Floorplan Design of VLSI Circuits

Arnau Abella[1]

[1]Universitat Politècnica de Barcelona

December 30, 2020

Table of Contents

Introduction

Floorplan design

It is the problem of placing a given set of circuits modules in the plane to minimize a weighted sum of the following two quantities:

- the **area** of the bounding rectangle containing all the modules
- an estimation of the total interconnection **wire length**

Module

A module can be classified as:

- **rigid**, if its shape and dimensions are rigid (e.g. library macrocells)
- **flexible**, if its shape and dimensions are not fixed (assume rectangular modules)

Floorplan design

It is the problem of placing a given set of circuits modules in the plane to minimize a weighted sum of the following two quantities:

- the **area** of the bounding rectangle containing all the modules
- an estimation of the total interconnection **wire length**

Module

A module can be classified as:

- **rigid**, if its shape and dimensions are rigid (e.g. library macrocells)
- **flexible**, if its shape and dimensions are not fixed (assume rectangular modules)

Floorplan I

A **floorplan** for n given modules (named $1, 2, \ldots, n$) consist of an enveloping rectangle R subdivided by horizontal and vertical line segments into n or more nonoverlapping rectilinear regions.

The **aspect ratio** of R must be between two given numbers p and q.

We are given an $n \times n$ **interconnection matrix** $C = (c_{ij})_{n \times n}$ with $c_{ij} \geq 0, 1 \leq i, j \leq n$ which provides information on the wiring density between each pair of modules. The *distance* between two regions is the Manhattan distance between their centers. For every pair of modules i and j, let d_{ij} be the distance between regions i and j.

Floorplan II

Let A be the area of R.

We use $W = \sum_{1 \leq i,j \leq n} c_{ij} d_{ij}$ as an estimate of the total interconnection wire length.

We use $A + \lambda W$ to measure the **quality** of the floorplan, where λ is a user-specified constant.

Table of Contents

The original work

This work is based on *"Floorplan Design of VLSI Circuits"* [Wong89].

The algorithm uses **Polish expressions** to represent floorplans and employ a search method called **simulated annealing** [Ki83]. The final solution is derived in two stages:

- first determine the relative positions of the modules using primarily interconnection information,
- then, use the area and shape information to minimize the area of the bounding rectangle

The original work

This work is based on *"Floorplan Design of VLSI Circuits"* [Wong89].

The algorithm uses **Polish expressions** to represent floorplans and employ a search method called **simulated annealing** [Ki83]. The final solution is derived in two stages:

- first determine the relative positions of the modules using primarily interconnection information,
- then, use the area and shape information to minimize the area of the bounding rectangle

The original work

This work is based on *"Floorplan Design of VLSI Circuits"* [Wong89].

The algorithm uses **Polish expressions** to represent floorplans and employ a search method called **simulated annealing** [Ki83]. The final solution is derived in two stages:

- first determine the relative positions of the modules using primarily interconnection information,
- then, use the area and shape information to minimize the area of the bounding rectangle

Rectangular Modules I

Rectangular floorplan

A *rectangular floorplan* is a floorplan where all the regions are rectangles.

We shall only consider rectangular floorplans.

Let $(A_1, r_1, s_1), (A_2, r_2, s_2), \ldots, (A_n, r_n, s_n)$ be a list of n triplets of numbers corresponding to the n given modules. The triplet of numbers (A_i, r_i, s_i), with $r_i \leq s_i$, specifies the area and the limits of the allowed aspect ratio for module i.

Rectangular Modules II

Each module can also have a *fixed* or *free* orientation (90° rotations are allowed). Let O_1 be the set of modules with fixed orientation, and O_2 be the set of modules with free orientation. Let w_i be the width and h_i be the height of module i, we must have:

$$w_i h_i = A_i \tag{1}$$

$$r_i \leq h_i/w_i \leq s_i \quad \text{if } i \in O_1 \tag{2}$$

$$r_i \leq h_i/w_i \leq s_i \text{ or } 1/s_i \leq h_i/w_i \leq 1/r_i \quad \text{if } i \in O_2 \tag{3}$$

Slicing Floorplans

Slicing floorplan

A *slicing floorplan* is a rectangular floorplan with n basic rectangles that can be obtained by recursively cutting a rectangle into smaller rectangles (see Figure 1(a)). It can be represented by an oriented rooted binary tree, called *slicing tree* (see Figure 1(b)).

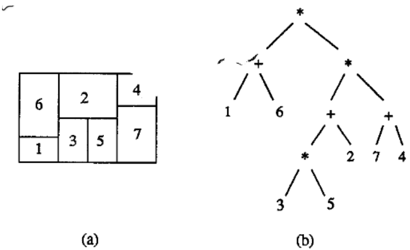

Figure: Slicing floorplan and its slicing tree representation.

Slicing Floorplans

No dimensional information is given by a slicing tree. Note that for a given slicing floorplan, there may be more than one slicing-tree representation.

Slicing Structure

Let A and B be two slicing floorplans. We define $A \sim B$ iff they have the same slicing tree representation. The equivalence relation $\sim$ partitions the set of slicing floorplans into equivalence classes. Each equivalence class of slicing floorplans with n basic rectangles is called a **slicing structure**.

Skewed slicing tree

A *skewed slicing tree* is a slicing tree in which no node and its right son have the same label in $\{*, +\}$ (see Figure 2).

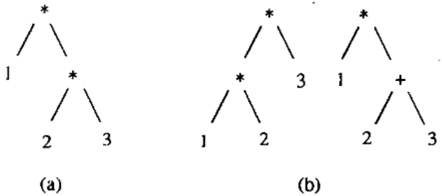

Figure: (a) A nonskewed slicing tree; (b) skewed slicing tree.

Solution Space

Lemma

There is a $1-1$ correspondence between all skewed slicing trees with n leaves and all slicing structures with n basic rectangles.

Instead of using the set of all slicing floorplans as the solution space, we can use the set of all slicing structures as the solution space. This **substantially reduces** the size of the solution space.

Solution Space

Polish expression

We use a representation of slicing structures called **normalized Polish expressions** that are particularly suitable for the method of simulated annealing.

Lemma

There is a $1-1$ *correspondence between the set of normalized Polish expressions of length* $2n-1$ *and the set of skewed slicing trees with* n *leaves.*

Solution Space

Theorem

There is a $1-1$ *correspondence between the set of normalized Polish expressions of length* $2n-1$ *and the set of slicing structures with* n *basic rectangles.*

We use the set of normalized Polish expressions as the **solution space**.

Polish Expression

Let $\alpha = \alpha_1\alpha_2\cdots\alpha_{2n-1}$ be a normalized Polish expression. Note that α can also be written as $c_0\pi_1c_2\pi_2c_2\cdots c_{n-1}\pi_nc_n$, where $\pi_1, \pi_2, \ldots, \pi_n$ is a permutation of $1, 2, \ldots, n$, the c_i's are chains. and $\sum_i l(c_i) = n-1$.

$$\underset{\pi_1}{1}\ \underset{\pi_2}{2}\ \underset{\pi_3}{3}\ \underset{c_3}{\underline{*\ +}}\ \underset{\pi_4}{5}\ \underset{\pi_5}{4}\ \underset{c_5}{\underline{+\ *}}$$

$c_0 = c_1 = c_2 = c_4 =$ the empty sequence.

Figure: Chains in a Polish expression.

Polish Expression

Two operands in α are said to be *adjacent* iff they are consecutive elements in $\pi_1 \cdots \pi_n$. An operand and operator are said to be *adjacent* iff they are consecutives elements in $\alpha_1 \alpha_2 \cdots \alpha_{2n-1}$.

We define three types of moves, $M1, M2,$ and $M3$, that can be used to modify a given normalized Polish expression (see Figure 3):

- $M1$. Swap two adjacent operands.
- $M2$. Complement a chain of nonzero length.
- $M3$. Swap two adjacent operand and operator.

Polish Expression

Two operands in α are said to be *adjacent* iff they are consecutive elements in $\pi_1 \cdots \pi_n$. An operand and operator are said to be *adjacent* iff they are consecutives elements in $\alpha_1\alpha_2 \cdots \alpha_{2n-1}$.

We define three types of moves, $M1, M2,$ and $M3$, that can be used to modify a given normalized Polish expression (see Figure 3):

- $M1$. Swap two adjacent operands.
- $M2$. Complement a chain of nonzero length.
- $M3$. Swap two adjacent operand and operator.

Polish Expression

Two operands in α are said to be *adjacent* iff they are consecutive elements in $\pi_1 \cdots \pi_n$. An operand and operator are said to be *adjacent* iff they are consecutives elements in $\alpha_1 \alpha_2 \cdots \alpha_{2n-1}$.

We define three types of moves, $M1, M2,$ and $M3$, that can be used to modify a given normalized Polish expression (see Figure 3):

- $M1$. Swap two adjacent operands.
- $M2$. Complement a chain of nonzero length.
- $M3$. Swap two adjacent operand and operator.

Two operands in α are said to be *adjacent* iff they are consecutive elements in $\pi_1 \cdots \pi_n$. An operand and operator are said to be *adjacent* iff they are consecutives elements in $\alpha_1 \alpha_2 \cdots \alpha_{2n-1}$.

We define three types of moves, $M1, M2,$ and $M3$, that can be used to modify a given normalized Polish expression (see Figure 3):

- $M1$. Swap two adjacent operands.
- $M2$. Complement a chain of nonzero length.
- $M3$. Swap two adjacent operand and operator.

Polish Expression

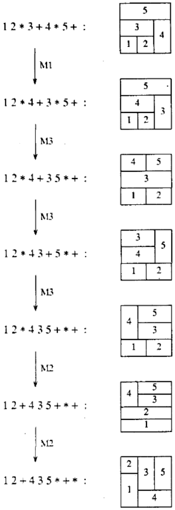

Figure: Illustration of the moves.

Cost Function

Let α be a normalized Polish expression. The expression α represents a set S_α of equivalent slicing floorplans. Let f_α be a floorplan in S_α with minimum area. Let $A(\alpha)$ and $W(\alpha)$ be the area and the total wirelength of f_α, respectively. The cost function we use is $\Psi(\alpha) = A(\alpha) + \lambda W(\alpha)$.

We can efficiently compute $\Psi(\alpha)$ for a given normalized Polish expression α.

Cost Function

Definition

Let Γ be a continuous curve on the plane. Γ is a **shape curve** if it satisfies the following conditions:

1. It is decreasing and lies completely in the first quadrant
2. $\exists k > 0$ such that all lines of the form $x = a, a > k$, intersect Γ, and
3. $\exists k > 0$ such that all lines of the form $y = b, b > k$, intersect Γ.

Definition

Let Γ and Λ be two shape curves. We define $\Gamma + \Lambda$ to be the curve $\{(u, v + w) | (u, v) \in \Gamma \text{ and } (u, w) \in \Lambda\}$ and define $\Gamma * \Lambda$ to be the curve $\{(u + v, w) | (u, w) \in \Gamma \text{ and } (v, w) \in \Lambda\}$. It is easy to see that $\Gamma + \Lambda$ and $\Gamma * \Lambda$ are also shape curves.

Cost Function

Definition

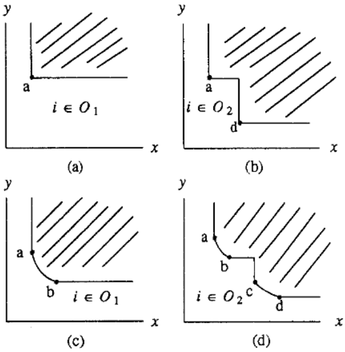

Figure: Shape curves for different shape constraints.

Cost Function

Area Computation

Let T_α be the slicing tree corresponding to α. For each node v in T_α, the subtree rooted at v defines a slicing structure R_v. Let Γ_v be the shape curve representing the shape constraints for R_v.

For every three node u, v, w in T_α with v being the father of u and w, Γ_v is either $\Gamma_u * \Gamma_w$ or $\Gamma_u + \Gamma_w$ depending on whether v is $*$ or $+$.

Once we have computed all the Γ_v's, we can compute the area measure $A(\alpha)$ from Γ_r where r is the root of the T_α.

Cost Function

Area Computation

Let T_α be the slicing tree corresponding to α. For each node v in T_α, the subtree rooted at v defines a slicing structure R_v. Let Γ_v be the shape curve representing the shape constraints for R_v.

For every three node u, v, w in T_α with v being the father of u and w, Γ_v is either $\Gamma_u * \Gamma_w$ or $\Gamma_u + \Gamma_w$ depending on whether v is $*$ or $+$.

Once we have computed all the Γ_v's, we can compute the area measure $A(\alpha)$ from Γ_r where r is the root of the T_α.

Cost Function

Area Computation

Let T_α be the slicing tree corresponding to α. For each node v in T_α, the subtree rooted at v defines a slicing structure R_v. Let Γ_v be the shape curve representing the shape constraints for R_v.

For every three node u, v, w in T_α with v being the father of u and w, Γ_v is either $\Gamma_u * \Gamma_w$ or $\Gamma_u + \Gamma_w$ depending on whether v is $*$ or $+$.

Once we have computed all the Γ_v's, we can compute the area measure $A(\alpha)$ from Γ_r where r is the root of the T_α.

Cost Function

Wire Length Computation

Since T_α is a slicing tree representation of f_α, each node v in T_α corresponds to a rectangle K_v in f_α. Let (x_v, y_v) be the dimensions of K_v. After we have computed (x_r, y_r) for K_r where r is the root of T_α, we can recursively compute the dimensions of all the basic rectangles (see [Wong89]).

Let (C_v^x, C_v^y) be the coordinates of the center of K_v. It follows that the d_{ij}'s (the distances between the basic rectangles) and consequently $W(\alpha)$ can be easily computed from the (c_v^x, c_v^y)'s.

Cost Function

Wire Length Computation

Since T_α is a slicing tree representation of f_α, each node v in T_α corresponds to a rectangle K_v in f_α. Let (x_v, y_v) be the dimensions of K_v. After we have computed (x_r, y_r) for K_r where r is the root of T_α, we can recursively compute the dimensions of all the basic rectangles (see [Wong89]).

Let (C_v^x, C_v^y) be the coordinates of the center of K_v. It follows that the d_{ij}'s (the distances between the basic rectangles) and consequently $W(\alpha)$ can be easily computed from the (c_v^x, c_v^y)'s.

Cost Function

Incremental Computation of Cost Function

For a given normalized Polish expression, the shape curves associating with the nodes of its slicing tree are needed in both the area and wire length computation.

In our simulated annealing algorithm, each move leads to only a minor modification of the Polish expression currently being examined.

Let $\alpha' = \alpha'_1 \alpha'_2 \cdots \alpha'_{2n-1}$ be the Polish expression obtained from $\alpha = \alpha_1 \alpha_2 \cdots \alpha_{2n-1}$ after a move. In general, $\Gamma_i = \Gamma'_i$ for many i's. Therefore, in computing the cost for α', we need only update those shape curves that are changed.

Cost Function

Incremental Computation of Cost Function

For a given normalized Polish expression, the shape curves associating with the nodes of its slicing tree are needed in both the area and wire length computation.

In our simulated annealing algorithm, each move leads to only a minor modification of the Polish expression currently being examined.

Let $\alpha' = \alpha'_1 \alpha'_2 \cdots \alpha'_{2n-1}$ be the Polish expression obtained from $\alpha = \alpha_1 \alpha_2 \cdots \alpha_{2n-1}$ after a move. In general, $\Gamma_i = \Gamma'_i$ for many i's. Therefore, in computing the cost for α', we need only update those shape curves that are changed.

Cost Function

Incremental Computation of Cost Function

For a given normalized Polish expression, the shape curves associating with the nodes of its slicing tree are needed in both the area and wire length computation.

In our simulated annealing algorithm, each move leads to only a minor modification of the Polish expression currently being examined.

Let $\alpha' = \alpha'_1 \alpha'_2 \cdots \alpha'_{2n-1}$ be the Polish expression obtained from $\alpha = \alpha_1 \alpha_2 \cdots \alpha_{2n-1}$ after a move. In general, $\Gamma_i = \Gamma'_i$ for many i's. Therefore, in computing the cost for α', we need only update those shape curves that are changed.

Cost Function

Theorem

Let α' be the Polish expression obtained from α after a move. The shape curves Γ_i's and the shape curves Γ'_i's differ only at a set of vertices that lie along one or two path in each of T_α and $T_{\alpha'}$

Figure 6 illustrates the statement in Theorem 7.

Cost Function

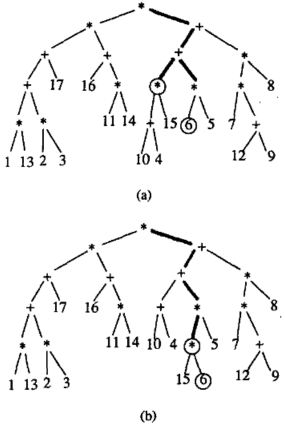

Figure: The path(s) in T_α and $T_{\alpha'}$ after a move.

Table of Contents

Implementation

The algorithm is implemented in *haskell* (compiler *GHC 8.8.4*). The executable can be build using *cabal*, *stack* or *nix*.

Apart from the algorithm, there is also a command-line interface executable with a custom problem file format with its corresponding parser. The executable allows to generate random instances for testing.

The code is public and available at `https://github.com/monadplus/floorplanning`.

Implementation

- app/
  - Main.hs (111 lines)
- src/
  - Floorplan.hs (51 lines)
  - Floorplan/
    - PolishExpression.hs (376 lines)
    - Pretty.hs (61 lines)
    - Problem.hs (226)
    - Report.hs (51 lines)
    - SimulatedAnnealing.hs (590 lines)
    - SlicingTree.hs (56 lines)
    - Types.hs (174 lines)
- test/
  - Spec.hs (22 lines)
  - Test/
    - PolishExpression.hs (121 lines)
    - Pretty.hs (77 lines)
    - Problem.hs (57 lines)
    - SimulatedAnnealing.hs (326 lines)

Table of Contents

Results

Problem	n	λ_w	A_{min}	Final solution A	Final solution W
P1	10	0.5	33.0	35.0	53.0
P1	10	1.0	33.0	35.0	46.0
P2	15	0.5	144.0	153.0	179.5
P3	20	0.5	92.0	108.0	151.5
P3	20	1.0	92.0	117.0	118.5
P4	30	0.5	199.0	231.0	1037.5
P5	40	1.0	325.0	400.0	1895.5
P6	50	1.0	445.0	529.0	4150.0

Table: Experimental results

Table of Contents

Bibliography

D. F. Wong and C. L. Liu.
Floorplan Design of VLSI Circuits.
Algorithmica, 4: 263–291, 1989.

S. KirkPatrick and C.D. Gelatt and M. P. Vecchi.
Optimization by Simulated Annealing.
Science, 220, 671–680, 1983.