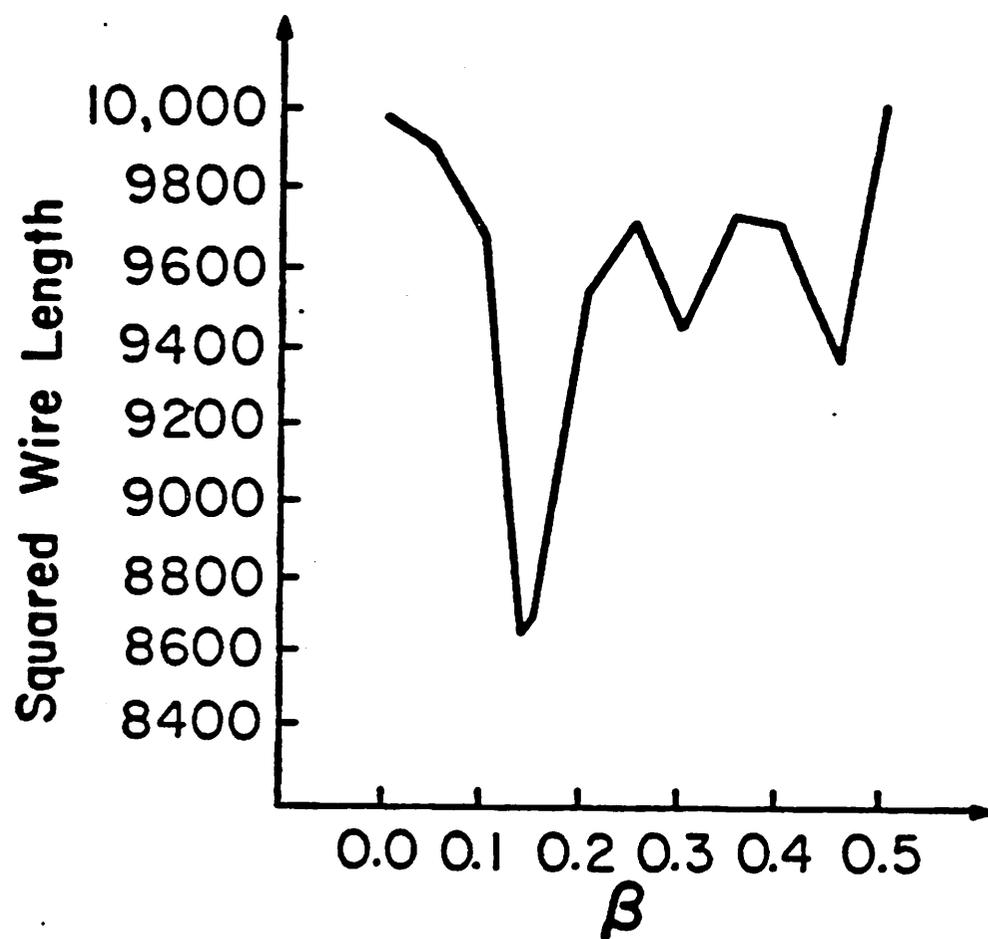


Fig. 2.8. Result on Steinberg's example, with different values of  $\beta$  in relaxation.

	9	2	
16	8	3	17
10	4	18	5
1	7	13	6
15	20	11	12
28	19	14	27
32	29	23	21
33	30	22	25
34	31	24	26

Squared length 8596  
Manhattan length 5316  
Euclidean length 4358.36

Fig. 2.9. Result on Steinberg's example, with  $\beta$  equal to 0.125.

Measure	Steinberg	Hall	Cheng-Kuh
Squared length	11875	9699	8596
Manhattan length	N.A.	5139	5316
Euclidean length	4894.54	4419.13	4358.36

Table2.1. Example 1: Steinberg's example

As a second example we use the ILLIAC IV PC Board problem given by Stevens[10]. Again we fix the IO Pads according to the placement result of Quinn and Breuer[1]. The result with  $\beta=0.25$  is given in the Table 2.2 together with those of Stevens and, Quinn and Breuer. This 136 modules, 432 nets example took 104.2 seconds of cpu time and 480K memory on VAX 11/780 machine.

Measure	Stevens	Quinn and Breuer	Cheng-Kuh
Squared length	N.A.	8794	7521
Manhattan length	2733	2558	2495

Table2.2. Example 2: Placement of ILLIAC IV Board IC136.

In both examples it is seen that in terms of our chosen objective function, i.e., the sum of squared length, our method yielded the best results by far.

## 2.5. Conclusion

The module placement problem has been formulated in terms of linear resistive network optimization. The objective function used is the sum of

squared wire length which corresponds to power dissipation in the resistive network. Fixed modules become nodes with constant voltages. Movable modules then correspond to nodes whose voltages are to be determined. Since modules must be put on slots, a set of constraint equations are imposed on the modules. We consider only the first order linear constraint which, in essence, fixes the center of gravity of the movable modules. The optimization calculation can thus take advantage of the sparse matrix technique, and is repeated in the over-all algorithm. To assign modules to slots, we need to perform scaling, relaxation, partitioning and assignment. These comprise the over-all algorithm.

We have tried our method on well-known examples and compared our results with other methods. So far, we always obtain the least squared wire length as we expected. The extensions to gate-array , standard-cell, and building block designs will be discussed in the next two Chapters.

## Chapter 3

### Gate-Array and Standard-Cell Placements

#### 3.1. Introduction

Gate-array and standard-cell designs are now widely used for automatic layout of VLSI circuits. In these approaches, the designs of basic circuit configurations (modules) are stored in a library. A typical chip structure is shown in Fig. 3.1. The IO pads are placed on the boundary of the chip, while the modules are assigned to rows inside the chip. Modules are assumed to have same height but varying width. The rectangular areas between rows are used for routing. This style significantly simplifies the design process. Often there are hundreds of modules on a chip. In such systems, automatic placement plays the important role of ensuring all modules are at their legal locations and furthermore achieving 100% routability.

In gate-array approach, rows of transistors are preprocessed. The chip size and routing areas are therefore fixed. In standard-cell, chip size and routing areas are adjustable. Thus, routing is different for these two systems. However, the formulation of placement problem is similar. Consequently, our placement algorithm is developed for both gate-array and standard-cell designs.

In this chapter, we extend the network optimization method to gate-array and standard-cell placement problems. The placement combined with a global routing system[11] and Yoshimura and Kuh's channel router[12] represents the major components of the Berkeley Automatic Gate-Array Layout System (Bagel)[13].

Since the modules have varying width, we have to consider the size of the modules in distributing them over the chip. Also, since pin locations can be far from the center of the module, we must modify our model of point modules pre-

viously used. Finally, in the assignment algorithm, we introduce a decompaction process to separate overlapping modules.

In section 2 we give a detailed formulation of our approach to the problem. Section 3 describes the extension of the network optimization method in gate-array and standard-cell approaches. Decompaction is also described in the assignment algorithm. Finally, we demonstrate the test results on 2K and 4K gate-array examples.

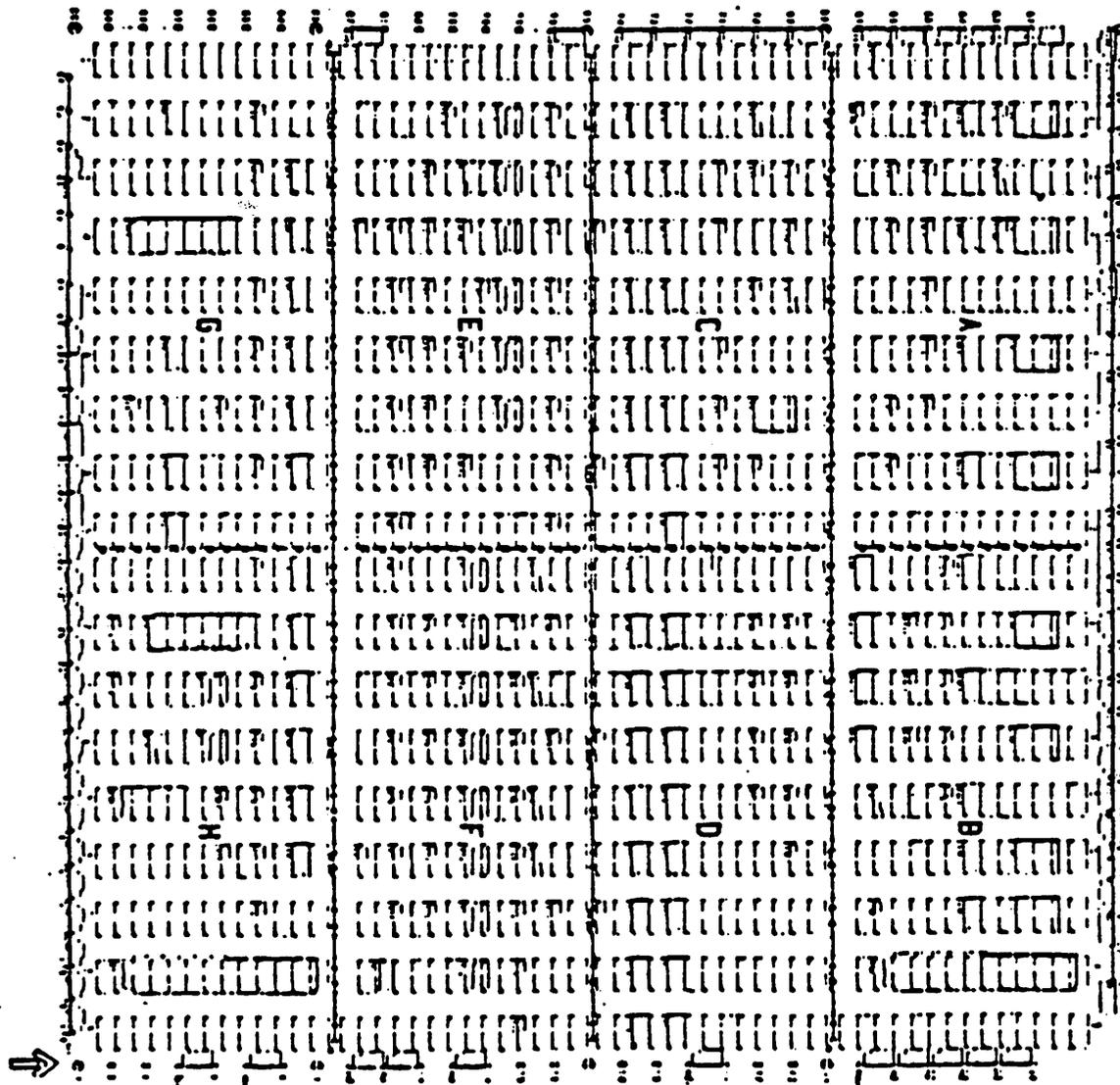
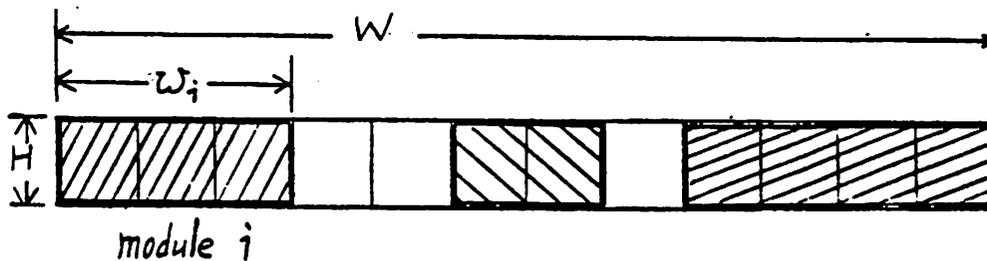


Fig. 3.1. An example with modules to be placed on rows within the chip and fixed IO pads on the boundary.

### 3.2. Formulation of the approach

We formulate the placement problem in gate-array and standard-cell chips layout. Refer to Fig. 3.1 where array of rows together with slots on the boundary for external I-O pads are shown. Given a set of modules, the modules have different width, but the height of modules is set equal to the height of rows of active areas. There are pins fixed on modules for the purpose of connection. Let  $N$  be a set of nets describing the connection of pins and I-O pads. Placement assigns modules on rows of active areas with the constraint that no module can overlap each other. For each row, we denote the capacity to be the area available for modules as shown in Fig. 3.2. We denote the size of the module to be the area occupied by this module. As usual, we replace all multi-pin nets with two-pin nets in the following formulation.



$$\text{capacity} = W \times H$$

$$\text{size of module } i = w_i \times H$$

Fig. 3.2. Illustration of the capacity of each row and the size of the module.

#### 3.2.1. Objective function

We choose the sum of squared wire lengths to be the objective function. Since the pin locations are considered, we formulate the function in terms of the pin coordinate  $(P_x, P_y)$  where  $i$  is the index of pin. Let  $c_{ij}$  denote the connectivity between pin  $i$  and pin  $j$ . The objective function can be written as:

$$\frac{1}{2} \sum_{(i,j) \in N} c_{ij} l_{ij}^2 = \frac{1}{2} \sum_{(i,j) \in N} c_{ij} \left[ (Px_i - Px_j)^2 + (Py_i - Py_j)^2 \right] \quad (1)$$

where  $(i,j)$  is a pair of pin indices, and  $l_{i,j}$  is the Euclidean distance between pin  $i$  and pin  $j$ .

However, the locations of pins in Eq. (1) are constrained with respect to the locations of modules to which the pins belong (Fig. 3.3). Let  $m(i)$  be the index of the module to which pin  $i$  belongs. In Fig. 3.3, the relative coordinate of pin  $i$  with respect to the center of the module is  $(dx_i, dy_i)$ . The pin location can then be formulated by the coordinates of module center,  $(x_{m(i)}, y_{m(i)})$ .

For each pin  $i$ ,

$$\begin{aligned} Px_i &= x_{m(i)} + Ox_{m(i)} * dx_i \\ Py_i &= y_{m(i)} + Oy_{m(i)} * dy_i \end{aligned} \quad (2)$$

In gate-array and standard-cell placements, the modules are not allowed to rotate by ninety degrees. They can only be reflected with respect to the  $x$  or  $y$  axis. In the above equation,  $(Ox_{m(i)}, Oy_{m(i)})$  denotes the reflection status of module  $m(i)$  in the  $x$  and  $y$  axes. The value of  $-1$  indicating reflection and the value of  $+1$  indicating no reflection.

The above equation indicates that the objective function in Eq. (1) can be formulated in terms of the coordinates of the module centers. Plugging Eq. (2) into Eq. (1), we see that the objective function is then formulated in terms of the coordinates of the module centers.

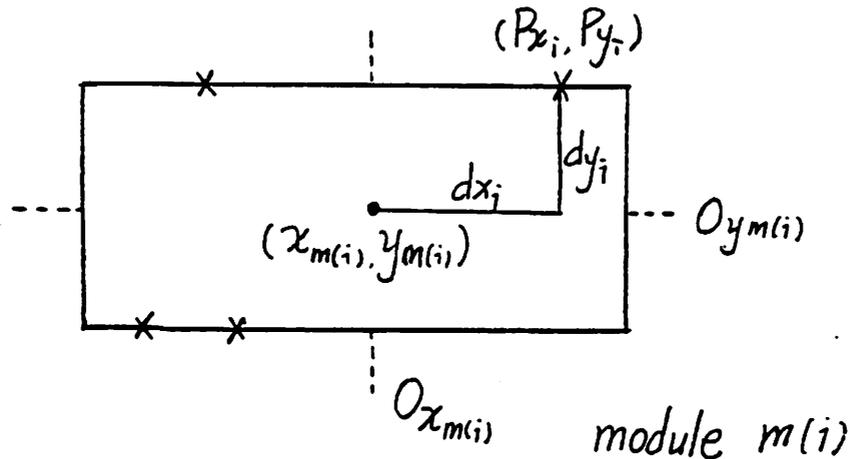


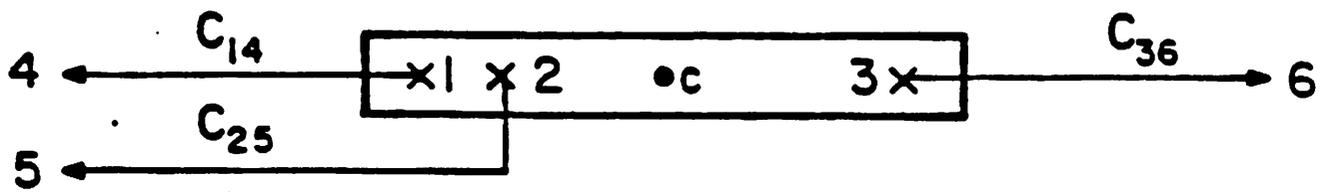
Fig. 3.3. Illustration of pin location with respect to module center.

### 3.2.2. Network analogy

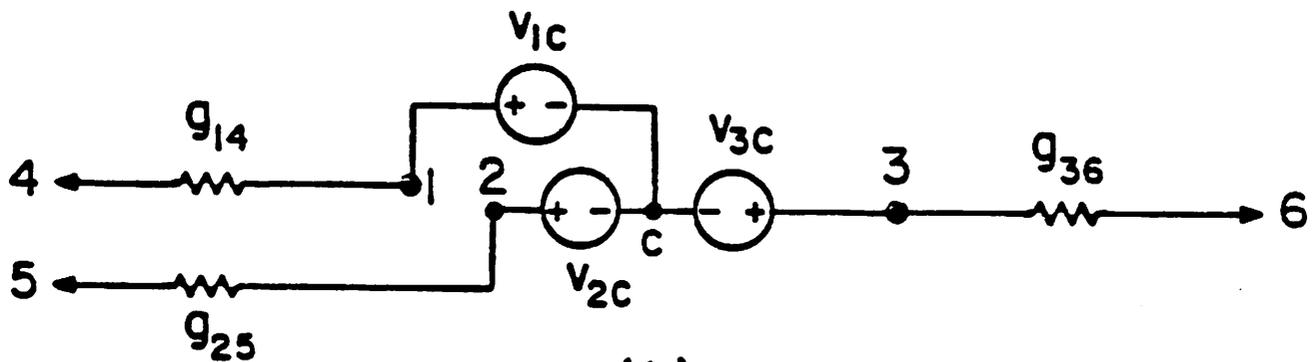
In Chapter 2, the model of point module is used for modules of same shape and size. Through network transformation, the point modules become the nodes of the corresponding linear resistive network. In this section, the model of modules is extended so that the distances between pin locations and module centers are taken into consideration.

Suppose the orientations of modules are given. The distance between pin and the center of the module is a constant. In the network analogy, we transform coordinates to voltage values. Thus, the distance between pin and module center is transformed to a branch voltage source between the corresponding nodes. Fig. 3.4 illustrates this transformation. In Fig. 3.4a, we have a module with three pins 1, 2 and 3, connected to modules 4, 5 and 6, respectively. Point  $c$  is the center of the module. Fig. 3.4b is the corresponding network of Fig. 3.4a. Pins and modules together with point  $c$  are transformed to the nodes of the network. Connectivity is transformed to the conductance and constraint of pin locations is transformed to the branch voltage sources. Through voltage transformation, we can shrink nodes 1, 2 and 3 to node  $c$  and

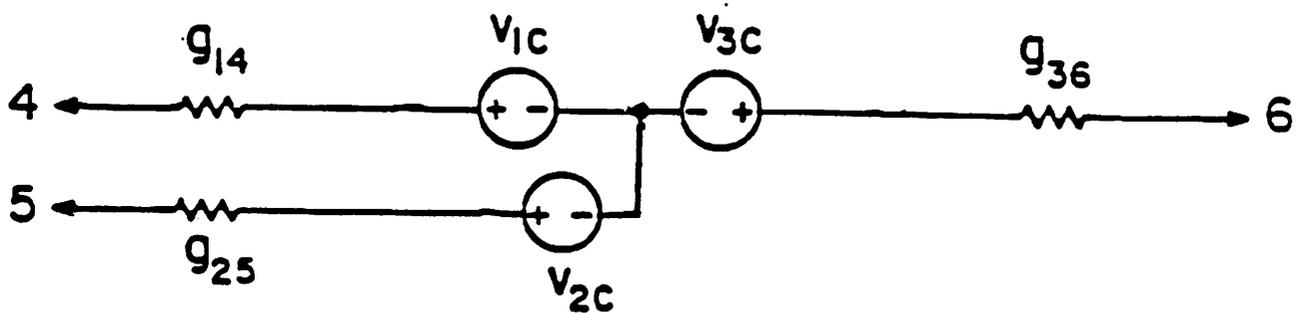
transform the branch voltage sources to the voltage sources on branches (1,4), (2,5) and (3,6) as shown in Fig. 3.4c.



(a)



(b)



(c)

Fig. 3.4. Network model of a large module with given orientation and pin location:

- (a) The module with three pins connected to other modules.  $c$  is the center of the module.
- (b) The network model of the module. Constraints on pin locations is modeled by branch voltage sources.
- (c) The network model after voltage transformation.

Thus, given  $n$  modules with pin locations on the modules, we have an analogy of an  $n$ -node linear resistive network with nodes representing the modules and branch voltage sources representing the constraint of pin locations.

Let  $G$  denote the branch conductance matrix,  $A$  the incidence matrix and  $E_b$  the branch voltage sources[6]. Let us set  $v$  to be the voltage vector of nodes. The power dissipation is now

$$P = (E_b + A^T v)^T G (E_b + A^T v) \quad (3)$$

### 3.2.3. Slot constraints

For network optimization, we can formulate the slot constraint in terms of a number of polynomial equations. As in Chapter 2, we use the first order equation only. The first order equation expresses the constraint of keeping the center gravity of modules at the center of chip. Let  $w_i$  be the size of module  $i$ . This constraint is now written as:

$$\sum_{i=1}^n w_i x_i / \sum_{i=1}^n w_i = c_x \quad (4)$$

where  $c_x$  is the center of the region.

## 3.3. Proposed method

The network optimization is modified to deal with modules of different size and the pin location constraints. A merging operation is introduced to improve the result after partitioning. We also describe the decompaction in subsection 3.3.5.

### 3.3.1. Optimization

Decomposing the voltage vector into floating voltage vector  $v_1$  and the fixed voltage sources  $v_2$ , we have the following formula for the power dissipation.

$$P = E_b^T G E_b + v_1^T y_{11} v_1 + 2v_1^T y_{12} v_2 + v_2^T y_{22} v_2 + E_b^T G A_1^T v_1 + E_b^T G A_2^T v_2 \quad (5)$$

where  $y_{11}, y_{12}, y_{22}$  are the short-circuit submatrices of the indefinite admittance matrix  $Y=AGA^T$ .  $A_1$  and  $A_2$  are the submatrices of  $A$  corresponding to vector  $v_1$  and  $v_2$ .

We want to minimize the power dissipation subject to the constraints in Eq. (4). Similar to the result of Chapter 2, we obtain the following results from the well-known Kuhn-Tucker conditions:

$$v_1 = v_a + y_{11}^{-1} i_a \quad (6a)$$

$$v_a = y_{11}^{-1} \left[ -y_{12} v_2 - A_1 G E_b \right] \quad (6b)$$

$$i_a = \frac{d - w^T v_a}{w^T y_{11}^{-1} w} w \quad (6c)$$

where  $w$  is a vector of elements  $w_i$  and  $d$  is the constant to maintain the first order constraint.

It is seen that that first term in Eq. (6a) represents the solution for which there is no constraint on slots. The second term of Eq. (6a) can be viewed as a correction term which attempts to align the solution on slots. In terms of electric network, we may use current sources to interpret the effect. Thus we have a linear resistive network with both voltage and current sources.

Any deviation from this solution can cause the power dissipation to increase. Let us assume that we deviate away from the solution  $v_1$  of Eq. (6) by  $\delta v_1$  under the constraint of Eq. (4), i.e.

$$w^T \delta v_1 = 0 \quad (7)$$

Then we claim that the power dissipation is increased by

$$\frac{1}{2} \delta v_1^T y_{11} \delta v_1. \quad (8)$$

**Proof:**

The proof is similar to the proof in subsection 2.2.1 except that the effect of the branch voltages and sizes of modules are considered here. From equation (5), we have

$$\Delta P = P\{v_1 + \delta v_1\} - P\{v_1\} = \delta v_1^T y_{11} \delta v_1 + 2v_1^T y_{11} \delta v_1 + 2\delta v_1^T y_{12} v_2 + E_b GA^T \delta v_1$$

From equation (6)

$$y_{11}(v_1 - v_a) = i_a$$

and using equation (7), we obtain

$$\Delta P = \delta v_1^T y_{11} \delta v_1$$

**Q.E.D.**

Therefore the increase in power dissipation has an upper bound which is proportional to the norm of the deviation  $\delta v_1$ .

### 3.3.2. Scaling

The formulation of scaling in Chapter 2 is modified to take into account the sizes of modules. Without loss of generality, we assume there are certain number of legal locations for modules to be assigned in order to formulate the mean position and root mean square amplitude of the desired module distribution. Let us assume that in a region or a subregion, there are  $k$  modules with  $m$  legal locations given by  $[p_1, p_2, \dots, p_m]$ . Let  $[x_{o1}, x_{o2}, \dots, x_{ok}]$  denote the solution obtained from optimization and let  $[x_{n1}, x_{n2}, \dots, x_{nk}]$  denote the new solution after scaling. For simplicity, we minimize the following objective function

$$\sum_{i=1}^k w_i (x_{ni} - x_{oi})^2 \quad (8)$$

under the first and second order slot constraints

$$\frac{\sum_{i=1}^k w_i x_{ni}}{\sum_{i=1}^k w_i} = \frac{\sum_{i=1}^m p_i}{m} \quad (9)$$

and

$$\frac{\sum_{i=1}^k w_i x_{ni}^2}{\sum_{i=1}^k w_i} = \frac{\sum_{i=1}^m p_i^2}{m} \quad (10)$$

The solution is given by the Kuhn-Tucker conditions, i.e. For  $i=1,2,\dots,k$

$$x_{ni} = \frac{x_{oi} - c_o}{a_o} a_n + c_n \quad (11)$$

where

$$c_n = \frac{1}{k} \sum_{i=1}^k p_i \quad (12)$$

$$a_n = \left[ \frac{1}{k} \sum_{i=1}^k (p_i - c_n)^2 \right]^{\frac{1}{2}} \quad (13)$$

$$c_o = \frac{\frac{1}{k} \sum_{i=1}^k w_i x_{oi}}{\sum_{i=1}^k w_i} \quad (14)$$

and

$$a_o = \left[ \frac{\sum_{i=1}^k w_i (x_{io} - c_o)^2}{\sum_{i=1}^k w_i} \right]^{\frac{1}{2}} \quad (15)$$

where  $c_o$  is the mean position of the computed module positions and  $a_o$  is the root mean square amplitude from  $c_o$ . If  $a_o$  approaches zero, so does  $x_{oi} - c_o$  in Eq. (11), and Eq. (11) can be replaced by

$$x_{ni} = c_n \quad (16)$$

It is easy to check that after scaling, the objective function in equation (8) is equal to

$$\sum_{i=1}^k w_i * \left[ (a_n - a_o)^2 + (c_n - c_o)^2 \right] \quad (17)$$

### 3.3.3. Relaxation

The result of relaxation, as described in Chapter 2, leads to modules more or less confined to the center of the region. Therefore, relaxation scheme is used to spread the modules to the whole region. The method calls for repeated use of scaling and optimization over the subregions specified by designers. Modules are selected to the subregions according to the order of their

coordinates. We try to make the total size of the selected modules equal the capacity of the subregion. Since the sizes of modules are not uniform, total size of the selected modules and the capacity of the subregion may not match. In order to make the difference small, we develop a function to define the number of modules to be selected. Let  $P$  be the capacity of the subregion. Let module  $i$  be the critical module such that by adding this module the sum of selected module sizes would increase from a value  $T$  to  $T + w_i$  where  $T < P \leq T + w_i$ . If  $P - T > T + w_i - P$  then we select module  $i$ . Otherwise it is not selected. Thus, the difference is smaller than the size of the largest module. We define the number of the selected modules by function  $f(P)$ .

Thus the relaxation is described as:

Input:

A one-dimensional region with coordinates of movable modules  $x_i$ ,  $i=1,2,\dots,m$  obtained from initial optimization in the entire region with specified fixed modules  $x_i$ ,  $i=m+1,m+2,\dots,n$  on the boundary. Sum of the movable modules is  $S$ . A parameter  $\beta$  is to be chosen by the designer with  $0 < \beta < 50\%$ .

Relaxation:

- (1) Order the modules left to right according to coordinates of the centers of modules
- (2) Choose  $f(S*\beta)$  modules from the left, setting other modules fixed and do scaling in the left  $\beta$  region.
- (3) Fix the modules so determined in the left  $\beta$  region and release the modules in the right  $(1-\beta)$  region. Do optimization.
- (4) Choose  $f(S*\beta)$  modules from the right, set other modules fixed and do scaling in the right  $\beta$  region.

- (5) Fix the modules in the right  $\beta$  region and release the modules in the left ( $1-\beta$ ) region. Do optimization.
- (6) Choose  $f(S*\beta)$  modules from the left, set other modules fixed and do scaling in the left  $\beta$  region.
- (7) Set modules in both the left  $\beta$  region and the right  $\beta$  region fixed and release the modules in the center subregion. Do optimization.

Output:

A one-dimensional region with  $m$  modules and their new coordinates  $x_i$ ,  $i=1,2,\dots,m$ .

### 3.3.4. Partitioning and Merging

In partitioning, we divide a given region into two subregions. The modules in the region are redistributed to the two subregions. Also we try to make the ratio of module sizes in two subregions approaches the ratio of the capacities of the subregions. Let  $R$  be the ratio of the capacities of two subregions and  $S$  be the total size of the modules to be partitioned. The intended total sizes of modules in the two subregions would be  $S*\frac{1}{1+R}$  and  $S*\frac{R}{1+R}$ . Thus,  $f(S*\frac{1}{1+R})$  defines the number of modules to be separated from other modules.

Each time we list all regions on the chip as current regions. The partitioning step divides the current regions into subregions. After all current regions are partitioned, we use a merging step to improve the result. We use a window to cover part of subregions, merge these subregions and do the partitioning again. We define two kinds of windows: horizontal and vertical. In the horizontal window, the number of columns of subregions is larger than the number of rows of subregions. In the vertical window we have a larger number of rows of subregions. In Fig. 3.5 we use a ( 1 x 4 ) horizontal window. We scan rows from right to left with an increment of 2 and top to bottom with an increment of 1. The

illustration of vertical window could be shown by rotating this picture 90 degrees and reflecting with respect to the x axis.

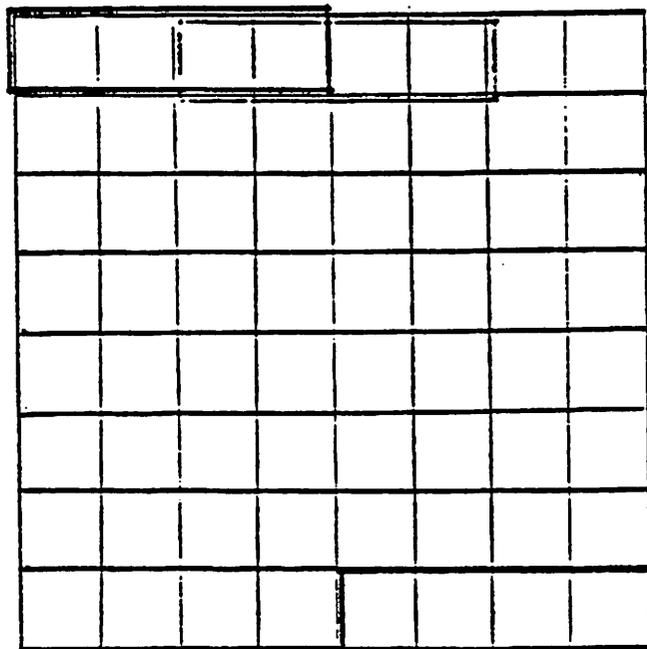


Fig. 3.5. Illustration of horizontal window.

### 3.3.5. Assignment

We continue partitioning the chip until all modules are assigned to the predefined rows, while keeping the total size of modules in each row approximately the same. Because the size of modules is not uniform, two different rows

of modules may not have the same size. When one row of the region is partitioned into two parallel rows of subregions, we check the difference of sums of module sizes between the two rows in the partitioning of each region. Thus, the difference of the total module sizes between the two rows is not larger than the size of the largest module.

Initially, because the orientation of modules is not determined yet, point model is assumed to represent all the modules. After optimization, the relative locations of modules are determined. Based on those locations, the reflection of each module is determined to minimize the wiring length. This process is repeated after the partitioning in each level.

When modules have been assigned to rows, they might overlap, because the modules have different shapes. A decompaction step is done on each row to separate the overlapping modules. Scanning from right to left, we assign the feasible range of location for each module. For the current module, its left bound is set by the right edge of the previous module, and the right bound is set so that the capacity of the right region is equal to the sum of sizes of modules on the right hand side. Then the module is set between the boundary and located as close as possible to its original position obtained after partitioning. Fig. 3.6 illustrates this process. The modules are initially located in one row as shown in Fig. 3.6a. Fig. 3.6b demonstrates the process of decompaction. The vertical dotted lines indicate the left and right bounds of module 3. Module 3 is located within the boundary and set near to its initial location. Fig. 3.6c shows the result of decompaction.

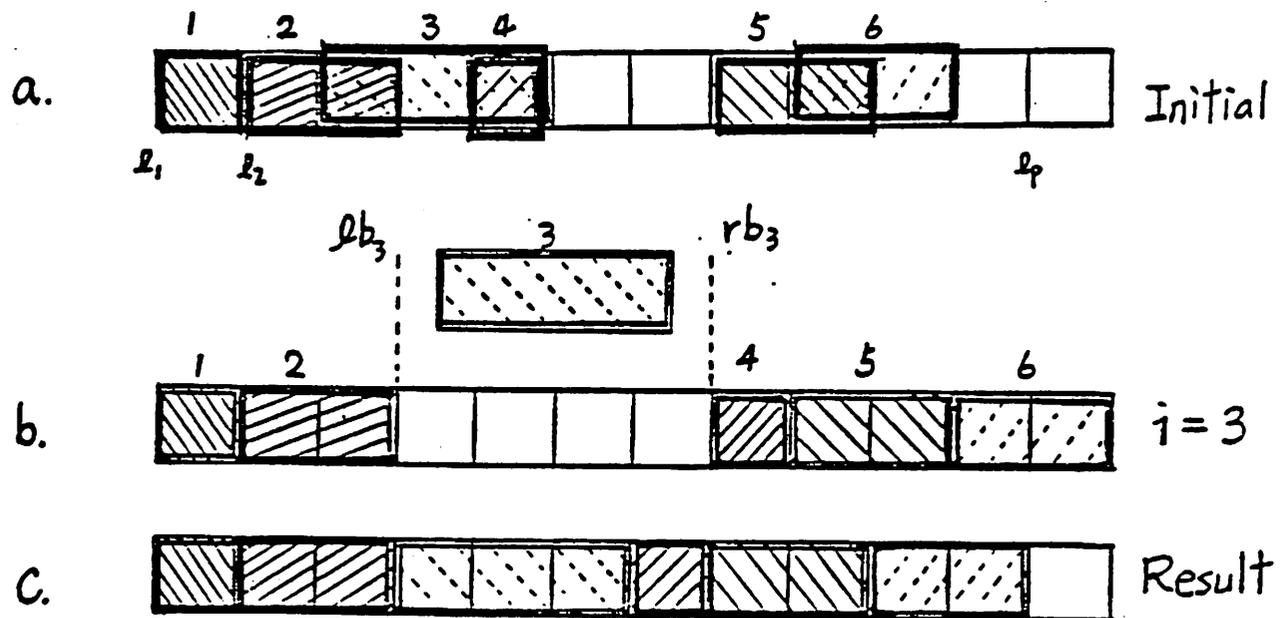


Fig. 3.6. Decomposition on one row:

- (a) Initial locations of modules.
- (b) Decomposition process on module 3.
- (c) Result of decomposition.

The following shows the assignment algorithm.

Input:

A 2-dimensional region to be partitioned into rectangles each containing a

module, a set of  $m$  movable modules together with their coordinates, and a set of  $n-m$  fixed modules.

**Assignment:**

(1) Do optimization on both the  $x$  coordinate and the  $y$  coordinate of the movable modules.

(2) While each region contains more than one module

Do

List all current regions.

For each region do partitioning.

Use horizontal window.

Do merging and repartitioning.

Use vertical window.

Do merging and repartitioning.

(3) For each row

Do decompaction.

### **3.4. Experimental results**

The above method has been implemented and tested with gate-array designs used at Hughes Aircraft Company and other industrial companies. Table 3.1 shows the comparison with manual designs on 4K gate-array chips. Four chips have been tested. It is shown that the sums of squared wiring lengths are reduced. Fig. 3.7 gives the solution of automatic placement of chip 1. Fig. 3.8 exhibits results in terms of wirability of chip 1 for both the automatic and manual designs. The abscissa represents cut-lines in the horizontal and vertical directions. The ordinate represents percentage of routing track demand oversupply. It is seen that in the horizontal direction both the manual and

automatic placements yield results which are easily routable. However, in the vertical direction, the manual placement requires over a hundred percentage of track demand over supply, which is clearly unroutable, while the automatic placement requires a peak percentage of less than 70. In general, automatic placement tends to distribute the wires more uniformly and thus achieves better routability. The cpu time of the above placement is about one minute on an Amdahl V8.

Chip #	manual placement	network optimization
1	$5.98 \times 10^9$	$1.31 \times 10^9$
2	$3.70 \times 10^9$	$3.37 \times 10^9$
3	$2.91 \times 10^9$	$1.83 \times 10^9$
4	$2.16 \times 10^9$	$0.94 \times 10^9$

Table 3.1. Placement results of four 4K gate-array chips in terms of sum of the squared wire length.

Chip 1: 317 modules, 676 nets and 2284 pins.

Chip 2: 255 modules, 916 nets and 2049 pins.

Chip 3: 442 modules, 983 nets and 3012 pins.

Chip 4: 484 modules, 1030 nets and 1969 pins.

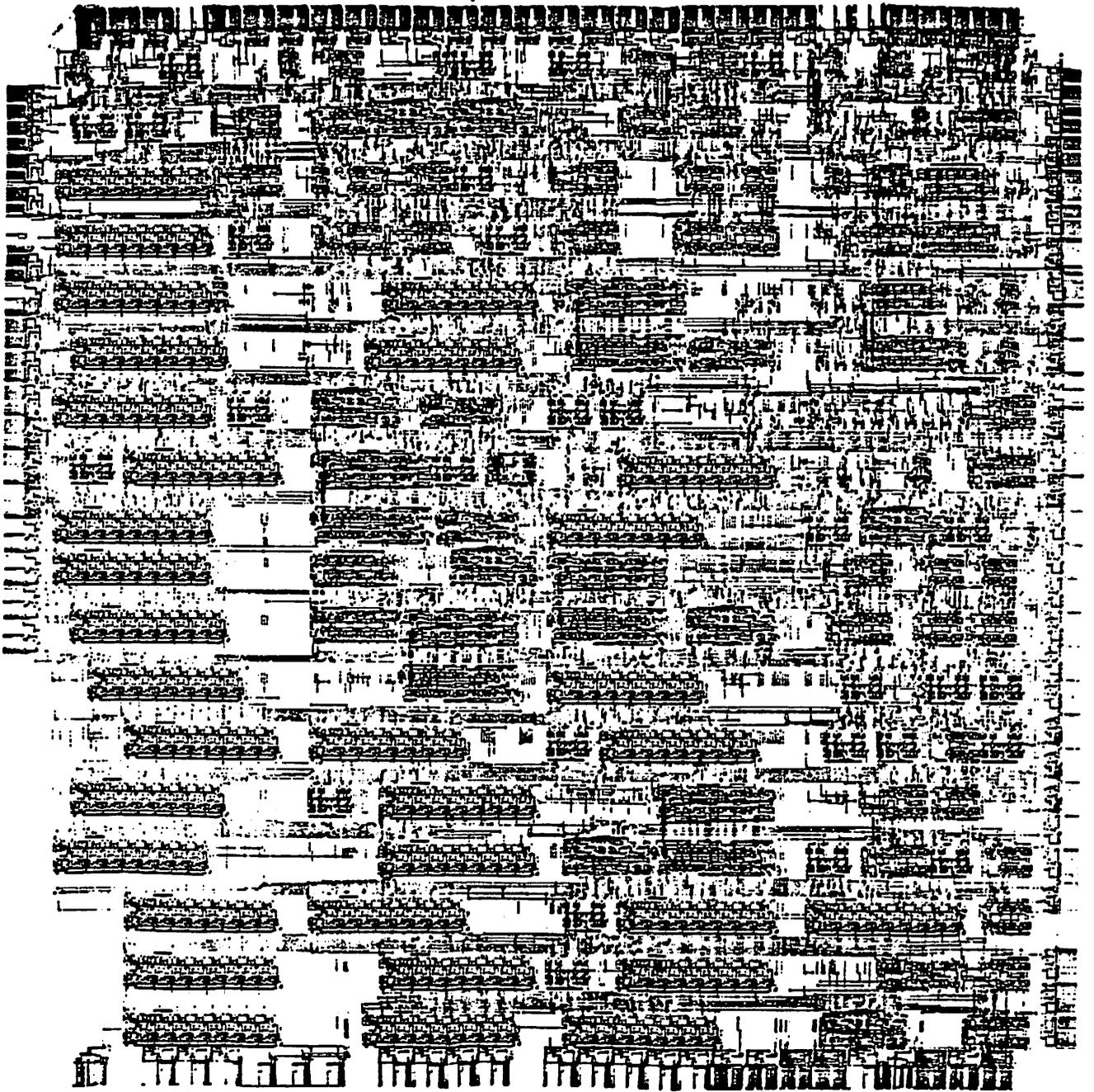


Fig. 3.7. Picture of the chip placement.

# Percentage of Track Demand vs. Supply

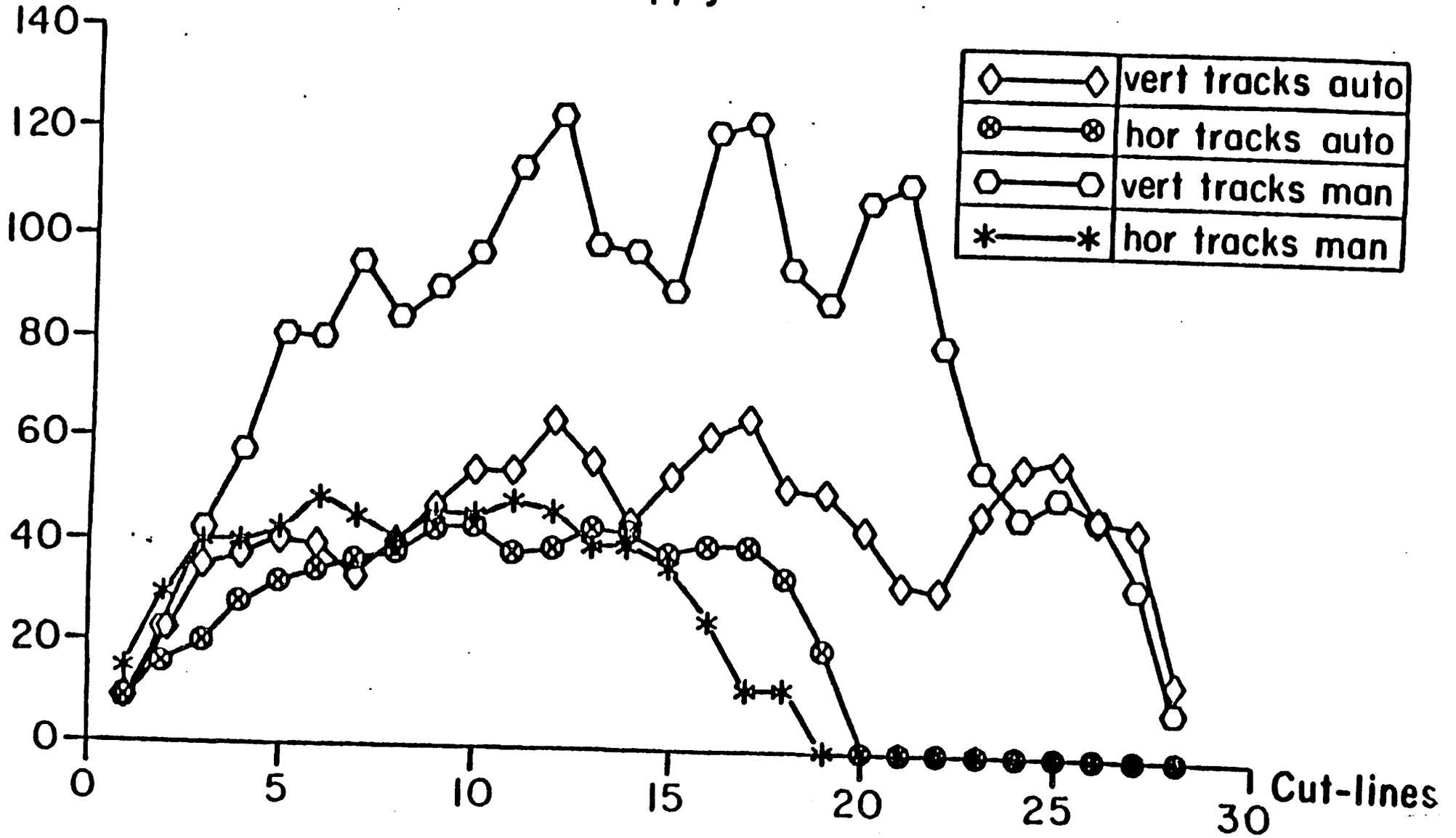


Fig. 3. Placement results measured in terms of the number of wires crossing the cut-lines on chip 1.

We also compare with the traditional iterative improvement method[14] on a 2K gate array example. The specification of the example and the results are given in Table 3.2. It is shown that not only the sum of squared wire lengths decreases, but the sum of wire lengths is also reduced by 26%. This example takes 783.2 seconds of cpu time on VAX 11/780.

	Iterative improvement	Network Optimization
Manhattan length	77326	57191
Squared length	$9.99 \times 10^6$	$3.01 \times 10^6$

Table 3.2. Comparison with iterative improvement on 2K gate-array chip

445 modules, 464 nets and 1713 pins

### 3.5. Conclusion

We have implemented a resistive network optimization method for gate-array and standard-cell placement. Point modules are replaced by modules of different sizes on cell rows. A decompaction process is proposed to separate modules on each row. The results applied to 4K gate-array chips lead to far superior results than that of manual placements. Also, the comparison with the traditional iterative improvement method on a 2K gate-array example shows that our method produces better results.

## Chapter 4

### Building Block Placement

#### 4.1. Introduction

The building block layout is a popular approach for developing high-density, high production-volume integrated circuit chips. However, due to the wide variety of sizes and shapes of modules used in building block systems, the placement problem becomes very complicated.

To dissect a rectangle into a finite number of non-overlapping squares, Tutte et al.[15] introduced a planar directed graph. This planar directed graph is later used as polar graph in building block placement. Based on the polar graph representation, many placement methods have been developed. Lauther[16] combined this graph representation with a min-cut placement algorithm to partition the modules into separated areas and improve the result by rotating and flipping of the modules. Hsueh and Pederson[17] also derived from the same graph a compaction algorithm to reduce the size of the chip. However, in the compaction, the connectivities among the modules are not taken into account. As a result, the wiring length might increase after a few interactions of compaction.

The structure of polar graphs represents the relative locations of the modules. There are different descriptions of the polar graph with respect to the implementations. This chapter uses a graph representation[17] which is extended from the traditional polar graph so that free spaces are allowed on the chip. Let us consider an example of such a graph illustrating placement of modules along the x-axis. In the graph, there are one source node and one sink node representing the left and right edges of the chip, respectively. Other vertices represent the modules. There is a branch  $(a,b)$  directed from a vertex  $a$  to vertex  $b$  if the vertices  $a$  and  $b$  in the graph correspond to "horizontal adjacent"

modules in the layout, and module a is on the left hand side of the module b. Two modules are horizontal adjacent if there exists a horizontal line which intersects with both modules, and no module placed between them is cut by this line. Each branch has a weight associated with it which equals the sum of half widths of the modules. This representation is shown in Fig. 4.1. The placement in Fig. 4.1a is represented by the graph shown in Fig. 4.1b. The free space is not included in the graph. Note that the topology of the graph might change when modules are moved from their current locations. It is important to note that the longest path in the graph determines the width of the chip.

In the following sections, we describe a novel algorithm. This method applies the network optimization method[18] to locate the initial relative module locations. In order to spread the modules, the partitioning step of the network optimization in Chapter 2 is first used to divide the chip into 4x4 subregions. The modules whose dimension is comparable to the dimension of subregions are considered as critical modules and placed at this stage. After their position has been fixed, the capacity of each chip region is updated and network optimization method is used to relocated other modules. A reduction process is used to separate the overlapping modules and reduce the chip size, while still maintaining the original relative module locations. In preliminary experiments, this method generates results comparable with manual placement.

## 4.2. Formulation

Consider a set of rectangular modules of different shapes and sizes to be located within chip area. On the four sides of the modules are pins for connections. Given a netlist and external pads fixed on the boundary of the chip, the nets connect external pads and modules. The placement problem is to place the modules with the goal of minimizing the chip area and keeping modules with strong connectivities near each other.

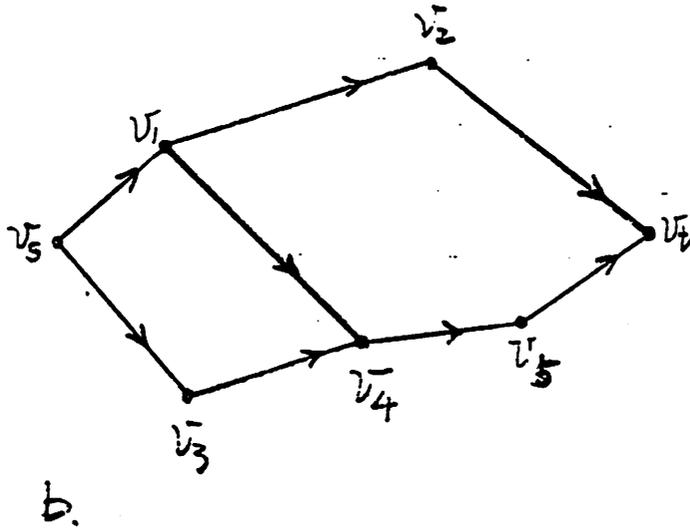
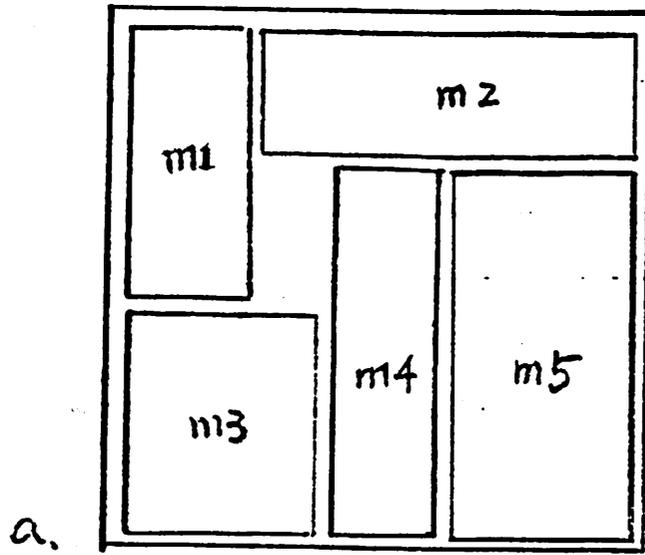


Fig. 4.1. Illustration of polar graph.

The two objectives of minimizing the area and placing modules with strong connectivities close together are related to the cost of chip manufacturing and to its performance. The advantage of small chip size is obvious, while the objective of contiguous placement of modules with strong connectivity is needed to reduce the time delay of the signals. We choose the objective function of sum of squared wire lengths since a small sum leads to both the chip size minimization and delay-time reduction.

The results of placement are input to a routing system which wires the nets among the modules and pads to complete the layout. Therefore, the size of the routing areas also affect the results of layout. To concentrate on the placement itself, we assume the routing areas are given. The modules are expanded in both dimensions by an amount equal to half the channel width to cover the routing areas.

We also use grids to divide the chip area into basic square cells. The size of modules are then rounded off to the multiple of the basic square cells. Thus, the numbers of columns and rows of the grids define the size of the chip. The placement problem amounts to assigning modules on grids. This formulation simplifies the explanation and programming of the algorithms. However, the algorithms can be easily extended to gridless cases.

Donze and Sporzynski[19] have introduced the masterimage approach. This approach has a basic chip structure of gate array. The modules are allowed to have different widths and height, however. This method is considered to be intermediate between unconstrained and constrained approach. Thus, in the case that the channel widths are forced to be a constant and the module orientations are suitably constrained, the general building block placement problem reduces to the masterimage placement problem.

### 4.3. Preliminary locations of the modules

The network optimization method is used to determine the preliminary locations of the modules with the objective of minimizing the sum of squared wiring lengths.

Initially, each module is modeled by a single node of the network. After the optimal locations of modules are obtained, we choose the orientation of the module to minimize the wiring length. Then, in the corresponding network, the relative positions of pins are modeled by voltage sources connecting nodes pertaining to the module.

We assign modules into subregions in order to distribute the size of modules evenly over the chip. Because the size of modules is not uniform, the partitioning process checks the sum of sizes of modules in each partitioned row or column. For instance, when one row of regions is further partitioned into two parallel rows of subregions, we check the difference of sums of module sizes between the two rows in the partitioning of each region. Thus, the difference of the total module sizes between the two rows is not larger than the size of the largest module.

### 4.4. Basic operations

With respect to the preliminary module locations, the modules might overlap due to their different sizes and shapes. This section introduces four basic operations, namely, compaction, decompaction, rotation and selection of preferable direction. The operations are processed in one dimension of the chip. Then the algorithms call for repeated use of the operations in each dimension to separate the modules and to minimize the chip size.

Based on the theory of the polar graph, we know that the longest path from the source to the sink is assigned to be equal to the dimension of the chip. The

basic operations are used to shift or rotate the modules so that the longest path length is reduced. However, instead of manipulating on the abstract polar graph, these operations deal with the modules directly on the chip. Therefore, the operations can handle the overlapping of the modules and take into account the preliminary locations of the modules. As a result, the operations become easy for coding and efficient for the layout.

#### 4.4.1. Compaction

Given the chip area, the objective of compaction is to minimize either the x or y dimension of the chip. Because the process is the same in the x and y directions, let us describe the compaction with respect to x direction. Along the x axis, the modules are first ordered from right to left. Following this order, we shift the modules to the right edge of the chip, with the constraint that no modules overlap.

Therefore, after the compaction, all modules are separated. However on the left edge of the chip, some modules might fall outside the chip area. Formulating this problem in the terminology of polar graph, we give the following definitions.

- i **Critical path:** After the modules are separated, we can construct the polar graph. Any path on the polar graph which is larger than the width of the chip forces the modules to fall outside the boundary of the chip. Let us define such a path to be the critical path.
- ii **Slackness:** Where there is no critical path, the width of the chip is larger than or equal to the longest path of the polar graph. In this case, we define the slackness to be the difference between the width of the chip and the length of the longest path.

#### 4.4.2. Decomposition

While compaction shifts the modules toward one side of the chip, decomposition pulls them back near the preliminary locations. The modules are now ordered in the reverse order. We scan from left to right and assign the feasible range of location for each module. For the current module, its left bound is set by the right edges of the previous modules, and its right bound is set by its right edge as determined by the compaction operation. Then the module is set between the left and right bounds and located as close as possible to its preliminary location. However, for the module on the critical path, there is no possible variation in position. Then, these modules are put back at the previous locations and some of the modules overlap again. In this way, next time, when decomposition is processed in the orthogonal direction, they are forced to split apart in that direction.

However, if the modules are also on the critical path in the orthogonal direction, the decomposition operation also puts these modules back at the same locations. As a result, no improvement can be achieved through further decompositions. In order to avoid this problem, the modules are shifted from the previous locations by one grid toward the location determined in the compaction process.

The decomposition is listed as follows:

Input:

The width and the height of the chip.

Preliminary locations of the modules derived from the network optimization method.

The module locations obtained from the last decomposition process.

Decompaction:

1. Do compaction in x direction.
2. Do from left to right

For each module

If it is not in the critical path

Then

Set

a.left bound: The left edge of the chip or the right edges of the previous modules.

b.right bound: Its location obtained from step 1 above.

Locate this module at a minimum distance from its preliminary location within the bounds.

Else

Set it at the previous module location with one grid deviation toward the location obtained from step 1.

Output:

The module locations.

### Example

This example demonstrates the compaction and the decompaction operation in the x direction. Suppose the preliminary locations of the modules are given as Fig. 4.2. In step 1, we order the modules from right to left. Each module is slid toward the right edge of the chip with the constraint that no modules should overlap[Fig. 4.3]. Then the modules are processed in the reverse order. Fig. 4.4 illustrates the process on the third module. Its left bound and right bound are shown by the bold lines on both sides of the free space in the chip. The module is pulled back near the preliminary

location within the bounds. Because no critical path occurs in this example, all modules are finally assigned within the chip area without overlapping [Fig. 4.5].

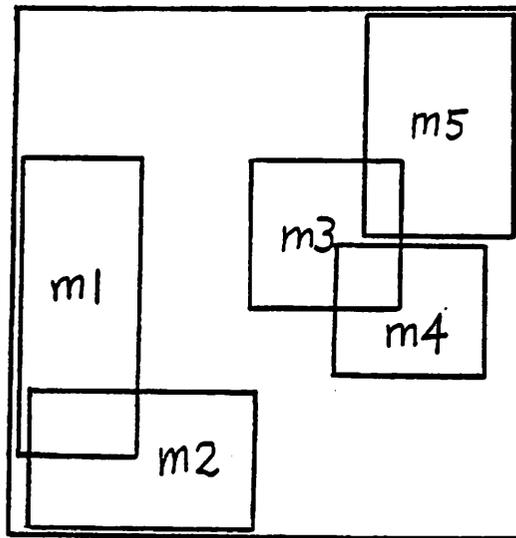


Fig. 4.2. Illustration of five-module example with preliminary locations.

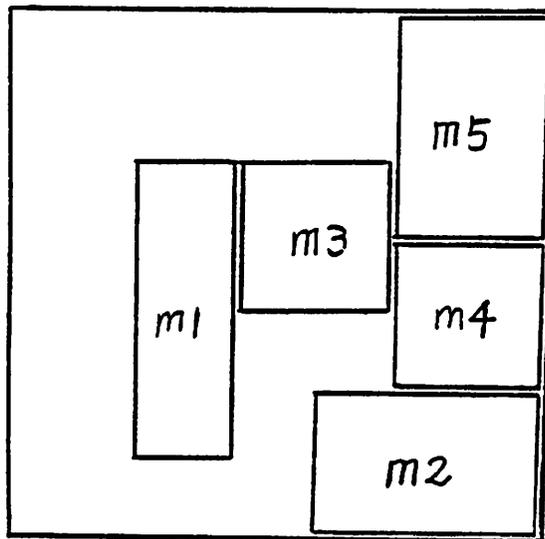


Fig. 4.3. Result of compaction on five-module example.

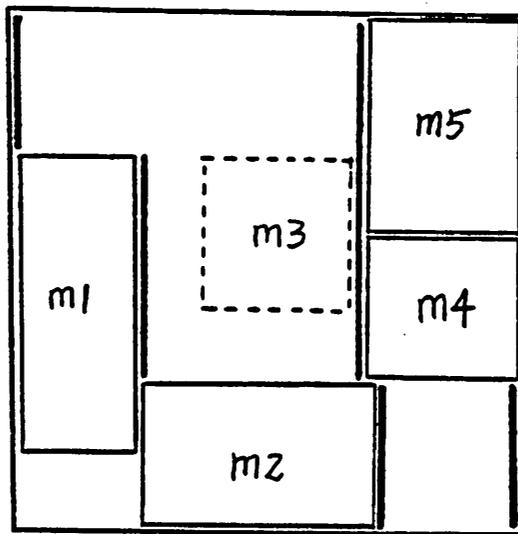


Fig. 4.4. Illustration of decompaction process on module m3.

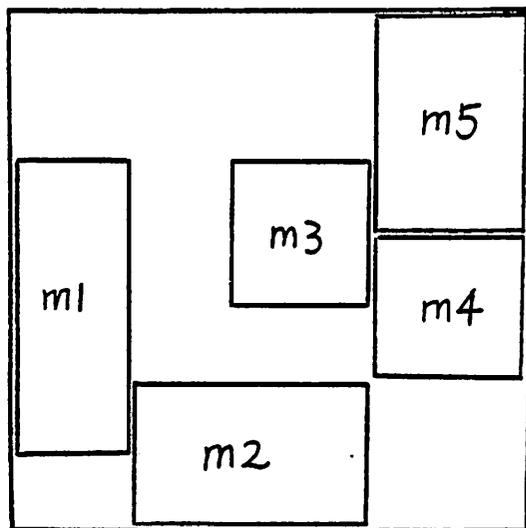


Fig. 4.5. Result of decompaction of five-module example.

#### 4.4.3. Rotation

Given the critical paths, the rotation operation selects modules on these paths and changes their orientation by ninety degrees, in order to reduce the length of the paths. In particular, the modules are chosen for rotation if their

longer edges are parallel to the direction of the critical path. Clearly, a ninety degree rotation places the module's shorter edge on the critical path, reducing the path length.

The situation in the direction orthogonal to the critical path is also considered in order to avoid the possibility of a rotation generating a new critical path in that direction. On each column of the grid, we define the **load** of the column by the sum of the heights of the modules which intersect this column. The modules on the column of minimum load are then chosen for rotation.

The rotation against the critical path in x direction is stated below to describe the process.

Input:

The locations of modules derived from decompaction.

The critical path in x direction.

Rotation with respect to the critical path in x direction:

1. Calculate the load of each column.
2. Among the modules on the critical path, select a set of the modules which have the width larger than the height. Check the loads of the columns which are crossed by the centers of these modules.
3. Among the selected set of the modules, select the modules whose centers are located on the column of the minimal load.
4. Rotate the chosen modules.

#### **4.4.4. Selection of preferable direction**

Since the above operations are done in one dimension, the placement result might depend on the choice of the direction of the operation. We set the preferable direction according to the lengths of the longest paths in the x and y direc-

tions. To begin with, compaction is used to find the longest path parallel to each axis. Then for each direction, the ratio of longest path length to chip dimension in that direction is calculated. The axis with a smaller ratio determines the preferred direction which is operated first.

#### 4.5. Algorithms

Given the preliminary locations of modules, and the width and height of the chip, the **spacing** algorithm is proposed to separate the overlapping modules by repeated use of decompaction and rotation. According to the slackness obtained in spacing, the **reduction** algorithm reduces the chip size and uses scaling to redistribute the modules. Given the new chip size, **spacing** is used again to separate the modules. The reduction of the chip size is repeated until spacing fails to assign the modules inside the chip.

##### 4.5.1. Spacing

Through iterations, the spacing algorithm uses decompaction and rotation to change the topology of the polar graph until there is no critical path.

Input:

The width and the height of the chip.

Preliminary locations and the previous locations of the modules.

Two integer constants: number of iteration, ( denoted  $\#iterations$  ) and number of loops, ( denoted  $\#loops$  ).

Spacing:

Do  $i = 1$  to  $\#iterations$

    Select the preferable direction.

    Do  $j = 1$  to  $\#loops$

Do decompaction in the preferable direction.

Do decompaction in the other direction.

Do rotation.

If there is no critical path,

Then jump outside the do-loop.

If there is no critical path

Then Output.

Else spacing fails.

Output:

The result of spacing.

The integer constants are set as the threshold for the number of processes. In the experiments, we set #loops as two. With a proper estimation on the chip size, it takes only a few iterations to eliminate the critical path. Therefore, ten is sufficiently large for the value of #iterations.

#### 4.5.2. Reduction

The chip size is decreased by the reduction algorithm. In each iteration, either the width or height of the chip is reduced by an amount determined from slackness which was previously calculated in compaction. Consequently, with this reduction, the preliminary locations of the modules are scaled to the new chip size. In order to allow freedom in the other direction, the dimension of the chip is reduced by only half the available slackness, but not smaller than a fixed value.

Input:

Initial width and height of the chip.

Preliminary locations of the modules.

Reduction:

Repeat

1. Do spacing

2. Check the slackness  $S_x$  and  $S_y$  in x and y directions, respectively

3. If  $(S_x > S_y)$  or  $(S_x = S_y \text{ and chip width} \geq \text{chip height})$

Set chip width = chip width -  $\max[S_x/2, 1]$

Else

Set chip height = chip height -  $\max[S_y/2, 1]$

Update the preliminary locations of the modules by scaling

according to the ratio of new chip dimension to old chip dimension.

Until spacing operation fails

Output:

Final spacing.

### Computational complexity

In compaction and decompaction, we sort the order of the modules, and assign them on the grids one by one. Therefore, given  $n$  modules, it takes  $O(n \log n)$  operations. In rotation, the search is done among the modules on the critical path. It is a linear time operation.

The spacing algorithm uses decompaction and rotation in each iteration. Because we set a threshold on the number of iterations, the computational complexity is also  $O(n \log n)$ .

Since the **spacing** is repeated in the reduction, the total computational complexity of reduction is  $O(\#\text{spacings} * n \log n)$  where  $\#\text{spacings}$  indicates number of iterations on **spacing**.

#### 4.6. Assignment

Since the shapes of the modules are irregular, we divide the modules into two types and place them in two stages. Modules with one edge larger than one third of the width or of the height of the chip are classified as **critical modules**. Critical modules tend to move more in the spacing algorithm. We therefore place the critical modules in the first stage. Then we locate other modules to fill into free spaces left by the critical modules. Finally, the **reduction** is used to minimize the chip size.

Input:

The initial width and height of the chip.

The netlist of irregular sized modules.

Assignment:

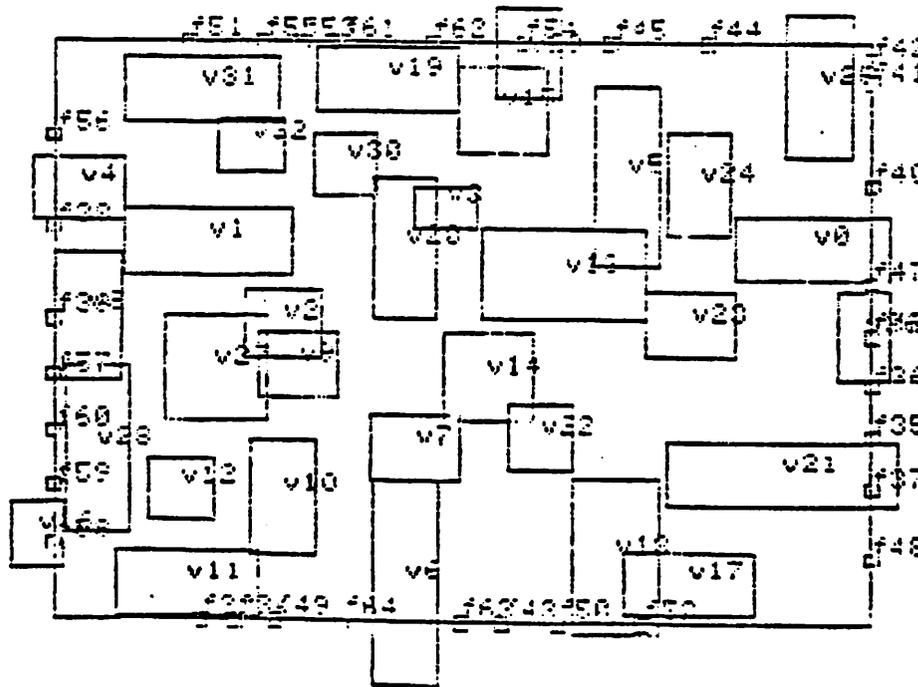
1. Use network optimization method to partition the chip into 4x4 subregions.
2. Do spacing on critical modules
3. Fix critical modules and update the capacities of the grids.
4. Use network optimization method to partition the chip into last level.
5. Do reduction on all modules.

#### 4.7. Experiments

The algorithm is implemented in C-language on a VAX 11/780 machine. In the experiment, we use the AMI example[20] which contains 33 modules, 38 external pads, 132 nets and 440 pins. The initial chip size is set as 209x294. The modules are expanded by the amount of 4 in both dimensions as a rough estimate of the routing area.

In this example, there are no critical modules. Hence the first three steps

of assignment are skipped. After the input phase is done, the network optimization method finds the preliminary locations of the modules [Fig. 4.6]. The modules spread over the chip; however, some of them overlap.



Outphase level=0

manhattan length = 10022.7  
squared length = 499216

Fig. 4.6. Result of network optimization on AMI 33-module example.

The process of **reduction** inputs the result of the network optimization method. Fig. 4.7 is the result of first iteration. All overlapped modules are separated. It takes 10 iterations to reduce the chip to a minimal area of 177x262 [Fig. 4.8].

The table below lists the sum of the wiring lengths, the sum of the squared lengths, and the CPU time of various steps in comparison with manual placement. The reduction algorithm reduces the chip size by 24.5%. Also, the sum of

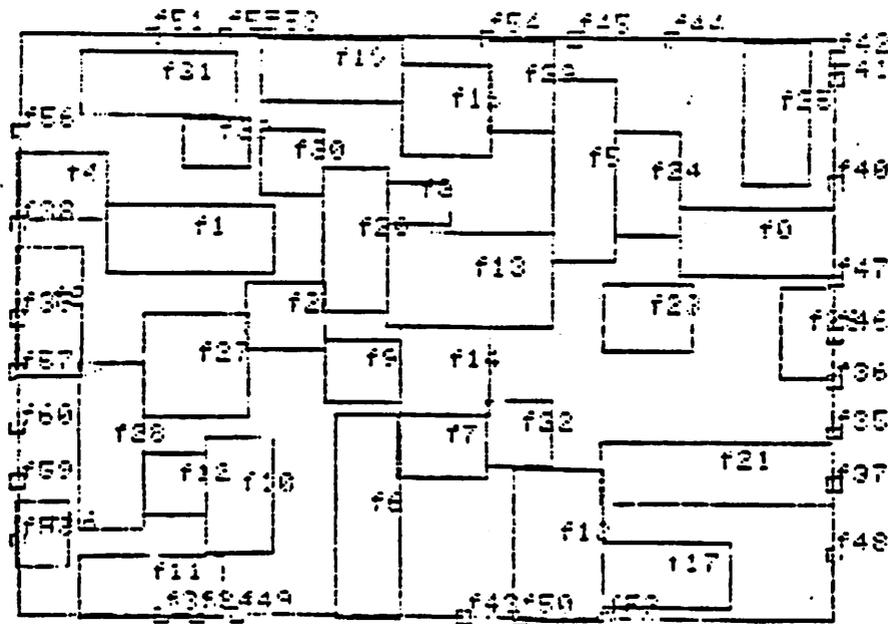
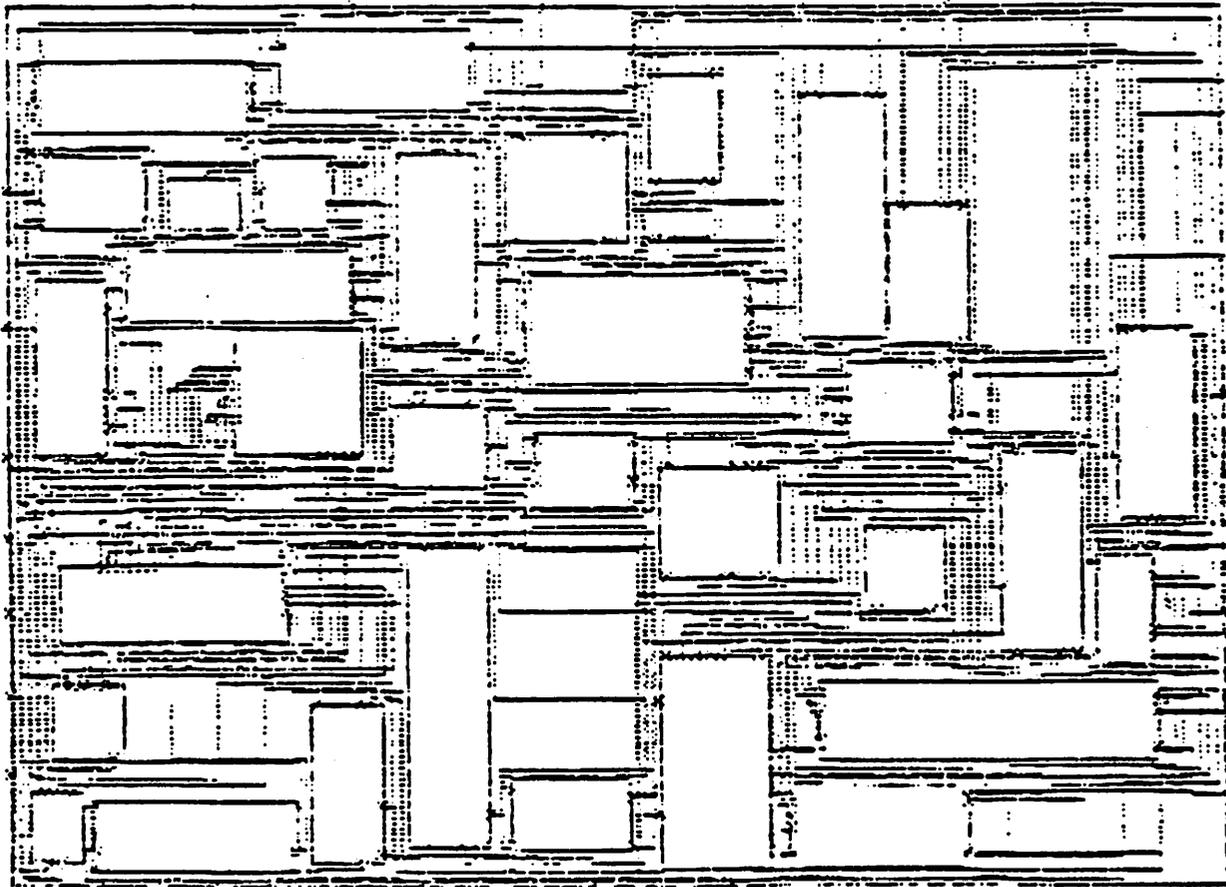


Fig. 4.7. Result of spacing after network optimization.

the wiring lengths and the sum of the squared lengths are much less than those obtained from manual placement.

	sum of the wiring lengths	sum of squared wiring lengths	CPU time (sec.)	chip size
network optimization	10022.7	488216	36.8	209x294
reduction algorithm	9440.5	446170	42.2	177x262
manual placement	13010	972228	N.A.	210x286





\*\*\*ELAPSED TIME = 202.28s\*\*\*

XMIN=0, XMAX=200, YMIN=0, YMAX=200

Fig. 4.9. Layout of AMI 33-module example from the result of automatic placement.

#### 4.8. Conclusion

We have proposed here new efficient placement algorithms. They are based on four elementary operations, namely: compaction, decompaction, rotation, and selection of preferable direction. Using these elementary operations, we developed the spacing and reduction techniques. The process of placement is an iteration of spacing and reduction. The placement is different from other

methods of iterative improvement. While in the other methods, relative placement in each iteration is changed drastically (for example by pair-wise interchange or by compaction), the proposed placement minimizes the chip area, but still maintains good relative locations among the modules. We have tried our algorithms on practical examples. The results are comparable to manual placement done by an experienced designer.

This method could be used as an automatic placement for the masterimage system. Further research on the routing area estimation would improve the layout result in general building block problems.

## Chapter 5

### Linear Placement

#### 5.1. Introduction

One important technique in circuit layout is linear placement, often used in gate matrix design and backboard ordering. The problem is known to be N-P complete. Therefore, algorithms proposed for solving the problem have been either heuristic or branch and bound. Gomory and Hu[21] introduced the useful concept of cut tree in dealing with network flow problems. Adolphson and Hu[22] have shown that when Gomory and Hu's cut tree is a chain, the sequence of this chain is optimal in terms of the sum of wire lengths and the maximum track density. However, this chain is only a special case of the linear placement problem. Lawler has determined that a linear placement problem is related to a job sequencing problem[23].

In this Chapter, we use the sum of wire lengths as the objective function. We propose algorithms for linear placement which can be represented by a parallel graph and an arbitrary graph. These algorithms decompose graphs and lead to optimal solutions. In application to circuit layout, the multi-pin net is first represented by a loop. An algorithm is constructed that can tackle very large problems efficiently.

In section 2, we give the formulation of our approach. Section 3 describes the theories related to the problem. Section 4 deals with the special case of the parallel graph, and section 5 handles the general case of the arbitrary graphs. In section 6, the application to VLSI design is discussed together with the computational complexity of the proposed method. Finally, some experimental results are given.

## 5.2. Formulation

In linear placement, the specification is the netlist together with a set of modules. The modules are to be assigned on slots equally spaced on a line. Suppose two modules are fixed on both ends of the line for the purpose of external connection. We assume that all nets are 2-pin nets and multi-pin nets have been preprocessed and replaced with 2-pin nets. Then the problem can be formulated by the following graph representation.

Given a graph  $G(V,E)$ , there is a set of vertices  $V$  of cardinality  $|V|$  and a set of edges  $E$  of cardinality  $|E|$ . The edge  $E_{i,j}$  connects vertex  $v_i$  and vertex  $v_j$ . Each edge  $E_{i,j}$  has associated with it a nonnegative number  $c_{i,j}$  denoting the connectivity between  $v_i$  and  $v_j$ . Let  $v_s$  and  $v_t$  be the two boundary vertices of  $V$  fixed at both ends of the line [Fig. 5.1].

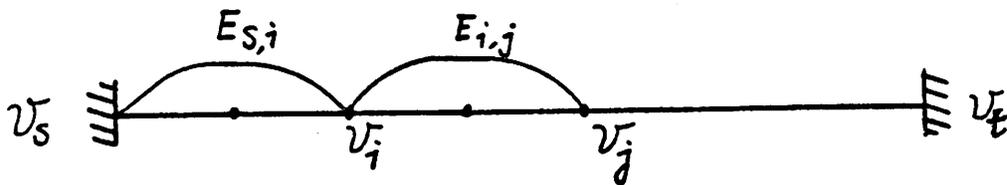


Fig. 5.1. Graph with  $v_s$  and  $v_t$  fixed on both ends and vertices to be assigned on slots.

The problem is to assign vertices onto the slots in an optimal order (OPO) under a given objective function. In [25], a method based on an interval graph model was proposed to minimize the track density. In this Chapter, the objective function used is the sum of wiring lengths, thus we wish to minimize

$$\frac{1}{2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c_{i,j} l_{i,j} \quad (1)$$

where  $l_{i,j}$  is the distance between vertex  $v_i$  and vertex  $v_j$ .

### 5.3. Theory

In terms of network flows, let  $v_s$  and  $v_t$  be the source and the sink, respectively.  $c_{i,j}$  is then the capacity between  $v_i$  and  $v_j$ . The max-flow min-cut method[26] finds a cut-line that separates  $v_s$  and  $v_t$  with a minimal sum of the capacities of lines crossing the cut-line. Hereafter, this cut is called the max-flow min-cut in order to differentiate from other cut-lines[27] used in placements.

Adolphson and Hu[22] have shown that max-flow min-cut makes a partition of OPO in terms of the sum of wiring lengths. They give the following theorem:

#### Theorem 1.

The max-flow min-cut defines a partition of OPO in the linear placement problem.

The theorem was proved by contradiction. Suppose there exists a cut-line which makes another partition of OPO in contradiction to the partition made by the max-flow min-cut. Then, based on the properties of the max-flow min-cut, it is shown that the partition made by the max-flow min-cut generates a better result.

This theorem provides a tool for graph decomposition. Consider the graph  $G(V,E)$ . Let  $A$ ,  $B$ ,  $C$  and  $D$  be four disjoint subsets of  $V$  such that  $V=A \cup B \cup C \cup D$ .

Given a placement C A B D as shown in Fig. 5.2a. Let us denote by  $CO_{XY}$  the sum of all the connectivities between the set  $X$  and the set  $Y$ . Suppose we shift B to the left hand side of set A as shown in Fig. 5.2b. Then the sum of the lengths changes. The length of the wires connecting between set A and sets C & D increases by  $[CO_{AC}-CO_{AD}]|B|$  and the length of the wires connecting between set B and sets C & D decreases by  $[CO_{BC}-CO_{BD}]|A|$ . The length of the wires connecting between set A and set B decreases no more than  $CO_{AB}[|A|+|B|]$ . Thus, the sum of the lengths increases by an amount larger than

$$\begin{aligned} & [CO_{AC}-CO_{AD}]|B| - [CO_{BC}-CO_{BD}]|A| - CO_{AB}[|A|+|B|] \\ & = [CO_{AC} - (CO_{AB} + CO_{AD})]|B| - [(CO_{BC} + CO_{AB}) - CO_{BD}]|A| \end{aligned} \quad (2)$$

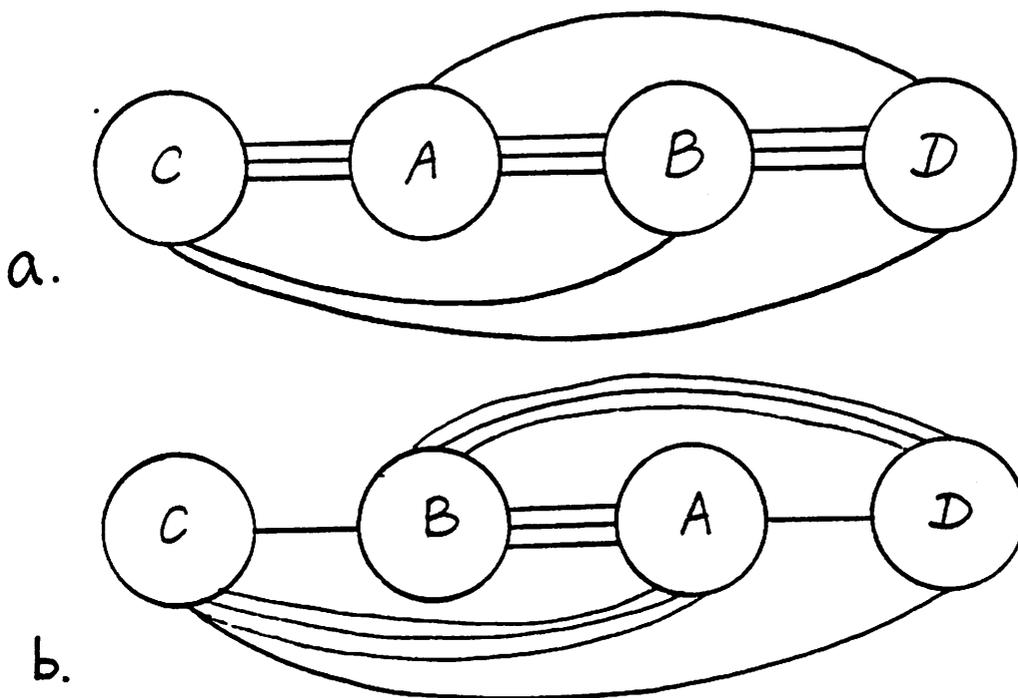


Fig. 5.2. Placement of A, B, C and D four disjoint sets.

With respect to Fig. 5.2a, let us denote by  $X_l$  and  $X_r$  the connectivity of X to the vertices on its left and on its right, respectively. That is

$$\begin{aligned} A_l &\equiv CO_{AC} & B_l &\equiv CO_{BC} + CO_{AB} \\ A_r &\equiv CO_{AB} + CO_{AD} & B_r &\equiv CO_{BD} \end{aligned} \quad (3)$$

Eq.(2) becomes

$$(A_l - A_r) | B | - (B_l - B_r) | A | \quad (4)$$

If  $\frac{Al - Ar}{|A|} > \frac{Bl - Br}{|B|}$ , then Eq. (4) becomes larger than zero. In this case, exchanging the order of set A and set B would increase the sum of lengths.

The expansion of this property leads to the following theorem. First, let us define

$$\gamma_A \equiv \left( \sum_{i \in A} c_{si} - \sum_{i \in A} \sum_{j \in V-A - \{v_s\}} c_{i,j} \right) / |A|$$

to be the **cost ratio** of set A. We can view the first two terms as the drag forces of the set A from both sides and  $|A|$  as the inertial of the set A. The ratio of the drag force to the inertial measures the priority of the order of subsets. The following theorem suggests that the subset which generates the maximal cost ratio should be placed ahead of other vertices in the OPO. Note  $\gamma_A$  can be negative.

**Theorem 2.**

Let  $\Gamma$  be the collection of all subsets of V such that for all A in  $\Gamma$   $v_s \notin A$ ,  $v_t \notin A$  and  $|A| > 0$ .

If  $A^*$  is the element of  $\Gamma$  such that

$$\gamma_{A^*} = \max_{A \in \Gamma} \gamma_A$$

then  $A^* \cup \{v_s\}$  and  $V - A^* - \{v_s\}$  make a partition of the OPO of the linear placement problem.

**Proof:**

For clarity, we divide the proof into two steps.

1. We modify the graph  $G(V,E)$  such that for every edge  $E_{i,j}$ ,  $i \in A^*$  and  $j \in V - A^* - \{v_s\}$ , we delete the edge  $E_{i,j}$  and add edges  $E_{i,t}$  and  $E_{s,j}$  as shown in

Fig. 5.3.

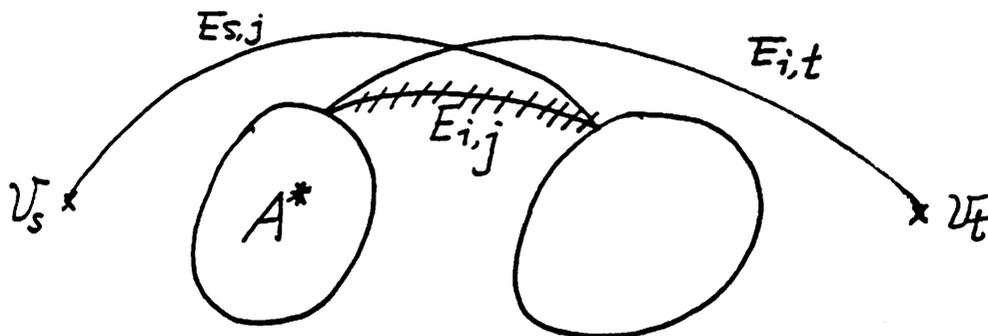


Fig. 5.3. Illustration of  $G'(V, E')$  in the proof of theorem 2.

Let this new graph be  $G'(V, E')$ . Then we make the following statement.

**Claim 1.**

$A^* \cup \{v_s\}$  and  $V - A^* - \{v_s\}$  make a partition of the OPO of  $G'$ .

2. Of the original graph, we state the following:

**Claim 2.**

If  $A^* \cup \{v_s\}$  and  $V - A^* - \{v_s\}$  make a partition of the OPO of  $G'$ , then both subsets also define a partition of the OPO of  $G$ .

Hence we view the modified graph  $G'$  in step one of the proof as the worst case. If the theorem is true for this case, then it should be true in the original graph.

**Proof (Claim 1):**

This statement is proved by contradiction. If the theorem is not true, then there exists an  $m > 1$  such that the OPO has the form of

$$v_s B_1 A_1 B_2 \dots B_m A_m B_{m+1} v_t \quad (\text{Order I})$$

where  $\bigcup_{i=1}^m A_i = A^\circ$ ,  $A_i \neq \emptyset$   $i=1,2,\dots,m$  and  $B_i \neq \emptyset$   $i=2,\dots,m$

In the following proof, we shall demonstrate that the order

$$v_s A_1 A_2 \dots A_m B_1 B_2 \dots B_{m+1} v_t \quad (\text{Order II})$$

is not worse than order I in terms of the sum of the wire lengths.

Let us denote by  $XL_i$ ,  $Xr_i$  the connectivity of  $X_i$  to the vertices on its left and on its right, respectively. Then in order II

$$\gamma_{A^\circ} = \frac{\sum_{i=1}^m (AL_i - Ar_i)}{\sum_{i=1}^m |A_i|} \quad (5)$$

and

$$\gamma_{A_1} = \frac{AL_1 - Ar_1}{|A_1|}$$

$$\gamma_{\bigcup_{i=1}^2 A_i} = \frac{\sum_{i=1}^2 (AL_i - Ar_i)}{\sum_{i=1}^2 |A_i|}$$

$$\vdots$$

$$\vdots$$

$$\gamma_{\bigcup_{i=1}^{m-1} A_i} = \frac{\sum_{i=1}^{m-1} (AL_i - Ar_i)}{\sum_{i=1}^{m-1} |A_i|} \quad (6)$$

Because  $\gamma_{A^\circ}$  is the maximum, combining Eq. (5) and Eq. (6) we have

$$\gamma_{A_1} \leq \gamma_{A^\circ} \implies \gamma_{A_1} \leq \frac{\sum_{i=2}^m (AL_i - Ar_i)}{\sum_{i=2}^m |A_i|}$$

$$\gamma_{\bigcup_{i=1}^2 A_i} \leq \gamma_{A^\circ} \implies \gamma_{\bigcup_{i=1}^2 A_i} \leq \frac{\sum_{i=3}^m (AL_i - Ar_i)}{\sum_{i=3}^m |A_i|}$$

(7)

$$\gamma_{\bigcup_{i=1}^{m-1} A_i} \leq \gamma_{A^*} \implies \gamma_{A^*} \leq \frac{Al_m - Ar_m}{|A_m|}$$

Using the same reasoning, we obtain

$$\gamma_{B_1} \leq \gamma_{A^*} \implies \frac{Bl_1 - Br_1}{|B_1|} \leq \gamma_{A^*}$$

$$\gamma_{\bigcup_{i=1}^2 B_i} \leq \gamma_{A^*} \implies \frac{\sum_{i=1}^2 (Bl_i - Br_i)}{\sum_{i=1}^2 |B_i|} \leq \gamma_{A^*}$$

(8)

$$\gamma_{\bigcup_{i=1}^m B_i} \leq \gamma_{A^*} \implies \frac{\sum_{i=1}^m (Bl_i - Br_i)}{\sum_{i=1}^m |B_i|} \leq \gamma_{A^*}$$

Now changing from order I to II, we decrease the cost by

$$\begin{aligned} & |B_1|(Al_1 - Ar_1) + \sum_{i=1}^2 |B_i|(Al_2 - Ar_2) + \dots + \sum_{i=1}^m |B_i|(Al_m - Ar_m) \\ & - \left[ \sum_{i=1}^m |A_i|(Bl_1 - Br_1) + \sum_{i=2}^m |A_i|(Bl_2 - Br_2) + \dots + |A_m|(Bl_m - Br_m) \right] \\ & = |B_1| \sum_{i=1}^m (Al_i - Ar_i) + |B_2| \sum_{i=2}^m (Al_i - Ar_i) + \dots + |B_m|(Al_m - Ar_m) \\ & - \left[ |A_1|(Bl_1 - Br_1) + |A_2| \sum_{i=1}^2 (Bl_i - Br_i) + \dots + |A_m| \sum_{i=1}^m (Bl_i - Br_i) \right] \end{aligned} \quad (9)$$

Plugging Eq. (7) and Eq. (8) in Eq. (9), we have the cost decreasing by the amount larger than

$$\begin{aligned} & \gamma_{A^*} \left[ |B_1| \sum_{i=1}^m |A_i| + |B_2| \sum_{i=2}^m |A_i| + \dots + |B_m| |A_m| \right] \\ & - \gamma_{A^*} \left[ |A_1| |B_1| + |A_2| \sum_{i=1}^2 |B_i| + \dots + |A_m| \sum_{i=1}^m |B_i| \right] \\ & = 0 \end{aligned} \quad (10)$$

This indicates order II is the OPO.

**Proof (Claim 2):**

We can prove claim 2 by contradiction. Suppose claim 2 is not true, then there exists an  $m > 1$  such that order I is OPO. We can check that, if we change from order II to order I the sum of the lengths in graph G does not decrease more than the sum of the lengths in graph G'. Now in graph G' order II is not worse than order I. Consequently, in graph G order II can not be worse than order I.  $\rightarrow\leftarrow$

Q.E.D.

**Example:**

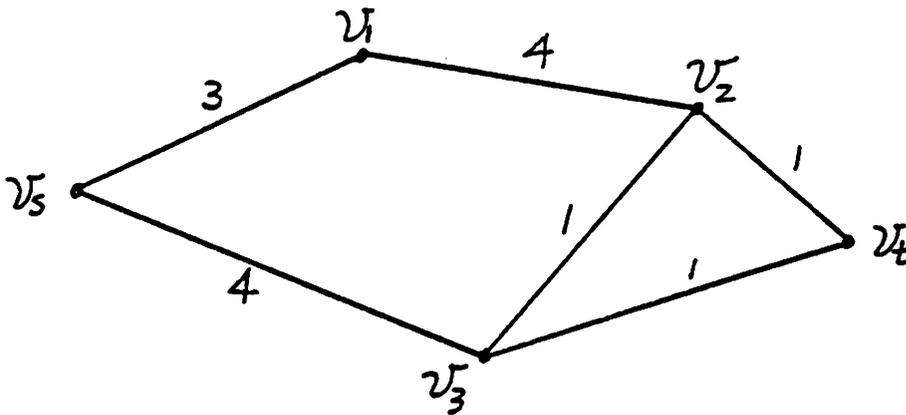


Fig. 5.4. five-vertex graph with connectivities labelled on the edges.

In Fig. 5.4, we have three vertices to be assigned. The following table lists the cost ratios with respect to different subsets. Of these, vertex  $v_3$  generates the maximal cost ratio. Thus  $v_3$  is ahead of other vertices in the

OPO.

vertex set	{1}	{2}	{3}	{1,2}	{1,3}	{2,3}	{1,2,3}
cost ratio	-1	-6	2	0.5	0.5	-1	1.67

Table 5.1.

After the graph is partitioned we have subsets of vertices  $\{S_i\}$ . For each such subset  $\{S_i\}$  we shrink all vertices on its left hand side to a new source  $v_s'$  and shrink all vertices on its right hand side to a new sink  $v_t'$ , as shown in Fig. 5.5. Let us define this new graph to be the **shrunk graph**  $G_{S_i}$ . Based on the following theorem, we can decompose the graph and deal with the shrunk graph  $G_{S_i}$  as a new linear placement problem.

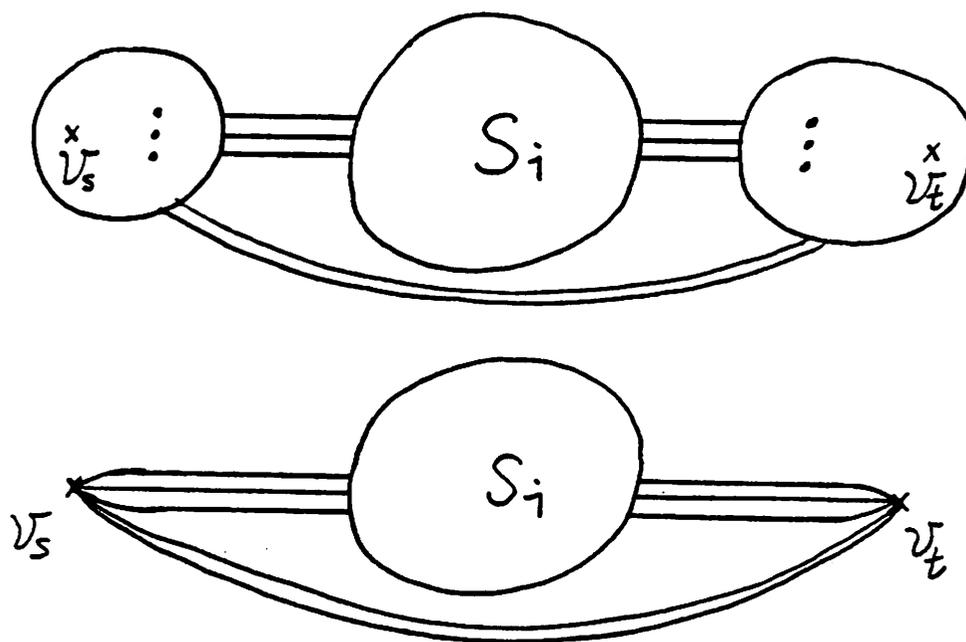


Fig. 5.5. Illustration of the shrunk graph.

**Theorem 3.**

Given a graph  $G(V,E)$ , and assume that  $A$  is a subset of  $V$  such that, in the OPO,  $A$  covers consecutive  $|A|$  slots. According to the definition of the shrunk graph, we construct graph  $G_A$ . Then, the OPO of  $A$  with respect to the shrunk graph  $G_A$  is also an optimal order of  $A$  with respect to the original graph  $G$ .

Proof:

Suppose that in the OPO, there are three subsets L, A and R with the order L A R. Let k and l be the indices of slots at both ends of set A. Then the sum of the wiring lengths is

$$\begin{aligned} \frac{1}{2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c_{i,j} l_{p(i),p(j)} &= \frac{1}{2} \sum_{i \in L} \sum_{j \in L} c_{i,j} l_{p(i),p(j)} + \frac{1}{2} \sum_{i \in A} \sum_{j \in A} c_{i,j} l_{p(i),p(j)} + \frac{1}{2} \sum_{i \in R} \sum_{j \in R} c_{i,j} l_{p(i),p(j)} \\ &+ \sum_{i \in L} \sum_{j \in A} c_{i,j} l_{p(i),p(j)} + \sum_{i \in L} \sum_{j \in R} c_{i,j} l_{p(i),p(j)} + \sum_{i \in A} \sum_{j \in R} c_{i,j} l_{p(i),p(j)} \end{aligned} \quad (11)$$

The terms that are not related to A could be represented by a constant.

$$\frac{1}{2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c_{i,j} l_{p(i),p(j)} = C + \frac{1}{2} \sum_{i \in A} \sum_{j \in A} c_{i,j} l_{p(i),p(j)} + \sum_{i \in L} \sum_{j \in A} c_{i,j} l_{p(i),p(j)} + \sum_{i \in A} \sum_{j \in R} c_{i,j} l_{p(i),p(j)}$$

Decomposing the second and third terms on the right hand side of the equation, we have

$$\begin{aligned} \frac{1}{2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c_{i,j} l_{p(i),p(j)} &= C + \frac{1}{2} \sum_{i \in A} \sum_{j \in A} c_{i,j} l_{p(i),p(j)} + \sum_{i \in L} \sum_{j \in A} c_{i,j} l_{k,p(j)} \\ &+ \sum_{i \in A} \sum_{j \in R} c_{i,j} l_{p(i),l} + \sum_{i \in L} \sum_{j \in A} c_{i,j} l_{p(i),k} + \sum_{i \in A} \sum_{j \in R} c_{i,j} l_{l,p(j)} \\ &= C + \frac{1}{2} \sum_{i \in A} \sum_{j \in A} c_{i,j} l_{p(i),p(j)} + \sum_{i \in L} \sum_{j \in A} c_{i,j} l_{k,p(j)} + \sum_{i \in A} \sum_{j \in R} c_{i,j} l_{p(i),l} \end{aligned} \quad (13)$$

This indicates the OPO of A in  $G_A$  is also the OPO of A in G.

Q.E.D.

Based on these results, we first deal with a special case: the parallel graph.

#### 5.4. Parallel graph

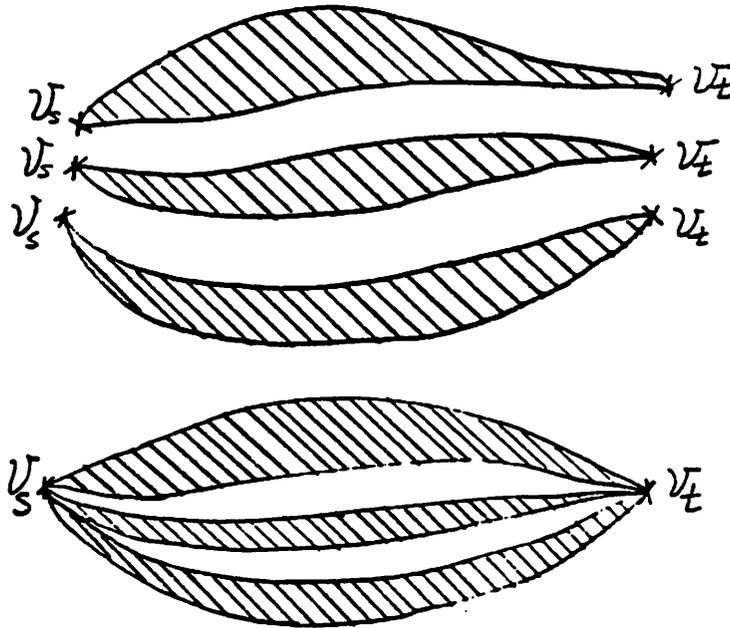


Fig. 5.6. The construction of parallel graph.

Given a set of disjoint graphs  $G_1, G_2, \dots, G_m$ , let each graph  $G_i$  contain a source and a sink for  $i=1, 2, \dots, m$ . A parallel graph is constructed by shrinking the sources and the sinks of this set of graphs to a single source and a single sink [Fig. 5.6], respectively. We denote this relation by

$$G(V, E) = \parallel_{i=1}^m G_i(V_i, E_i) \quad (14)$$

Suppose the OPOs of graphs  $G_1, G_2, \dots, G_m$  are known. Then, based on the following lemma, an algorithm is proposed to find the OPO of graph  $G$ . The lemma suggests that, from one of the parallelised graphs, we can find the vertex set which generates the maximal cost ratio.

#### Lemma 1.

Given  $G(V, E) = \parallel_{i=1}^m G_i(V_i, E_i)$ , there exist an  $A^*$  and an  $i$  in  $\{1, 2, \dots, m\}$  such that

$$A^* \subset V_i \text{ and } \gamma_{A^*} = \max_{A \in \mathcal{A}} \gamma_A$$

Proof:

This lemma is proved by contradiction. Suppose the statement is not true.

Then there exists a  $k > 1$  and a permutation function  $q(i)$  such that

$A^* = \bigcup_{i=1}^k A_{q(i)}$ , where  $A_{q(i)} \subset V_{q(i)}$ . If  $j \in \{1, 2, \dots, k\}$  is the index such that

$\gamma_{A_{q(j)}} \geq \max_{i=1}^k \gamma_{A_{q(i)}}$ , then  $\gamma_{A_{q(j)}} \geq \gamma_{A^*}$ . As a result,  $A_{q(j)}$  satisfies the conditions of the lemma.

Q.E.D.

#### 5.4.1. Algorithm

In the following, the algorithm deals with the parallel graph.

Input:

$$G(V, E) = \parallel_{i=1}^m G_i(V_i, E_i)$$

Algorithm:

1. Set  $W \equiv V$  and set  $\alpha$  an empty permutation.
2. Find a subset  $S$  of  $W$  such that  $\gamma_S = \max_{A \in \Gamma} \gamma_A$ .
3. Find OPO of  $G_S$ .
4. Append the order of  $S$  to the end of the permutation  $\alpha$ .
5. Set  $W = W - S$ , updating the graph to be shrunk graph  $G_W$ .
6. If  $W = \phi$  then stop, else go to 2.

Output:

$\alpha$ , the ordering of  $V$ .

Based on lemma 1, the subset  $S$  can be found from one of the parallelised graphs. Suppose the optimal order of graphs  $G_i$   $i=1, 2, \dots, m$  is given. The computational complexity of the algorithm is polynomial. Next we prove the output is an OPO solution.

**Theorem 4.**

The above algorithm generates an OPO of graph  $G$ .

**Proof:**

This theorem is proved by induction.

**Case 1.** The process stops at the first iteration

In this case, step 3 finds the OPO solution.

**Case 2.** The process stops at the  $k$ 'th iteration

Here, we assume the solution is the OPO at the  $k$ 'th iteration. Then we prove that the solution is the OPO after  $k+1$  iterations.

**Case 3.** The process stops at the  $k+1$ 'th iteration

In the first iteration, from theorem 2 and lemma 1,  $V-W$  and  $W$  define a partition of the OPO. Step 3 also finds the OPO of  $V-W$  in  $G$ . From case 2 above and theorem 3, the OPO of  $W$  is found in the next  $k$  iterations. From theorem 3 we know the output  $\alpha$  is the OPO solution.

Q.E.D.

Based on the algorithm and theorem 4, we state the following:

**Theorem 5.**

The OPO of  $V_i$  in  $G_i$   $i=1,2,\dots,m$  is the OPO of  $V_i$  in  $G$ .

**5.4.2. Example:**

We have a parallel graph  $G(V,E)$  formed by two chains  $G_1(V_1,E_1)$  and  $G_2(V_2,E_2)$  as shown in Fig. 5.7.

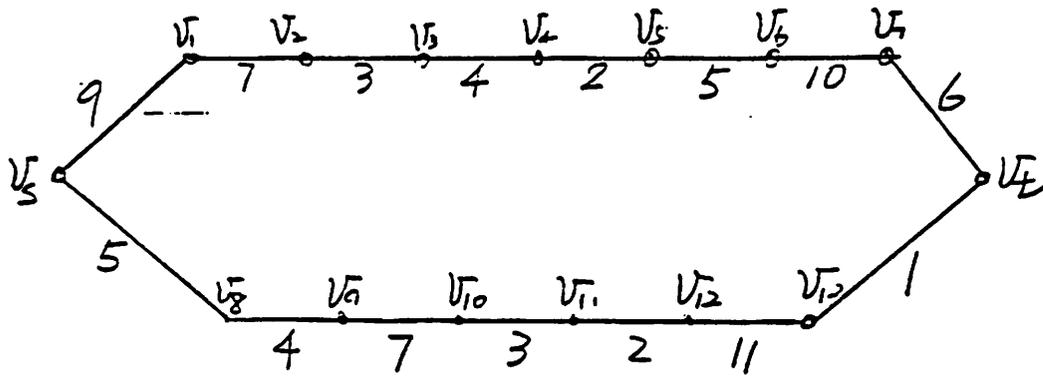


Fig. 5.7. 15-vertex example.

Since Gomory and Hu's cut trees of  $G_1$  and  $G_2$  are chains, the OPOs of  $G_1$  and  $G_2$  have the same order as the chains. Thus we can easily check the  $\gamma$  values with respect to different cut-lines as shown in Fig. 5.8.

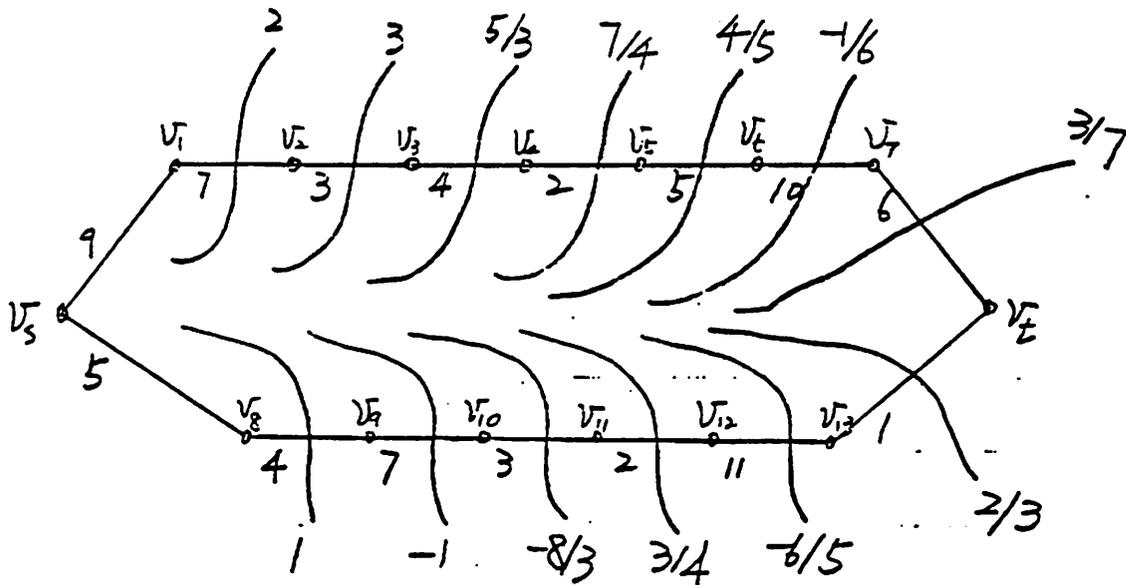


Fig. 5.8. Cost ratios with respect to different cut-lines.

Following the algorithm, in step 2 the cut-line that separates vertices  $v_s$ ,  $v_1$  and  $v_2$  from other vertices, generates the highest  $\gamma$  value. Consequently, vertices  $v_1$  and  $v_2$  are placed ahead of other vertices in the order.

In the second iteration, we update the  $\gamma$  values as Fig. 5.9. Again, by checking the  $\gamma$  value, we set vertex  $v_8$  in front of other vertices. It takes six iterations to find the OPO solution [Fig. 5.10].

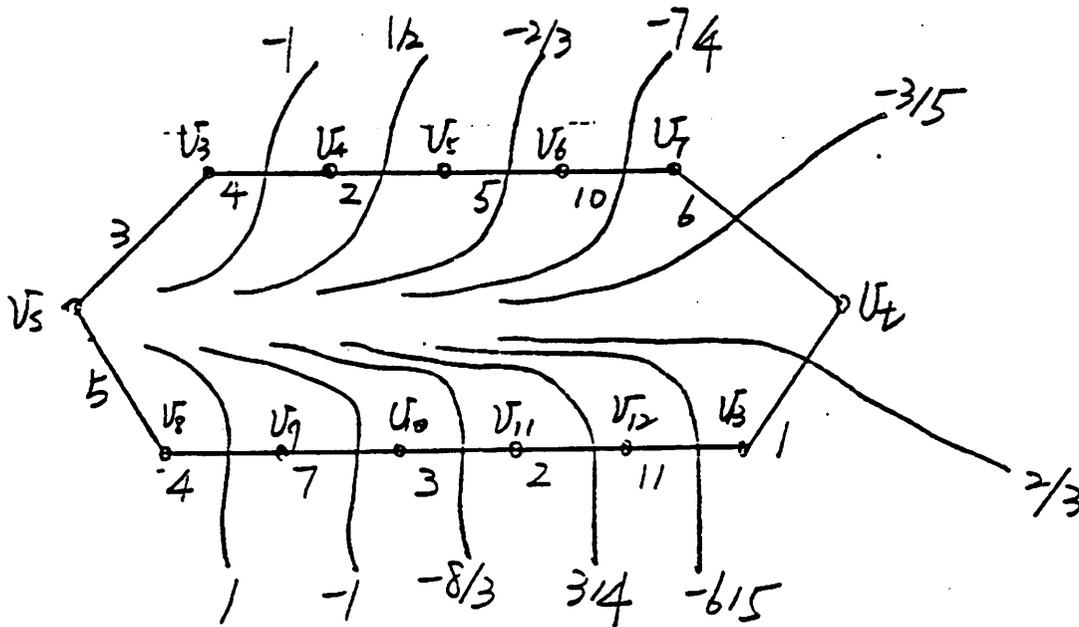


Fig. 5.9. Shrunk graph updated from Fig. 5.8.

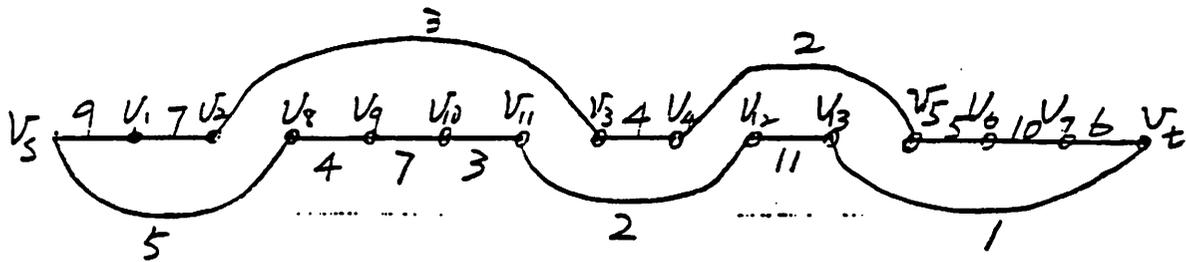


Fig. 5.10. Optimal order of 15-vertex example.

### 5.5. Arbitrary graph

In the general case, we encounter arbitrary graphs. Although, theorem 2 gives a criterion to partition the OPO, it is in general difficult to find the subset which generates the maximal cost ratio for an arbitrary graph.

The max-flow min-cut makes a partition of OPO. However, in the case that either the cut-line separates the source  $v_s$  from all other vertices or the cut-line separates  $v_t$  from all other vertices, no information of the optimal order can be obtained. In order to do further partitioning, we need a strategy which modifies the graph without disturbing the optimal order of the original graph.

We then repeat this process on each partitioned subset until graph modification fails. It is shown that this process finds the subset which generates the maximal cost ratio.

#### 5.5.1. Graph modification

Given a graph  $G(V,E)$ , we can modify the graph by

- i. Adding an edge  $E_{s,t}$  with connectivity  $a$ .
- ii. Adding an edge  $E_{i,t}$  for all  $i \neq s$  with connectivity  $b$ .
- iii. Adding an edge  $E_{s,i}$  for all  $i \neq t$  with connectivity  $c$ .
- iv. Adding an edge  $E_{i,j}$  for all  $(i,j) \in \{(i,j) | i \neq j, i \neq s, i \neq t, j \neq s, j \neq t, v_i \in V \text{ and } v_j \in V\}$  with connectivity  $d$ .
- v. Adding an edge  $E_{s,i}$  and  $E_{i,t}$  for any  $i \neq s, t$  with connectivity  $e_i$ .

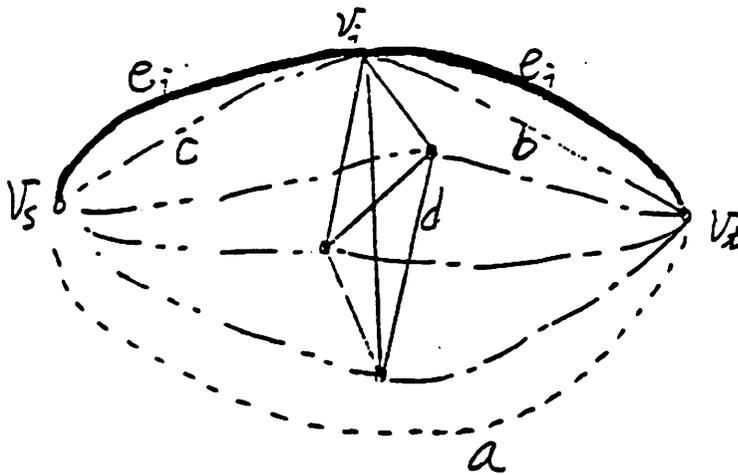


Fig. 5.11. Illustration of graph modification.

Fig. 5.11 is an illustration for the graph modification. It is obvious that the modified graph has the same OPO as the original graph.

**Constraint:**

The max-flow min-cut method is applied to the graph with non-negative connectivities.

Otherwise, the max-flow min-cut problem becomes an N-P complete problem. Therefore, we have to restrict the connectivities  $a$ ,  $b$ ,  $c$ ,  $d$  and  $e_i$  such that the modified graph does not contain a negative connectivity branch.

With regard to this constraint, a strategy setting the values of  $a$ ,  $b$ ,  $c$  and  $e_i$  is developed. This can be divided into three cases.

Case(1)  $a$  and  $e_i$ : Since the values of  $a$  and  $e_i$  do not affect the configuration of the max-flow min-cut, they are ignored.

Case(2)  $b$  and  $c$ : With reference to Fig. 5.12, line 1 separates  $\{v_s\}$  from  $V - \{v_s\}$ , line 2 separates  $\{v_t\}$  from  $V - \{v_t\}$  and line 3 separates  $|V|$  into  $m+1$  vertices and  $|V| - m - 1$  vertices.

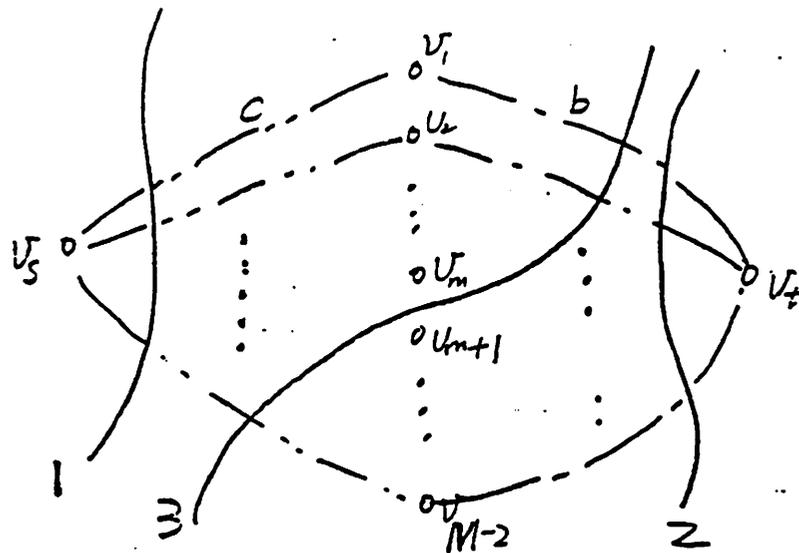


Fig. 5.12. Graph modified by adding edges with connectivities  $b$  and  $c$ .

Let us denote by  $F_1$ ,  $F_2$  and  $F_3$  the sums of connectivities of lines crossing

line 1, line 2 and line 3, respectively. After step ii and step iii are applied to modify the graph,  $F_i$  becomes  $F_i'$   $i=1, 2$  and 3.

$$\begin{aligned} F_1' &= F_1 + (|V| - 2) * c \\ F_2' &= F_2 + (|V| - 2) * b \\ F_3' &= F_3 + m * b + (|V| - m - 2) * c \end{aligned} \quad (15)$$

Thus

$$F_3' - F_1' = F_3 - F_1 + m * (b - c) \quad (16)$$

and

$$F_3' - F_2' = F_3 - F_2 + (|V| - m - 2) * (c - b) \quad (17)$$

To avoid the case that the max-flow min-cut lies on line 1 or line 2 in Fig. 5.8, the variables must be set so that  $F_3' - F_1' \leq 0$  and  $F_3' - F_2' \leq 0$ . As a result, the value of  $b - c$  is set to maximize the minimum of  $\{ F_1 + m * (c - b), F_2 - (|V| - m - 2) * (c - b) \}$ . Thus, the solution is

$$b - c = \frac{F_1 - F_2}{|V| - 2} \quad (18)$$

Case(3) d: By applying step iv of the graph modification, the following results are obtained.

$$\begin{aligned} F_1' &= F_1 \\ F_2' &= F_2 \end{aligned} \quad (19)$$

$$F_3' = F_3 + m * (|V| - m - 2) * d$$

In order to have  $F_3' - F_1' \leq 0$  and  $F_3' - F_2' \leq 0$ ,  $d$  is set to as small a value as possible. Thus,  $d$  is set to be the negative value of a maximal connectivity clique which contains all vertices but  $v_s$  and  $v_t$  before the modification.

**Example**

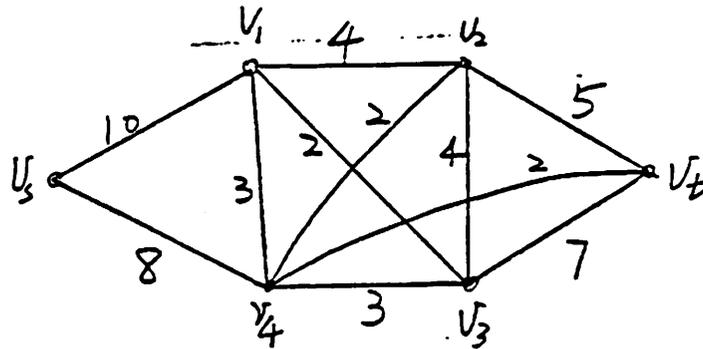


Fig. 5.13. six-vertex example from [22].

Fig. 5.13 is the six-vertex example from [22]. On the graph, we have the sums of connectivities to the source  $F_1 = 18$  and that to the sink  $F_2 = 14$ . Consequently, from Eq. 18, we set  $b-c = (18-14)/4 = 1$ . There is also a clique connecting  $v_1, v_2, v_3$  and  $v_4$  with connectivity 2. Thus we modify the graph by adding edges with connectivities  $b=1, c=0$  and  $d=-2$  [Fig. 5.14]. Fig. 5.15 is the result after graph modification.

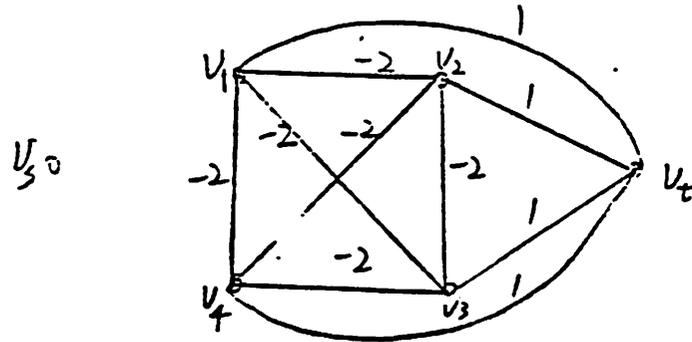


Fig. 5.14. The edges with connectivities  $b$  and  $d$  added to the graph.

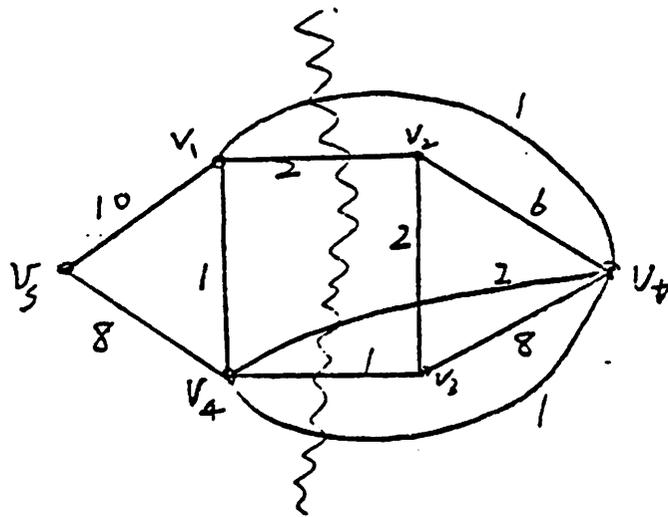


Fig. 5.15. Modified graph partitioned by max-flow min-cut line.

### 5.5.2. Algorithm

For the modified graph, we partition the vertex set with the max-flow min-cut method. Then for each obtained subset of vertices  $S_i$ , the graph  $G_{S_i}$  is constructed ( see Sec. 5.4 ). We continue this process until no vertex set  $S_i$  can be partitioned further. Finally, we find the OPO of each set  $S_i$  and concatenate the sub-sequences.

Input:

Graph  $G(V,E)$

Algorithm:

1. Set  $G(V,E)$  to be the current graph.
2. List all current graphs.
3. For each current graph:
  - i. Modify the graph.
  - ii. Partition the vertices with the max-flow min-cut.
4. If no more partitioning of  $V$  is obtained from 3, then go to 8.
5. List all sets of partitioned vertices.
6. For each partitioned vertex set  $A$ 
  - i. Construct shrunk graph  $G_A$  as stated in Section 3.
7. Go to 2.
8. Find the OPO of all partitioned vertex sets.

Output:

The order of vertices.

### 5.5.3. Example

We use the previous six-vertex example [Fig. 5.13]. After the graph is modified, the max-flow min-cut separates  $v_s$ ,  $v_1$  and  $v_4$  from other vertices.

After partitioning, we construct two graphs with respect to the two subsets of vertices [Fig. 5.16]. The OPO solution is obtained after the second level partition [Fig. 5.17]. The sum of wiring lengths is 78 which is the best result for this problem.

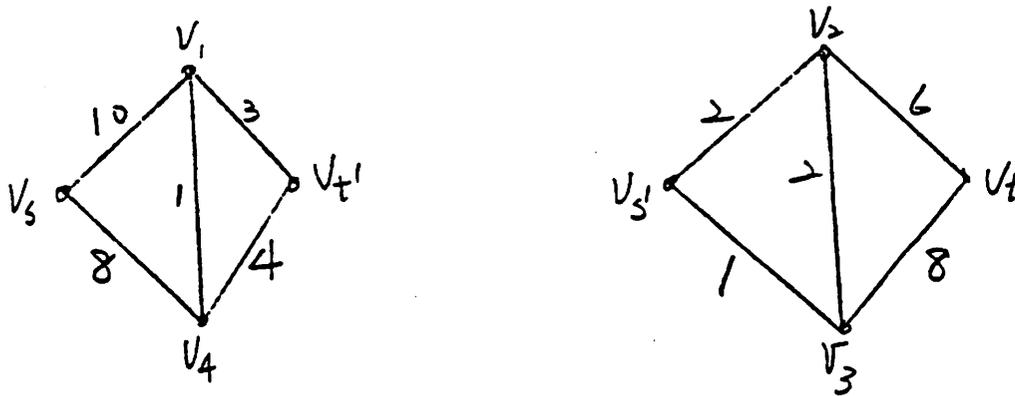


Fig. 5.16. Shrunk graphs constructed after partitioning.

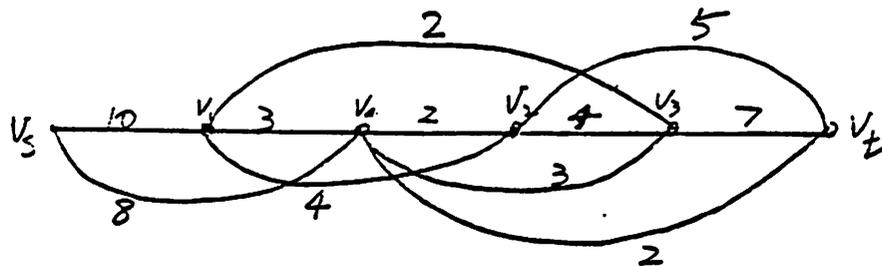


Fig. 5.17. Optimal order of six-vertex example.

#### 5.5.4. Theorems

According to theorem 1, theorem 3 and the properties of the modified graph, the output is an OPO solution. Thus we have theorem 6.

##### Theorem 6.

The result of the above algorithm is an OPO solution.

In the following theorem, we also prove that the algorithm can still find the cut-line that makes the same partition as the max-flow min-cut of the original graph.

##### Theorem 7.

In the above algorithm there exists one cut-line which makes the same partition as the max-flow min-cut of the original graph  $G(V,E)$ .

**Proof:**

Let us assume that the vertices are partitioned into  $m$  subsets  $S_1, S_2, \dots, S_m$  such that these subsets can not be partitioned into smaller subsets by further operations. From theorem 6, we know subsets  $S_1, S_2, \dots, S_m$  make

partitions of OPO. Based on the proof of theorem 1, the partitions of OPO do not contradict with the partition made by the max-flow min-cut of the original graph  $G$ . In other words, there exists a max-flow min-cut of graph  $G$  and an index  $i$  such that this max-flow min-cut divides the vertex set  $V$  through subset  $S_i$ .

In the case that the max-flow min-cut does not separate the set  $S_i$  into smaller subsets, this max-flow min-cut makes the same partition as one of the cut-lines which separate the subset  $S_i$ . Then the statement of the theorem is true.

Suppose the max-flow min-cut separates subset  $S_i$  into two smaller subsets. Then, in the shrunk graph  $G_{S_i}$ , this cut-line should also make a max-flow min-cut partition on the subset  $S_i$ . While the graph  $G_{S_i}$  can be partitioned by max-flow min-cut, based on the strategy of graph modification,  $S_i$  can also be partitioned into smaller subsets in step 3 of the algorithm. As we have assumed that  $S_i$  can not be partitioned into smaller subsets, such partitioning contradicts our assumption.  $\rightarrow\leftarrow$

Q.E.D.

Furthermore, the algorithm can also find the cut-line that partitions the vertex set  $V$  into  $A^* \cup \{v_s\}$  and  $V - A^* - \{v_s\}$ .

### Example

In example 5.3, shown in Fig. 5.17, the cut-line which cuts between  $v_4$  and  $v_2$  is the max-flow min-cut of the original graph. In this example, we have four vertices to be assigned. Thus, there are  $2^4 - 1$  different subsets of vertices. Among them,  $\{v_1, v_4\}$  generates the maximal cost ratio of 2.5. In Fig. 5.17, these two vertices are separated from other vertices by the cut-line cutting through  $v_4$  and  $v_2$ .

**Theorem 8.**

In the above algorithm there exists one cut-line which partitions  $V$  into  $A^* \cup \{v_s\}$  and  $V - A^* - \{v_s\}$  such that, in the original graph,  $\gamma_{A^*} = \max_{A \in \Gamma} \gamma_A$ .

**Proof:**

Suppose in the original graph, we have  $A^*$  and  $\gamma^*$  such that  $\gamma^* \equiv \gamma_{A^*} = \max_{A \in \Gamma} \gamma_A$ .

Without loss of generality we assume  $\gamma^* \geq 0$ . If we add edges to  $G(V, E)$  such that for all  $i \in |V| - \{v_s\} - \{v_t\}$  we add edge  $E_{i,t}$  with connectivity  $\gamma^*$ , then we can make the following statement.

**Claim:**

The max-flow min-cut of the new graph partitions  $V$  into  $A^* \cup \{v_s\}$  and  $V - A^* - \{v_s\}$ .

**Proof:**

Suppose the max-flow min-cut separates  $V$  into  $A' \cup \{v_s\}$  and  $V - A' - \{v_s\}$ .

Then the sum of the connectivities of lines crossing the cut-line is

$$\begin{aligned} & \sum_{i \in A'} \sum_{j \in V - A' - \{v_s\}} c_{i,j} + \sum_{j \in V - A' - \{v_s\}} c_{s,j} + \gamma^* |A'| \\ &= \min_{A \in \Gamma} \left[ \sum_{i \in A} \sum_{j \in V - A - \{v_s\}} c_{i,j} + \sum_{j \in V - A - \{v_s\}} c_{s,j} + \gamma^* |A| \right] \\ &= \min_{A \in \Gamma} \left[ \sum_{i \in A} \sum_{j \in V - A - \{v_s\}} c_{i,j} - \sum_{j \in A} c_{s,j} + \sum_{j \in V - \{v_s\}} c_{s,j} + \gamma^* |A| \right]. \end{aligned} \quad (20)$$

After the term  $\sum_{j \in V - \{v_s\}} c_{s,j}$  is deleted from both sides of Eq. (20), we have

$$\begin{aligned} & \sum_{j \in A'} c_{s,j} - \sum_{i \in A'} \sum_{j \in V - A' - \{v_s\}} c_{i,j} - \gamma^* |A'| \\ &= \max_{A \in \Gamma} \left[ \sum_{j \in A} c_{s,j} - \sum_{i \in A} \sum_{j \in V - A - \{v_s\}} c_{i,j} - \gamma^* |A| \right]. \end{aligned} \quad (21)$$

If Eq. (21)  $> 0$  then  $\gamma_A > \gamma^*$ . That contradicts the definition of  $\gamma^*$ . Thus Eq. (21)  $\leq 0$ .

On the other hand, when  $A=A^*$ ,  $\sum_{j \in A} c_{s,j} - \sum_{i \in A} \sum_{i \in V-A - \{v_s\}} c_{i,j} - \gamma^* |A| = 0$ . we

also know Eq. (21) can not be less than zero. Consequently, Eq. (21)

can only be zero. In other words this cut-line separates  $V$  into  $A^* \cup \{v_s\}$

and  $V-A^* - \{v_s\}$ .

Based on this claim, with the same reasoning as the proof of theorem 7 we

know that one of the cut-lines should partition  $V$  into  $A^* \cup \{v_s\}$  and

$V-A^* - \{v_s\}$ .

Q.E.D.

Since there are  $|V|-2$  vertices to be assigned, it takes at most  $|V|-3$  max-flow min-cut operations to partition all the vertices to the last level. Based on Theorem 8, lemma 2 is stated as follows:

**Lemma 2:**

It takes at most  $|V|-3$  max-flow min-cut operations to partition  $V$  into

$A^* \cup \{v_s\}$  and  $V-A^* - \{v_s\}$  such that  $\gamma_{A^*} = \max_{A \in \Gamma} \gamma_A$ .

## 5.6. Application

In the VLSI linear placement problem, there exist large numbers of components. Furthermore, many of the nets are multi-pin nets. A simple and effective model is developed here to decompose multi-pin nets into two-pin nets. Then an efficient algorithm is proposed for applications to VLSI design.

### 5.6.1. Multi-pin nets

Schuler and Ulrich[29] used a clique[Fig. 5.18a] to replace the multi-pin net. This model is independent of the order of modules. However, it is not an accurate estimate of the actual wiring length of the net. On the other hand, a chain[Fig. 5.18b] that can really represent the length of the net is very sensitive to the order of modules.

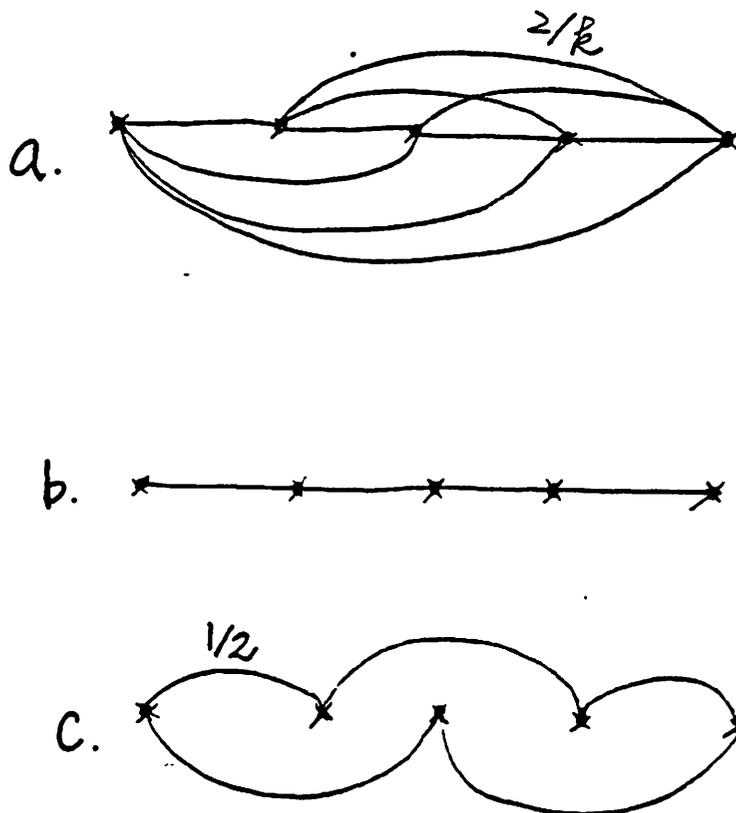


Fig. 5.18. Decomposition of multi-pin net.

- (a) Model of a clique.
- (b) Model of a chain.
- (c) Model of a loop.

Instead, we use a loop [Fig. 5.18c] to connect the multi-pin net. The connectivities of the edges in the loop are set as half the weight of the multi-pin net. The relative module locations are estimated by the network optimization method [18]. This method utilizes the resistive network analogy to solve the

optimization problem of linear placement. Based on circuit theory and optimization techniques, a relaxation scheme is developed to calculate the module locations. Although the network optimization method has the objective of minimizing the sum of the squared lengths, the method generates very good estimates of the module locations[18]. We sequence the modules according to the relative module locations. As shown in Fig. 5.18c, the loop is connected in such a way that even if the order of any pair of neighboring modules is reversed, the model is still accurate.

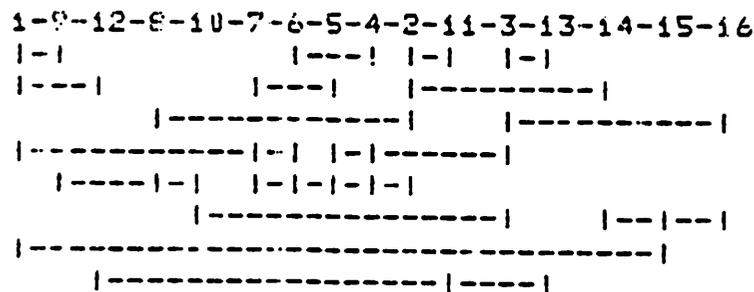


Fig. 5.19. Layout of the 16-vertex example.

### 5.6.2. Algorithm

An efficient algorithm is developed to handle large and arbitrary graphs. Wherever the max-flow min-cut method can not do further partitioning, we apply the relaxation scheme[18] to fix part of vertices at both ends of the partitioned subregion and merge the fixed vertices to vertex  $v_s$  and vertex  $v_t$ . In this way, the process can keep cutting to the last level and find the sequence of the

vertices.

Input: Net-list with two vertices fixed at both ends.

Algorithm:

1. Set the current region be the whole region.
2. Do {
  - List current regions.
  - For each region
    - Do partitioning {
      - a. Do graph modification.
      - b. Use max-flow min-cut to partition the modified graph.
      - c. If step b above fails, use the relaxation scheme to fix parts of vertices at both ends.
      - d. Update the graph, using max-flow min-cut to partition the modified graph.
    - }
  - } until all vertices are partitioned.

Output: The order of vertices.

### 5.6.3. Discussion

In VLSI design, the graph is always sparse. Based on the sparsity matrix technique, the relaxation scheme takes approximately  $O(|V|^{1.4})$  operations in each iteration. After rounding off the connectivity of the edges to rational numbers, the Ford and Fulkerson's max-flow min-cut method takes about  $O(|E|^2)$  operations. It takes at most  $|V|-3$  iterations to partition all the vertices. Hence, the computation time complexity is  $O(|V|^3)$ . However, this is only an estimation for the worst case. In the following experiments, we shall demonstrate the

efficiency of this method in terms of CPU running time.

#### 5.6.4. Experiments

Our linear placement method is implemented in C-language and tested on VAX 11/780 machine. Three examples from [28] and [30] are used to test this method. We use three measurements: sum of the lengths, sum of the square lengths, and the number of tracks required, to compare this method with the network optimization method and published results[18,29,30]. For simplicity, we set the constant,  $\beta$  at the value of 0.25 in the relaxation step[18].

Example one contains nine vertices[28]. We fix vertices two and six following the result of [29]. In this example both the linear placement method and the network optimization method generate the same result [Tab. 5.2].

	Our linear placement	Network Optimization	Kang's result
Manhattan length	50	50	50
Squared length	152	152	152
#Tracks required	11	11	11
CPU time (sec.)	2.1	1.6	N.A.

Table 5.2. Example 1: 9 vertices, 16 nets and 43 pins.

Example two contains 16 vertices[30]. To compare with the result of [30], vertices one and sixteen are fixed at the two ends [Tab. 5.3]. The ordering result is shown in Fig. 5.19 with the layout of tracks.

	Our linear placement	Network Optimization	Wing's result
Manhattan length	73	79	78
Squared length	509	451	628
#Tracks required	8	8	8
CPU time (sec.)	2.7	1.7	N.A.

Table 5.3. Example 2: 16 vertices, 17 nets and 42 pins.

Example three contains 31 vertices[28]. We fix gates OP1 and P1 at both ends following the result of [29]. It was mentioned in [29] that the best solution known has the total length 91. However, we generate an ordering with the total length of 88 [Tab. 5.4]. The ordering result is shown below

P1-B-A-C-G-H-I-D-E-F-J-Q-R-M-K-L-P2-T-S-N-O-P-Z-Y-OP2-X-W-V-U-OP3-OP1

	Our linear placement	Network Optimization	Kang's result
Manhattan length	88	102	95
Squared length	556	464	887
#Tracks required	6	8	6
CPU time (sec.)	7.3	4.6	N.A.

Table 5.4. Example 3: 31 vertices, 31 nets and 79 pins.

Therefore, our linear placement method achieves the best result in terms of sum of the lengths. On the other hand, the network optimization method achieves the best result in terms of sum of the squared lengths. Our linear

placement method also generates good placement results in terms of the number of tracks required. All these examples take less than 10 seconds of CPU time to execute.

### 5.7. Conclusion

We have explored the properties of linear placement problems. Two criteria, max-flow min-cut and cost ratio, are proposed to make partitions of the optimal order. The shrunk graph is then constructed to decompose the problem into two smaller sized problems. Based on the partitioning process, an optimal ordering method is first proposed to handle the special case: parallel graphs. However, in the general case: arbitrary graphs, a graph modification is developed to modify the graph without disturbing the optimal order of the original graph. A strategy is set so that max-flow min-cut method can make further partitioning. Based on this strategy, an optimal ordering method is also proposed for arbitrary graphs. The method is shown to find the subset which generates the maximal cost ratio. This method is extended to the applications to VLSI design, with multi-pin nets modelled by loops. The relaxation step in network optimization method is utilized to continue the partitioning. It is demonstrated with examples that this method achieves equal or better results than current methods.

## References

- [1] N. R. Quinn, Jr. and M. A. Breuer, "A Force Directed Component Placement Procedure for Printed Circuit Boards," *IEEE Trans. on Circuits and Systems*, vol. CAS-26, No. 6, pp. 377-388, June 1979.
- [2] K. J. Antreich, F. M. Johnnes and F. H. Kirsch, "A New Approach for Solving the Placement Problem Using Force Models," *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 481-486, 1982.
- [3] H. R. Charney and D. L. Plato, "Efficient Partitioning of Components," *Proc. of the 5th Annual Design Automation Workshop*, pp. 16-1 to 16-21, 1968.
- [4] D. G. Schweikert and B. W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits," *Proc. 9th Annual Design Automation Workshop*, pp. 56-62, Jun. 1972.
- [5] K. M. Hall, "An r-Dimensional-Quadratic Placement Algorithm," *Management Sci.*, vol. 17, no. 3, pp. 219-229, Nov. 1970.
- [6] C. A. Desoer and E. S. Kuh, *Basic Circuit Theory*, McGraw-Hill Book Company, 1969.
- [7] N. Jacobson, *Basic Algebra*, W.H. Freeman and Company, pp. 133-135, 1974.
- [8] R. S. Varga *Matrix Iterative Analysis*, Prentice-Hall, Inc., 1962.
- [9] L. Steinberg "The Backboard Wiring Problem: A Placement Algorithm," *SIAM Rev.*, Vol. 3, no. 1, pp. 37-50, Jan. 1961.
- [10] J. E. Stevens *Fast Heuristic Techniques for Placing and Wiring Printed Circuit Boards*, Ph.D. Thesis, Comp. Sci. Dep., Univ. of Illinois, 1972.
- [11] M. Marek-Sadowska and J.T. Li, "Global Router for Gate Array," *IEEE ICCAD*, pp. 131-132, 1983.

- [12] T. Yoshimura and E.S. Kuh, "Efficient algorithms for channel routing," Memo. UCB/ERL M80/43, Aug. 11, 1980, Electronics Research Laboratory, College of Engineering, Univ, California, Berkeley.
- [13] J.T. Li, C.K. Cheng, M. Turner, E.S. Kuh and M. Marek-Sadowska, "Automatic Layout of Gate Arrays," submitted to Custom Integrated Circuits Conference, May 1984.
- [14] S. Goto, "An Efficient Algorithm for the Two-Dimensional Placement Problem in Electrical Circuit Layout," IEEE Trans. on Circuits and Systems, vol. cas-28, no. 1, Jan. 1981.
- [15] R.L. Brooks, C.A.B. Smith, A.H. Stone and W.T. Tutte "The Dissection of Rectangles into Squares," Duke Math, Journal, Vol. 7, pp.312-340, 1940.
- [16] U. Lauther "A Min-cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," Proc. 16th Design Automation Conference, pp. 1-10, June 1979.
- [17] M.Y. Hsueh and D.O. Pederson "Computer-Aided Layout of LSI Circuit Building-Blocks," Proc. IEEE Int. Symp. on Circuits and Systems, pp. 474-477, 1979.
- [18] C.K. Cheng and E.S. Kuh "Partitioning and Placement Based on Network Optimization," IEEE Int. Conf. on Computer-Aided Design, pp 86-87, 1983.
- [19] R.L. Donze and G. Sporzynski "Masterimage Approach to VLSI Design," IEEE Computer Mag., pp. 18-25, Dec. 1983.
- [20] N.P. Chen, C.P. Hsu, E.S. Kuh, C.C. Chen and M. Takahashi "BBL: A Building-Block Layout System For Custom Chip IC Design," IEEE Int. Conf. on Computer-Aided Design, pp 40-41, 1983.
- [21] R.E. Gomory and T.C. Hu "Multi-Terminal Network Flows," J. Soc. Indust. Appl. Math. 9, pp. 551-570, 1961.

- [22] D. Adolphson and T.C. Hu "Optimal Linear Ordering," SIAM J. Appl. MATH. Vol. 25, No. 3, pp. 403-423, November 1973.
- [23] E.L. Lawler "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints," Annals of Discrete Mathematics 2, pp. 75-90, 1978.
- [24] J.B. Sidney "Decomposition Algorithms for Single-Machine Sequencing with Precedence Relations and Deferral Costs," Operations Research, Vol. 23, No. 2, pp. 283-298, March-April 1975.
- [25] T. Ohtsuki, H.M. Mori, E.S. Kuh, T. Kashiwabara and T. Fujisawa "One-Dimensional Logic Gate Assignment and Interval Graphs," IEEE Trans on Circuits and Systems, pp 675-683, September 1979.
- [26] L.R. Ford, Jr. and D.R. Fulkerson, Flows in Networks, Princeton University Press, 1962. Canad. J. Math., 8, 1956.
- [27] B.W. Kernighan and S. Lin "An Efficient Procedure for Partitioning Graphs," Bell Syst. Tech, J, pp. 291-307, Feb. 1970.
- [28] D.M. Schuler and E.G. Ulrich "Clustering and Linear Placement," Proc. 9th Design Automation Workshop, pp. 50-56, 1972.
- [29] S. Kang "Linear Ordering and Application to Placement," Proc. 20th Design Automation Conference, pp 457-464, 1983.
- [30] O. Wing "Interval-Graph-Based Circuit Layout," IEEE Int. Conf. on Computer-Aided Design, pp 84-85, 1983.