# EST8717: Algorithms and Machine Learning in VLSI Physical Design
# VLSI物理设计及自动化算法

## Project #3 -
## Color Balancing for Double/Triple/Multiple Patterning

Yongfu Li
Dept. of Micro/Nano Electronics
Shanghai Jiao Tong University

# Color Balancing Project

## Part I

# Purpose of Color Balancing Solution

**Problem**

- The traditional lithography using 193 nm wavelength light cannot print patterns beyond 14nm Technology nodes.

**Solution**

- Foundry provides a scheme employing extra masks (multiple patterning technology) to make existing lithography work beyond 14nm Technology nodes.

**How?**

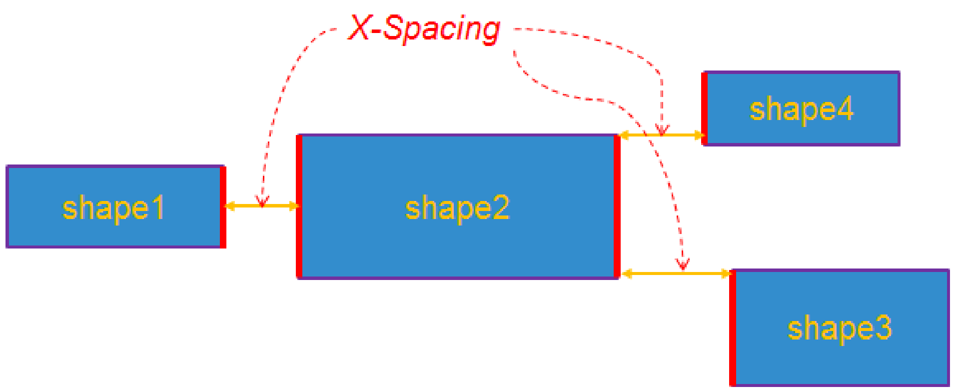
- Double patterning technology (DPT) is a method of breaking up a layout so that sub-resolution patterns are separated onto two distinct masks. These two masks are exposed and processed sequentially to create the original design patterns by composing the layout features obtained from the independent patterning steps.

# Purpose of Color Balancing Solution

**Details**

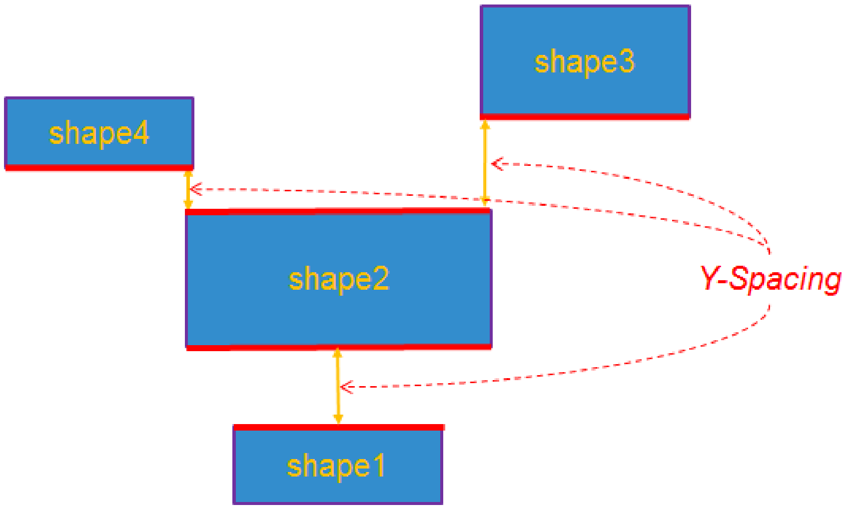
- Patterns being assigned the same color will be on the same mask.
- Non-uniform color (pattern) density on a mask can cause more pattern distortion and create more hotspots, while a balanced coloring would allow more space for scattering bar insertion during OPC (optical proximity correction) that leads to better patterning quality.
- Thus, balanced and uniform color density is preferred during layout decomposition.

# Terminology




- Shape: A shape is a rectangle which is also a coloring unit in layout. This implies that a shape can only be assigned at most one color. If a shape has a color, it is called a *colored shape*. Otherwise, it is called a *non-colored shape*.
- *X-Spacing*: If a vertical edge of a shape can see a vertical edge of another shape in X-coordinate direction, the distance between the two vertical edges is *X-Spacing*. If *X-Spacing* is < α, the two shapes must have different colors. Here, α is called *minimum X-Spacing*.

# Terminology




- *Y-Spacing*: If a horizontal edge of a shape can see a horizontal edge of another shape in Y-coordinate direction, the distance between the two horizontal edges is *Y-Spacing*. If *Y-Spacing* is < β, the two shapes must have different colors. Here, β is called *minimum Y-Spacing*.

# Terminology

- **Coloring Graph:** A coloring graph is a connected graph where a vertex represents a shape and an edge is created between two vertices (shapes) whose X-Spacing is $< \alpha$ or Y-Spacing is $< \beta$. That is, the two shapes must be assigned different colors.
- **Color Conflict:** If there exists an odd cycle in a coloring graph, the coloring graph is not 2-colorable. None of the vertices in a non-2-colorable graph will be assigned a color.
- **Coloring Bounding Box:** A smallest bounding box that contains all colored shapes. Note that there is only one coloring bounding box for a given layout design. A coloring bounding box may contain non-colored shapes.
- **Color Density Window:** It is a square inside a coloring bounding box.
- **Color Density Window Size ($\omega$):** It is the width and height of a color density window.
- **Color Density:** it is defined as  where C is the total area of colored shapes of the same color in a color density window (excluding the portion outside the color density window) and U is the area of the color density window. Keep two digits below the floating point and use rounding to keep hundredth precision. If a colored shape is not completely inside a color density window, only the area contained in the color density window is counted when color density is calculated.

# Problem Specification

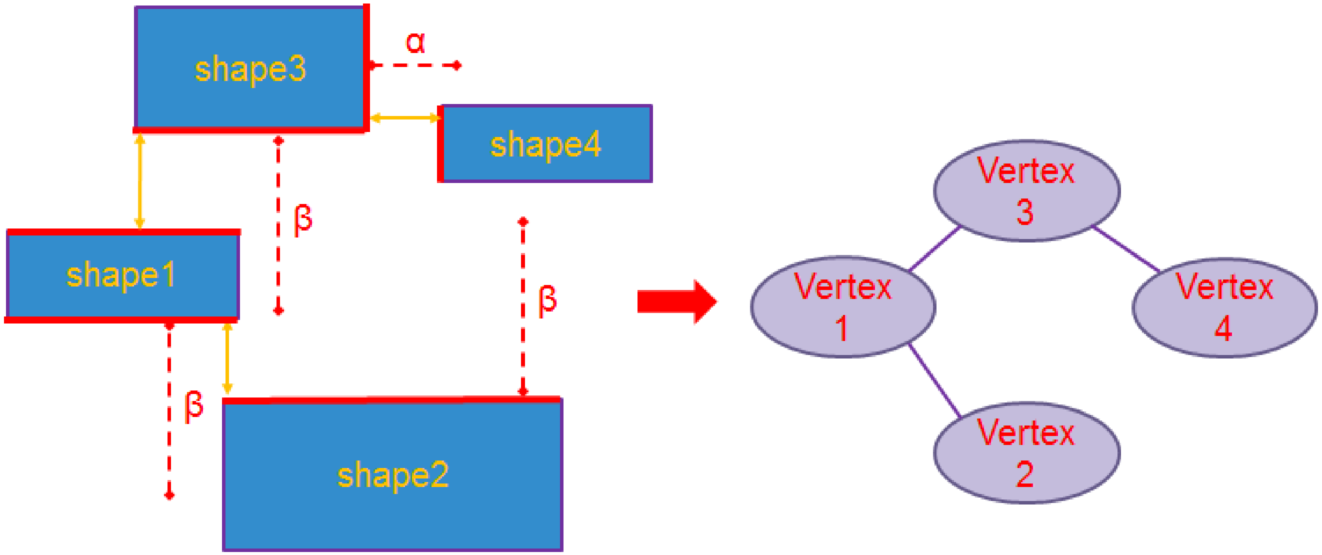
Given a layout design which contains only non-abutting and non-overlapping shapes, minimum X-Spacing a, minimum Y-Spacing b, and color density window size ω, color all shapes in the design using 2 colors, *color-A* and *color-B*. If a coloring graph is not 2-colorable, all of its vertices must not be colored. The objective is to minimize the difference between color-A density and color-B density as much as possible for all color density windows.

# Build Coloring Graph

Given any two shapes, if they satisfy (*X-Spacing* < α) or (*Y-Spacing*< β), generate an edge connecting the two vertices corresponding to these two shapes.

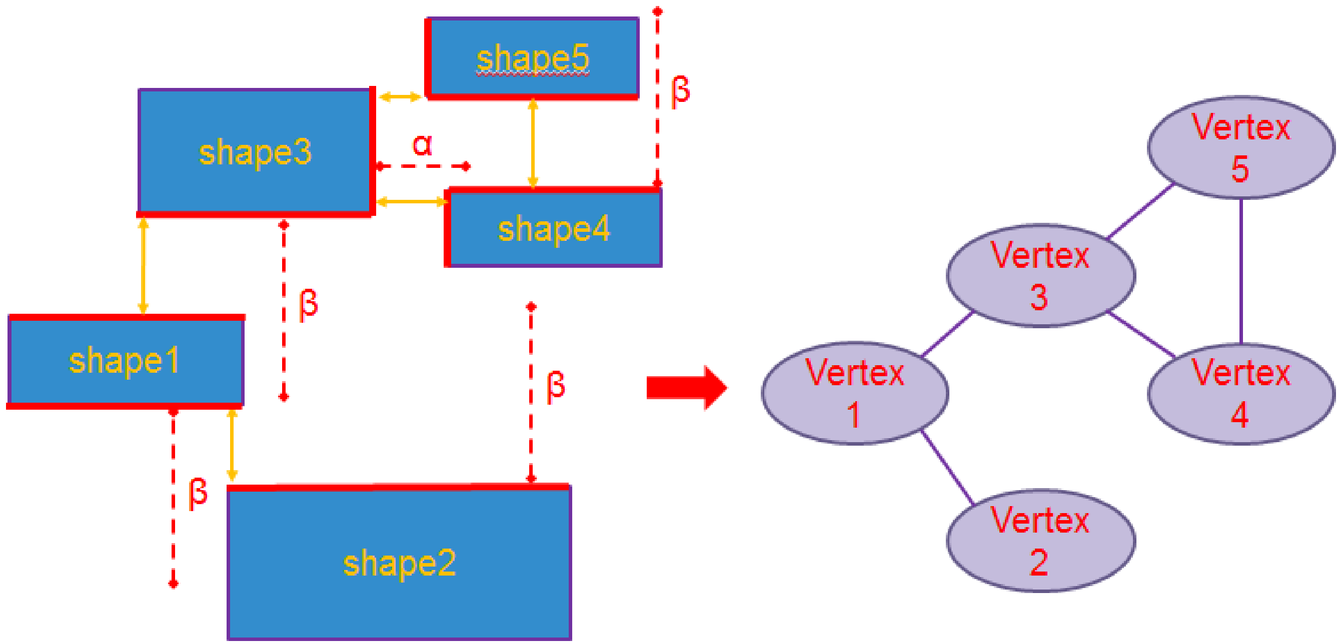
# Build Coloring Graph

**Example One:** Because shape3 and shape1 have their Y-Spacing < β, we generate an edge connecting Vertex 3 and Vertex 1. Similarly, we generate an edge connecting Vertex 1 and Vertex 2. Because shape3 and shape4 have their X-Spacing < α, we generate an edge connecting Vertex 3 and Vertex 4.



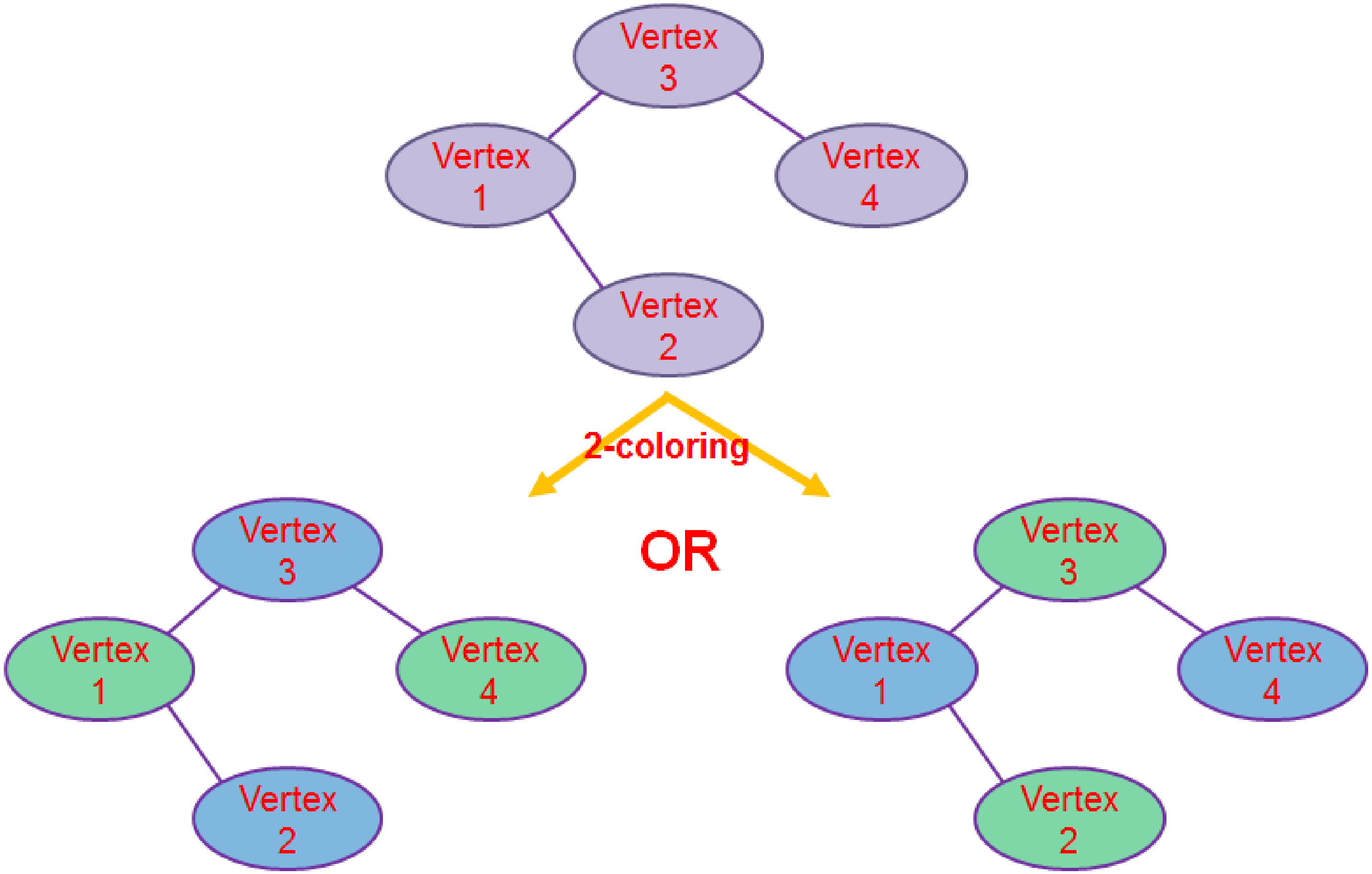
# Build Coloring Graph

**Example Two:** Because shape3 and shape1 have their *Y-Spacing* < β, we generate an edge connecting Vertex 3 and Vertex 1. Similarly, we generate an edge connecting Vertex 1 and Vertex 2 and an edge connecting Vertex 5 and Vertex 4 . Because shape3 and shape4 have their *X-Spacing* < α, we generate an edge connecting Vertex 3 and Vertex 4. Similarly, we generate an edge connecting Vertex 3 and Vertex 5.



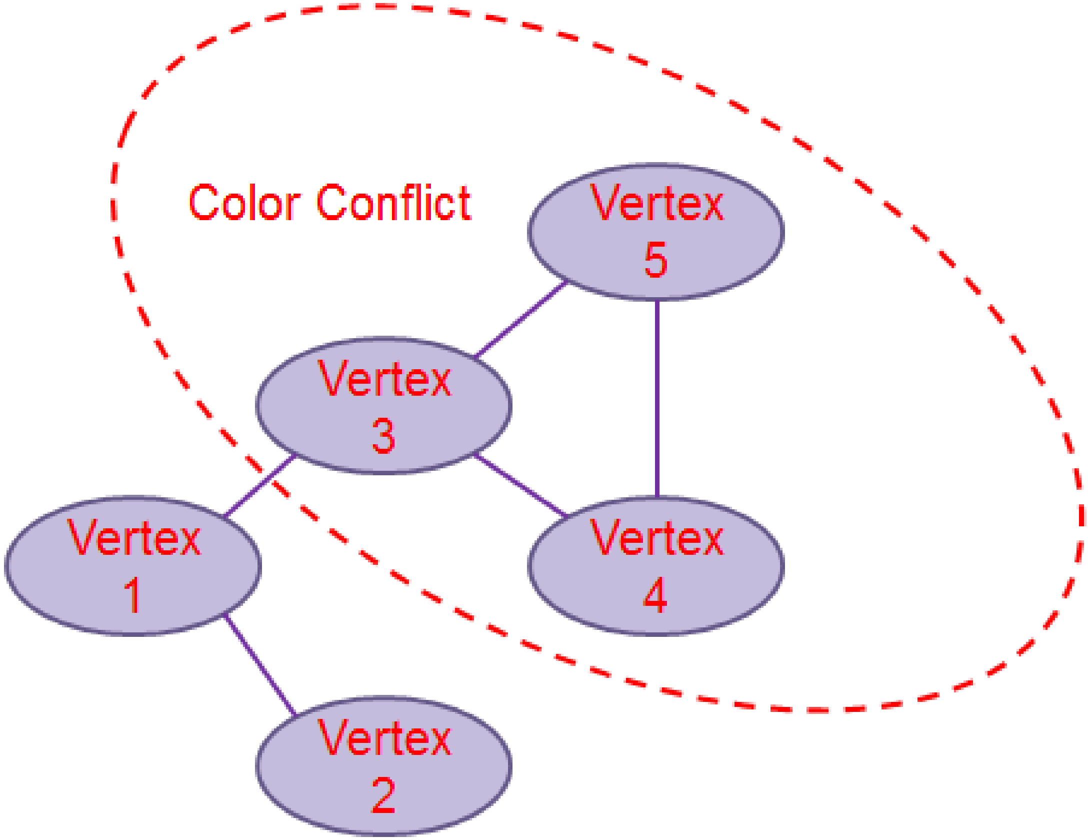
# Coloring

Any two adjacent vertices must have different colors. If one has color-A, the other must have color-B.

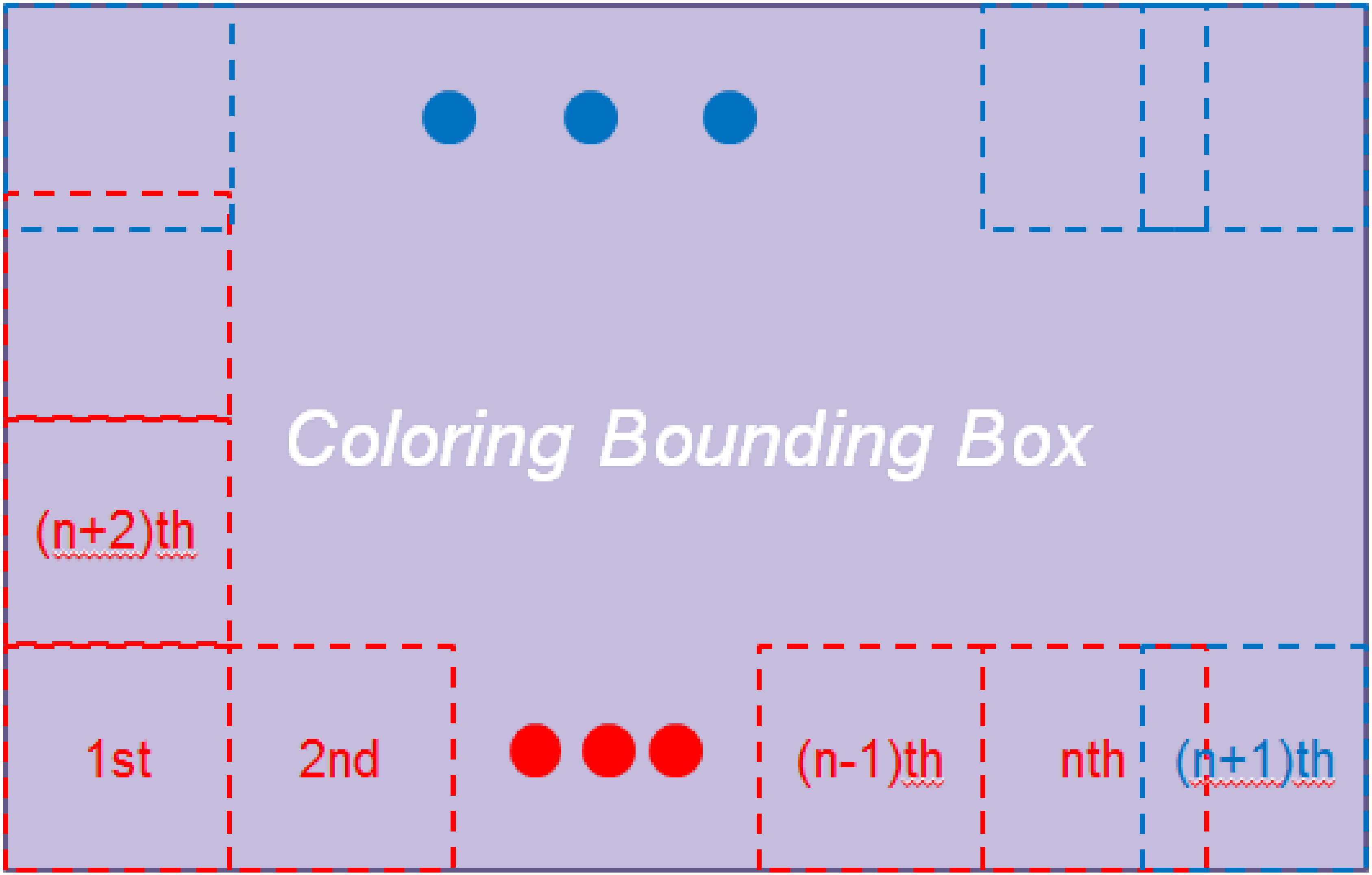


# Coloring

If a coloring graph is not 2-colorable, do not assign any color to its vertices. An illustration is as follows. There are no colors on Vertex 1, Vertex 2, Vertex 3, Vertex 4, and Vertex 5 that correspond to the five shapes.



# Calculating Color Density - Windows



# Calculating Color Density - Windows

Generate the color density windows to totally cover a coloring bounding box based on the following guidelines.

1. The first color density window is placed at the bottom-left corner of the coloring bounding box.
2. The second color density window is placed at the right hand side of the first color density window.
3. The next color density window is placed at the right hand side of the current color density window. If the new color density window overlaps the right edge of the coloring bounding box, the window is shifted left until it abuts the right edge of the coloring bounding box.
4. The next color density window is placed at the left-most position of the next row. The next row is on the top of the current row.
5. If the new color density window overlaps the top edge of the coloring bounding box, the window is shifted down until it abuts the top edge of the coloring bounding box.

# Calculating Color Density

Given a color density window, its color density is calculated as follows.

1. The area of color-A: Sum the area of the shapes having color-A in the color density window.
2. The area of color-B: Sum the area of the shapes having color-B in the color density window.
3. Calculate color-A density based on the color density definition.
4. Calculate color-B density based on the color density definition.

# Calculating Color Density

**The area of the color density window**

= 900×900 = 810000

**The area of color-A in the color density window**

= (940-720)(220-120)+(820-640)(470-400) = 34600

**The area of color-B in the color density window**

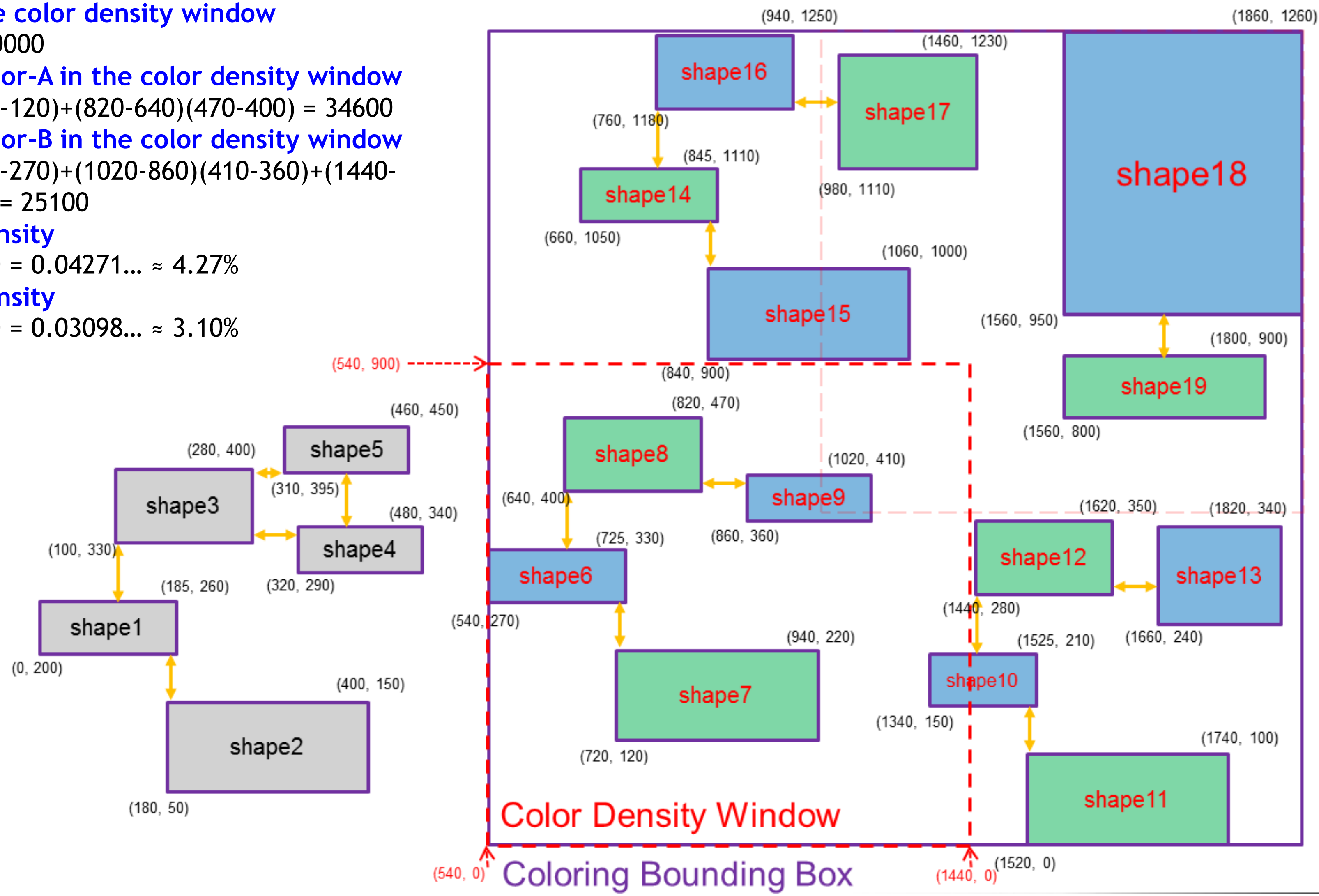
= (725-540)(330-270)+(1020-860)(410-360)+(1440-1340)(210-150) = 25100

**The color-A density**

= 34600÷810000 = 0.04271… ≈ 4.27%

**The color-B density**

= 25100÷810000 = 0.03098… ≈ 3.10%



# Input Format

The input file is an ASCII text file giving α, β, ω, and the coordinates of rectangles. There is a comma (,) between two consecutive coordinates. Below is an input file for a design with *m* rectangles.

ALPHA=α

BETA=β

OMEGA=ω

$x1_1, y1_1, x1_2, y1_2$

…

$xi_1, yi_1, xi_2, yi_2$

…

$xm_1, ym_1, xm_2, ym_2$

$xi_1$ and $yi_1$ are the X and Y coordinates of the bottom-left corner of the *i*-th rectangle.
$xi_2$ and $yi_2$ are the X and Y coordinates of the top-right corner of the *i*-th rectangle.

# Output Format

The execution of your program should create a file that consists of two parts.

The first part gives the coordinates of color density windows and the densities for color-A and color-B respectively.

The second part shows the colors being assigned to the shapes in each individual coloring graph. There is a comma (,) between two consecutive coordinates. Below is the format of an output file.

# Output Format

$WIN[d]=x\_bottom\_left_d, y\_bottom\_left_d, x\_top\_right_d, y\_top\_right_d(color\_A\_density_d\ color\_B\_density_d)$

...

GROUP

*NO[i]* indicates the *i*-th shapes of a non-2-colorable coloring graph for *i* = 1, 2, 3, … .

$NO[i]=x\_bottom\_left_i, y\_bottom\_left_i, x\_top\_right_i, y\_top\_right_i$

...

GROUP

*CA[a] shows the color assigned to the a-th shape for a = 1, 2, 3, … .*

$CA[a]=x\_bottom\_left_a, y\_bottom\_left_a, x\_top\_right_a, y\_top\_right_a$

...

$CB[b]=x\_bottom\_left_b, y\_bottom\_left_b, x\_top\_right_b, y\_top\_right_b$

...

GROUP

*CB[b] shows the color assigned to the b-th shape for b = 1, 2, 3, … .*

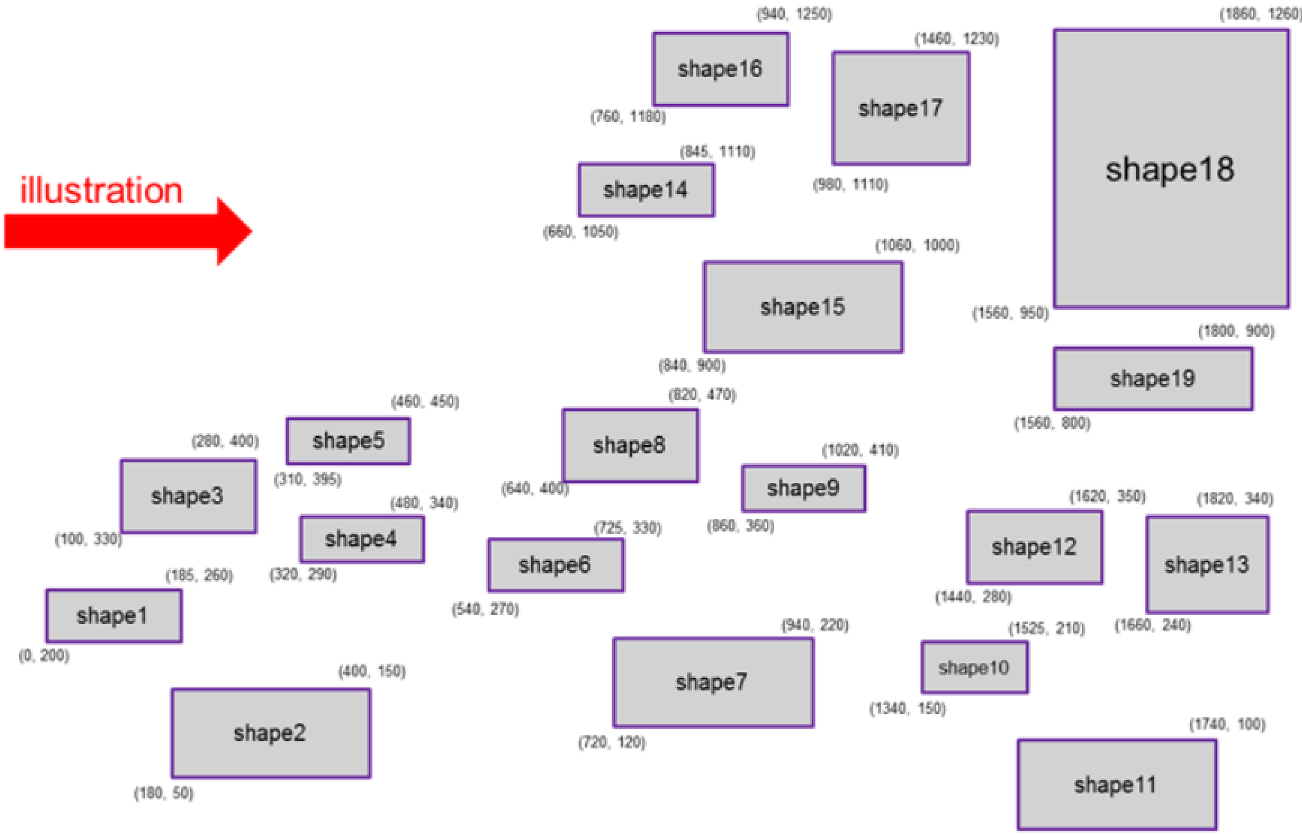
...

# Example - Input

### Input File

ALPHA=50
BETA=100
OMEGA=900
0,200,185,260
180,50,400,150
100,330,280,400
320,290,480,340
310,395,460,450
540,270,725,330
720,120,940,220
640,400,820,470
860,360,1020,410
1340,150,1525,210
1520,0,1740,100
1440,280,1620,350
1660,240,1820,340
660,1050,845,1110
840,900,1060,1000
760,1180,940,1250
980,1110,1460,1230
1560,950,1860,1260
1560,800,1800,900

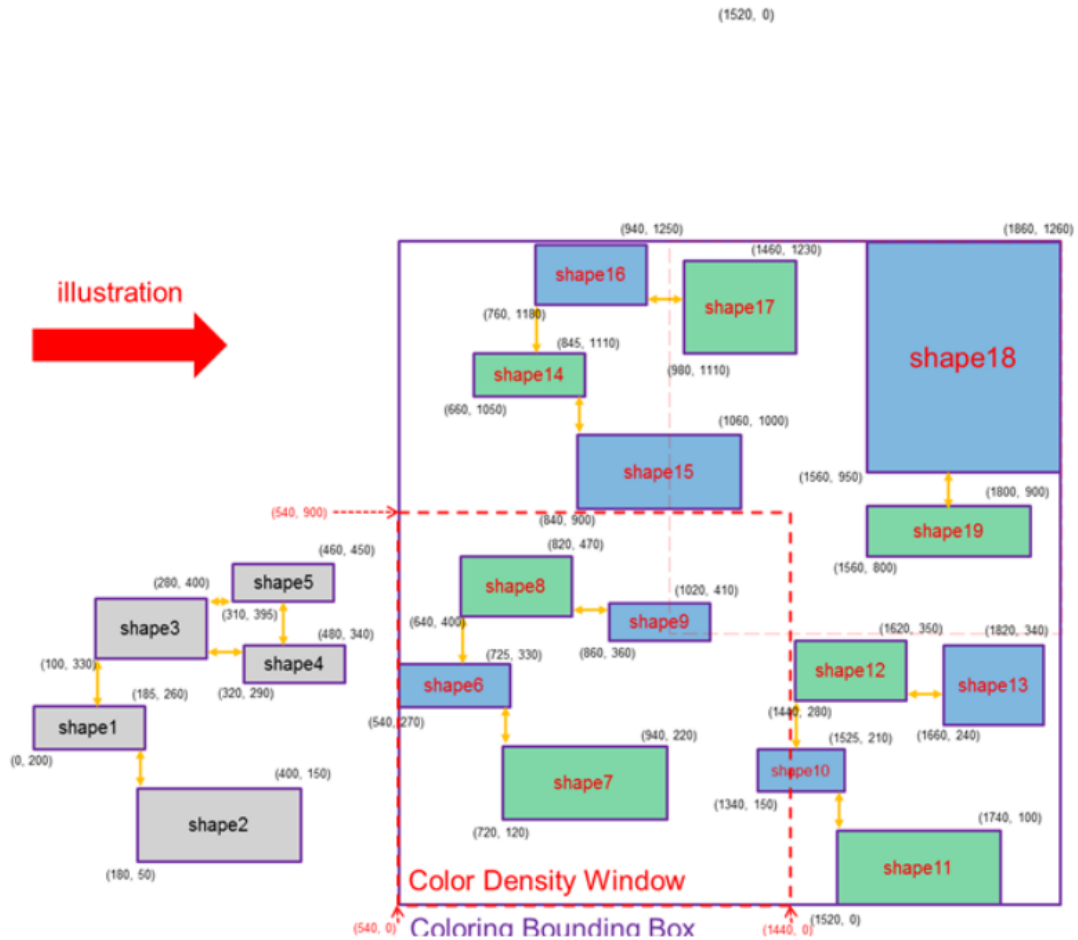
illustration



# Example - Output

## Output File

*WIN[1]=540,0,1440,900(4.27 3.10)*
*WIN[2]=960, 0,1860,900(7.23 3.72)*
*WIN[3]=540,360,1440,1260(9.51 5.26)*
*WIN[4]=960,360,1860,1260(10.07 13.09)*
*GROUP*
*NO[1]=0,200,185,260*
*NO[2]=180,50,400,150*
*NO[3]=100,330,280,400*
*NO[4]=320,290,480,340*
*NO[5]=310,395,460,450*
*GROUP*
*CA[1]=720,120,940,220*
*CA[2]=640,400,820,470*
*CB[1]=540,270,725,330*
*CB[2]=860,360,1020,410*
*GROUP*
*CA[1]=1520,0,1740,100*
*CA[2]=1440,280,1620,350*
*CB[1]=1340,150,1525,210*
*CB[2]=1660,240,1820,340*
*GROUP*
*CA[1]=660,1050,845,1110*
*CA[2]=980,1110,1460,1230*
*CB[1]=840,900,1060,1000*
*CB[2]=760,1180,940,1250*
*GROUP*
*CA[1]=1560,800,1800,900*
*CB[1]=1560,950,1860,1260*



# Programming Language

Use Tcl or Python or C or C++ to do programming.

The program file should be called as DPT_balance_color.
Please follow the following format to get the output file.

./DPT_balance_color ($input_file_name)
($input_file_name).out

./DPT_balance_color case1 case1.out

# Evaluation

The score is 100 for each test case. If your program encounters compilation errors, crash (coredump), runs more than 1 hour or needs more than 4G-byte memory for a test case, the score for this test case will be 0. The final score is the average of the scores of all the test cases.

# Thank you

Yongfu.li@sjtu.edu.cn