Lecture 9: Double Patterning

@luk036

2022-11-23

Background

- In the past, chips have continued to get smaller and smaller, and therefore consume less and less power.
- However, we are rapidly approaching the end of the road and optical lithography cannot take us to the next place we need to go.

Sub-wavelength Lithography

- Feature size « lithography wavelength
  - 45 nm vs. 193 nm

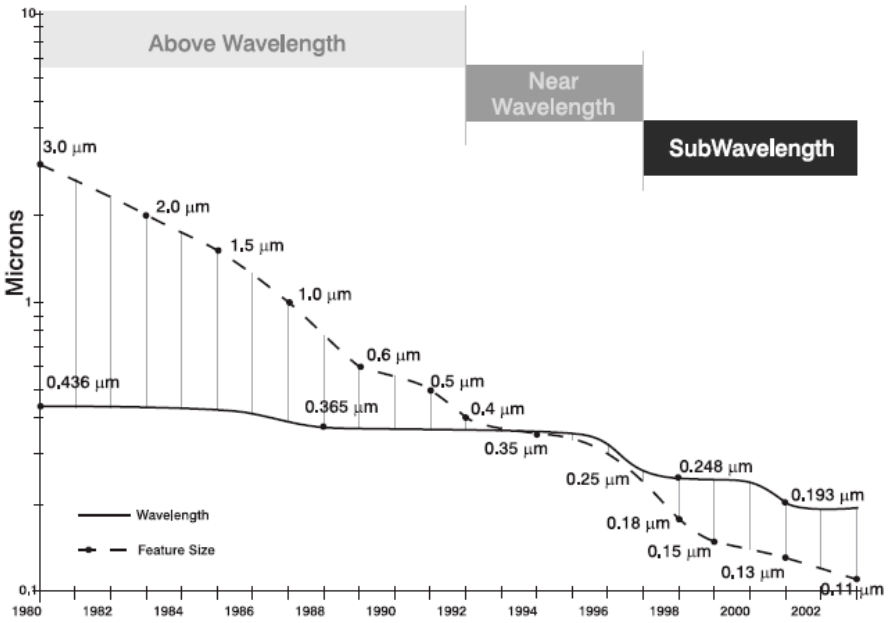


Figure 1: Shift to subwavelength optical lithography since the 0.35-micron process generation.

Figure 1: image

- What you see in the mask/layout is **not** what you get on the chip:
  - Features are distorted

– Yields are declined

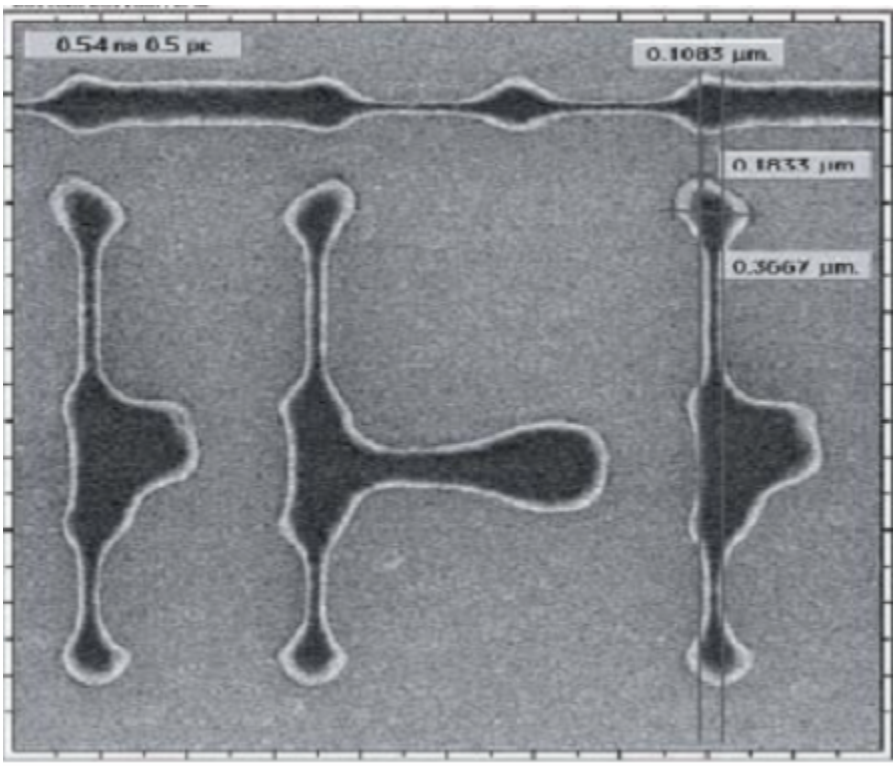


Figure 2: image

What is Double Patterning?

Unlike conventional optical lithography, which exposes the photoresist once under one mask, masks is exposed twice by splitting them into two, each with half its feature density.

Key technologies

- *Layout fracturing* algorithm to reduce the number of rectangles and the total cut length.
- *Dynamic priority search tree* for plane sweeping.
- Graph-theoretic approach:
  - Convert the coloring problem to a T-join problem and then solve it with Hadlock's algorithm.

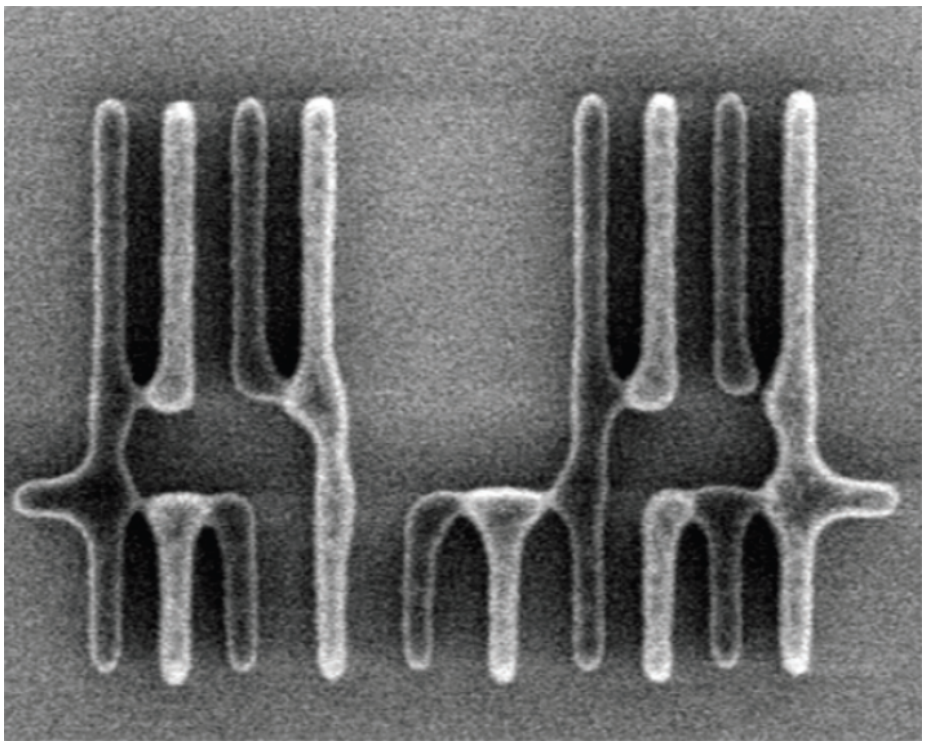


Figure 3: image

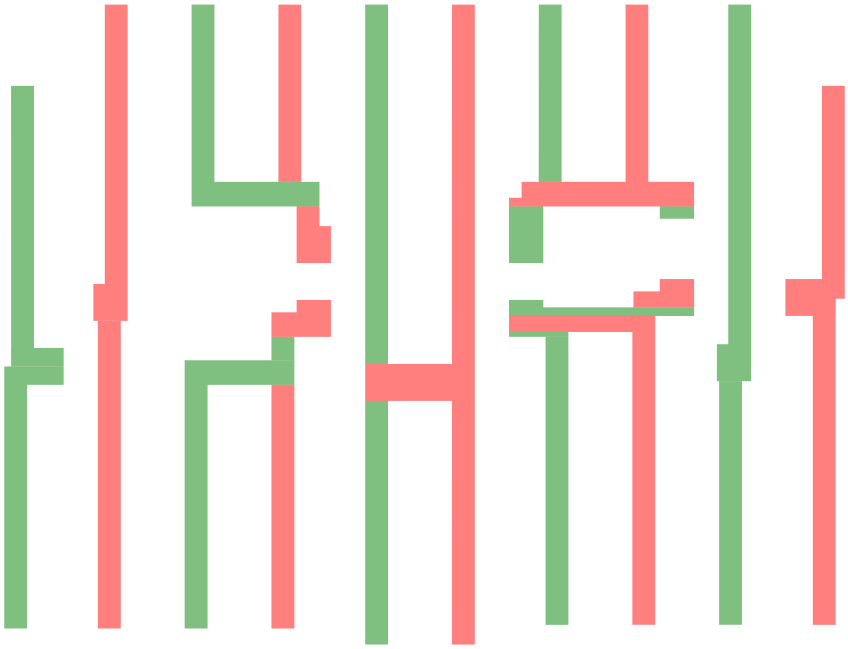


Figure 4: image

- Decompose the underlying conflict graph into its tri-connected components using a data structure named *SPQR-tree*.

Polygon Fracturing Algorithm

- Allow minimal overlap to reduce the number of rectangles, and thus the number of conflicts.

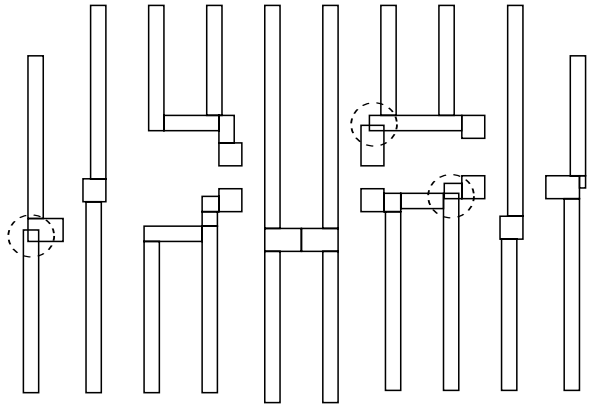


Figure 5: image

Conflict Detection

- Rule 1: If the distance between two rectangles is $\geq b$, then the two rectangles are *not* in conflict.
- Rule 2: Two overlapping/contacting rectangles are **NOT** conflict.

- Rule 3:
  - Definition: A polygon is said to be *rectilinearly convex* if it is both *x-monotone* and *y-monotone*.
  - Two rectangles X and Y are in conflict if they are $\leq b$ apart and there is a path from X to Y that reconstructs a "concave" polygon.
  - Conflicting: (A, C), (B, D), but not (A, B), (A, D) and (B, C).

Conflict Graph

- Blue edge: positive weight (opposite color preferred)
- Green edge: negative weight (same color preferred)

Formulation of the Layout Decomposition Problem

- INSTANCE: Graph $G = (V, E)$ and weight function $w : E \to Z$
- SOLUTION: Disjoint subsets of vertices V_0 and V_1 so that $V_0 \cup V_1 = V$ and $V_0 \cap V_1 = \emptyset$.
- MINIMIZE: total cost $\sum_{e \in E_c} w(e)$ where $E_c = \{(u, v) : u, v \in V_0$ or $u, v \in V_1, (u, v) \in E\}$

Note: the problem is - Linear time solvable for bipartite graphs. - Polynomial time solvable for planar graphs. - But in general, NP-hard (even for tripartite graphs)

Graph-Theoretic Approach

- Q: How can we produce a high-quality result when the problem is NP-hard?
- A: Observe that G is a nearly planar graph: we can use Hadlock's algorithm.
- However, the time complexity of this method is still very high.
- Solution: Graph division methods.

SPQR-Tree

Connected Graph

- Recall that a graph $G = (V, E)$ is a *connected* if every pair of vertices u, v in G is connected by a path.
- A graph can be divided into its connected components in linear time.
- Clearly, the color assignment problem can be solved independently for each connected component without affecting any QoR.

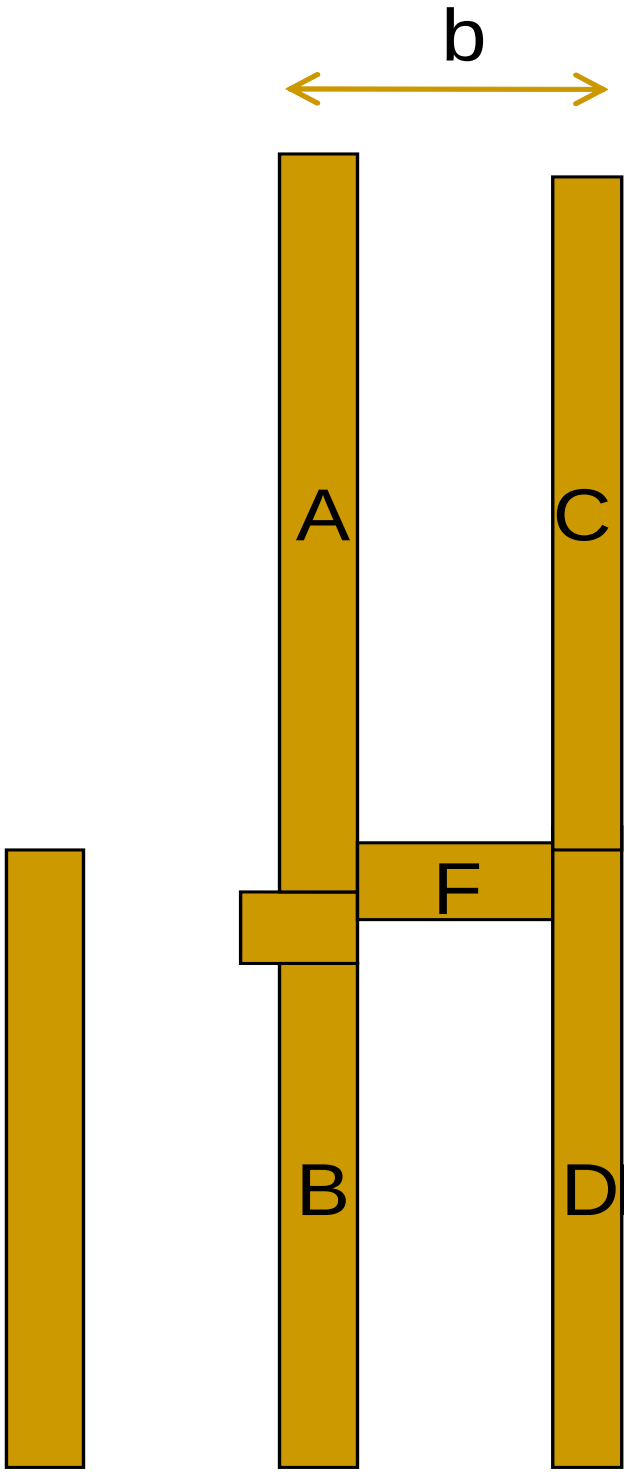


Figure 6: image

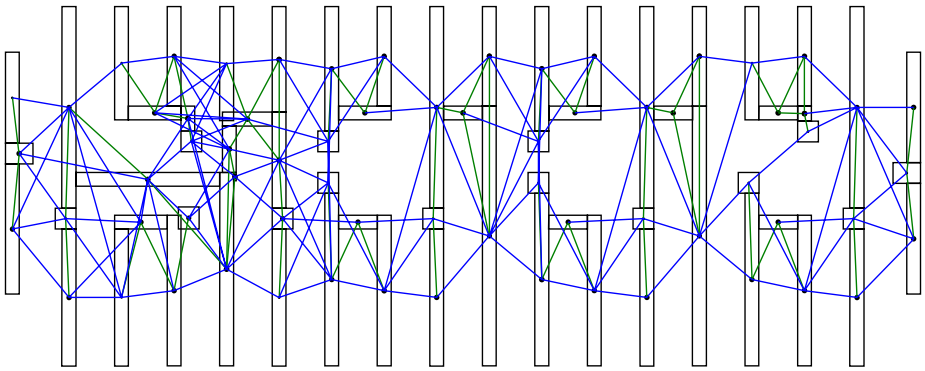


Figure 7: image

Bi-connected Graph

- A vertex is called a *cut-vertex* of a connected graph G if removing it disconnects G.
- If no cut-vertex is found in G, then the graph is called a bi-connected graph.
- In the following example, a, b and c are cut-vertices.

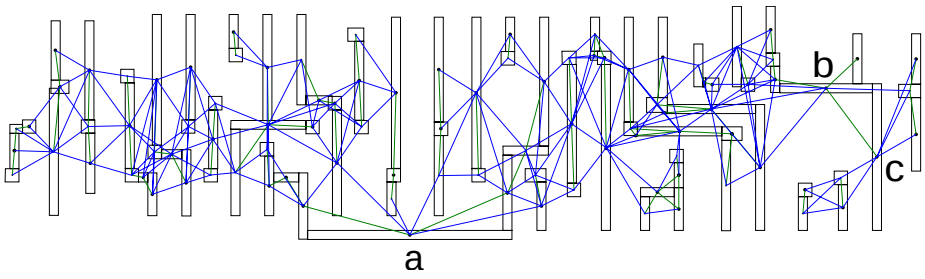


Figure 8: An example of a conflict graph with its bi-connected components. Vertices a, b, and c are cut-vertices.

Bi-connected Components G'

- A division of G into its bi-connected components can be performed in linear time by using a simple depth-first search to identify cut-vertices.
- It can be easily shown that the color assignment problem can be solved for each bi-connected component separately without affecting any QoR [@chiang_fast_2005]
- Question: Is it possible to further decompose the graph?

Tri-connected Graph

- If removing a pair of vertices will disconnect G', the pair is called a *separation pair* of G'.
- If no separation pair can be found in G', then it is called a *tri-connected graph*.
- In the following example, (a, b), (g, h), (c, d), (c, e) and (c, f) are separation pairs.

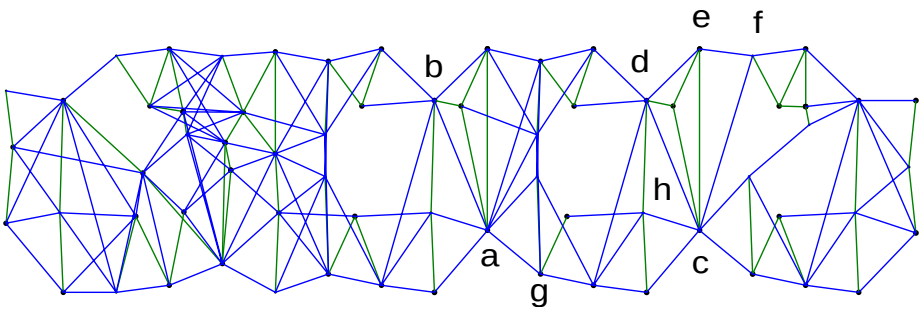


Figure 9: An example of a conflict graph and its tri-connected components. $\{a, b\}$, $\{c, d\}$, $\{c, e\}$, $\{c, f\}$ and $\{g, h\}$ are separation pairs.

Tri-connected Graph Division

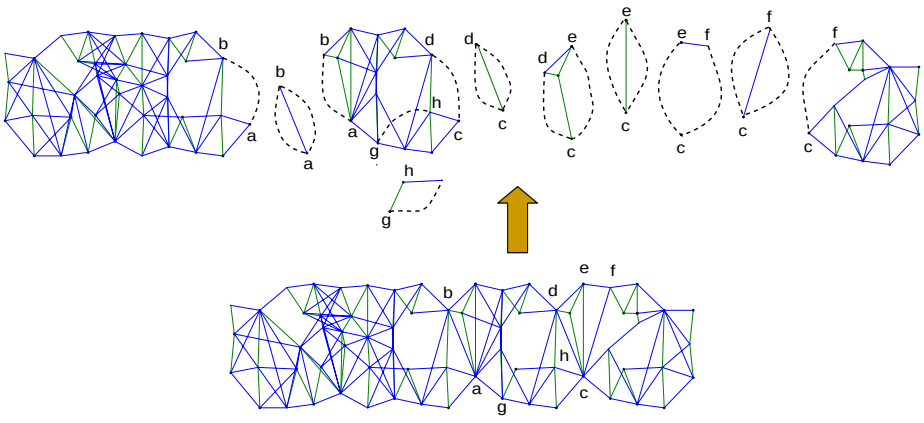


Figure 10: image

SPQR-tree

- A division of G' into its tri-connected components can be performed by identifying the separation pairs in linear time with the help of SPQR-

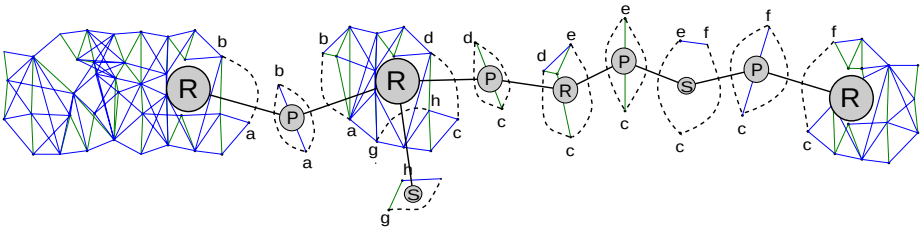


Figure 11: image

tree [@gutwenger_linear_2001].

Skeleton

- Each tree node of SPQR-tree is associated with a tri-connected component of G' called *skeleton*
- A skeleton represents a contraction of G' based on a set of *virtual edges*.
- A skeleton was classified into four types:
  - Series (S): the skeleton is a cycle graph.
  - Parallel (P): the skeleton contains only two vertices s and t, and k parallel edges between s and t where $k \geq 3$.
  - Trivial (Q): the skeleton contains only two vertices s and t, and two parallel edges between s and t, one of which is virtual and the other is real.
  - Rigid (R): the skeleton is a tri-connected graph of a type other than the above.

Divide-and-Conquer Method

Divide-and-conquer method

Consists of three basic steps:

1. Divide a conflict graph into its tri-connected components.
2. Conquer each tri-connected component in a bottom-up manner.
3. Combine the solutions into a complete solution in a top-down manner.

We calculate two possible solutions for each component, namely (s, t) of the same color and (s, t) of the opposite color.

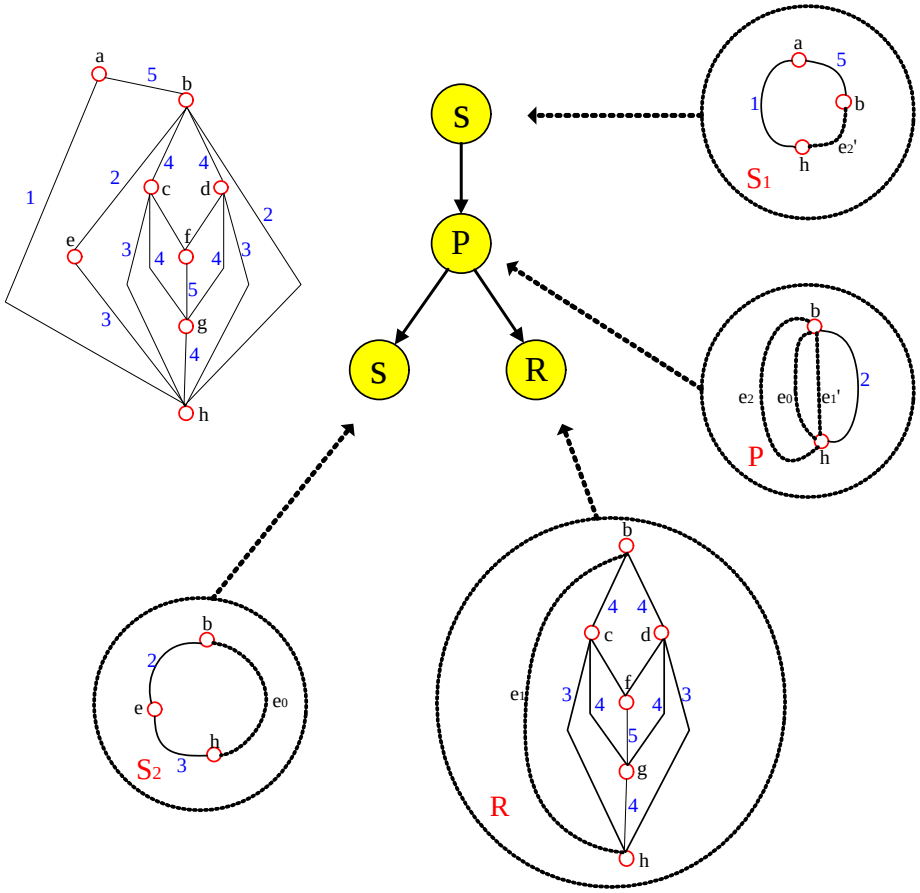


Figure 12: image

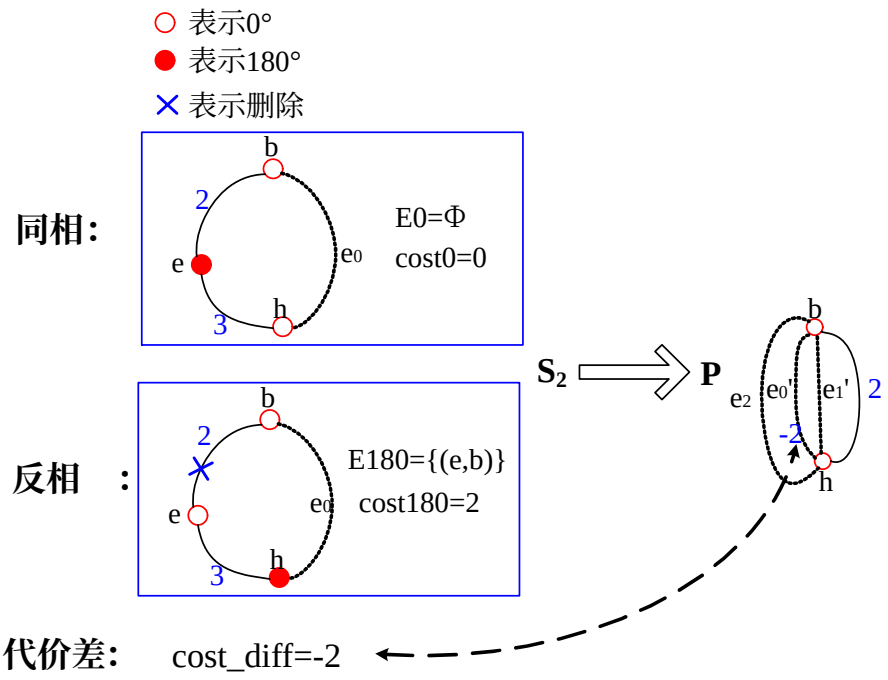


Figure 13: image

Bottom-up Conquering: S Type

P Type

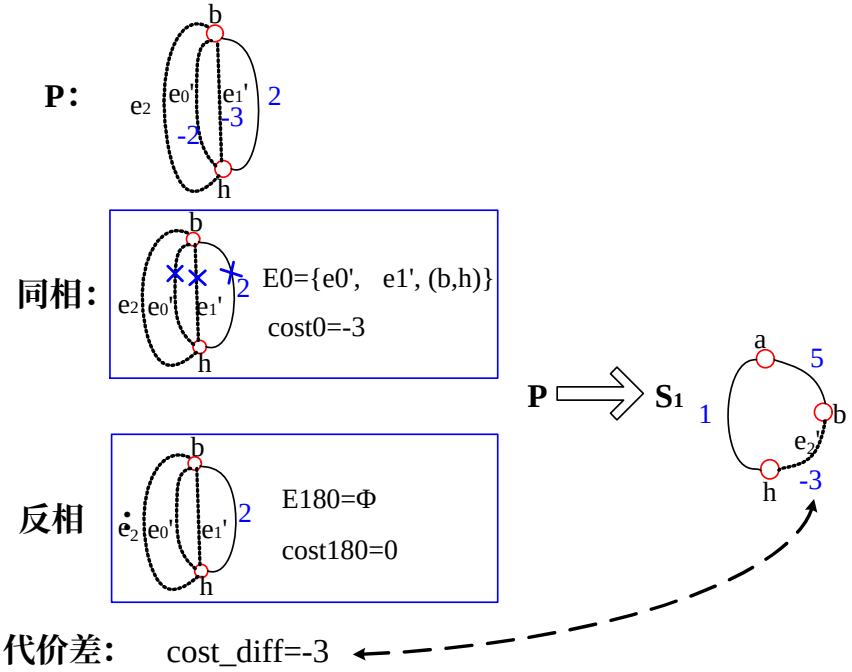


Figure 14: image

R Type

Top-down Merging

Node Splitting

- Node splitting (additional rectangle splitting) for resolving conflicts.
- To reduce the number of "cuts", we apply node splitting after one color assignment and then recolor.

.pull-left[

Before:

] .pull-right[

After:

]

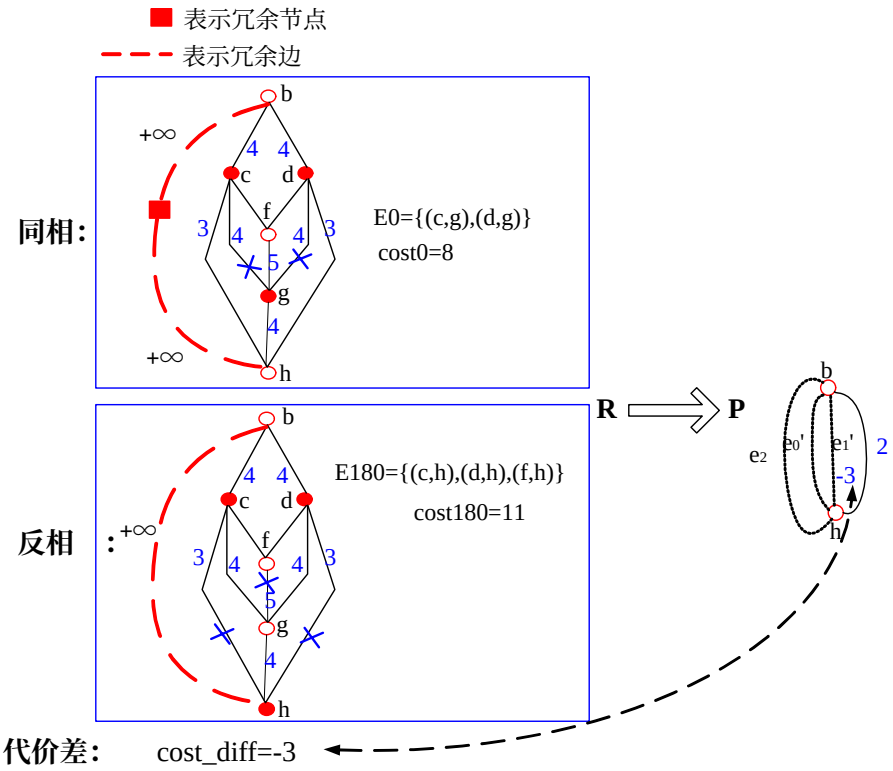
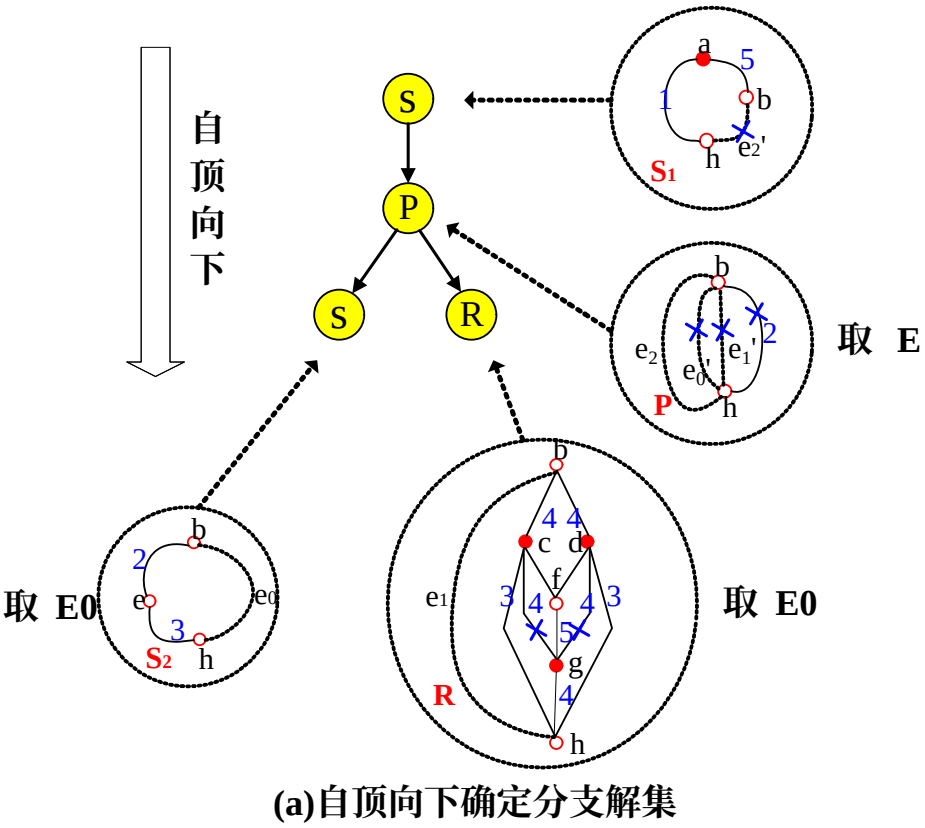


Figure 15: image



(a)自顶向下确定分支解集

Figure 16: image

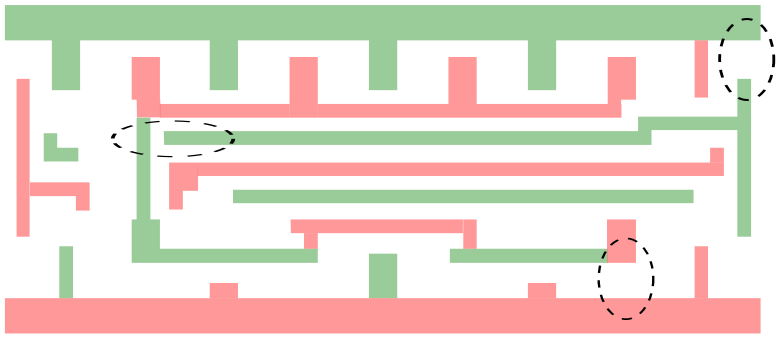


Figure 17: image

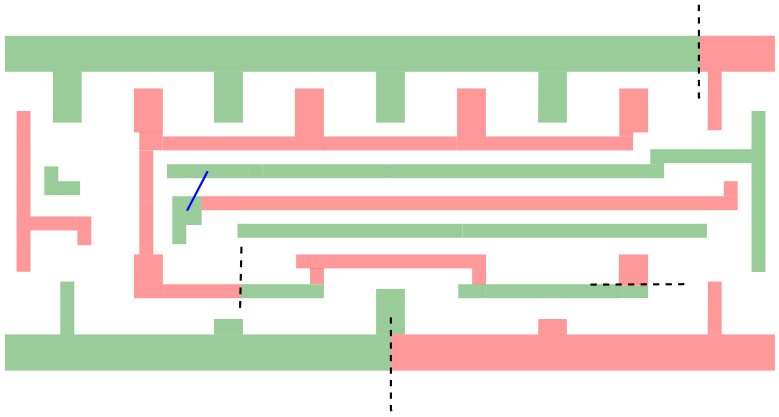


Figure 18: image

class: middle, center

Experimental Results

45 nm SDFFRS_X2 Layer 11, 9

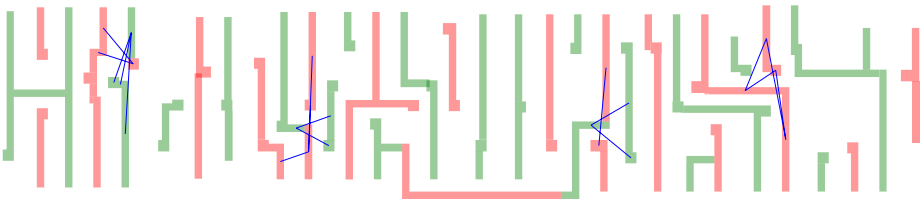


Figure 19: image

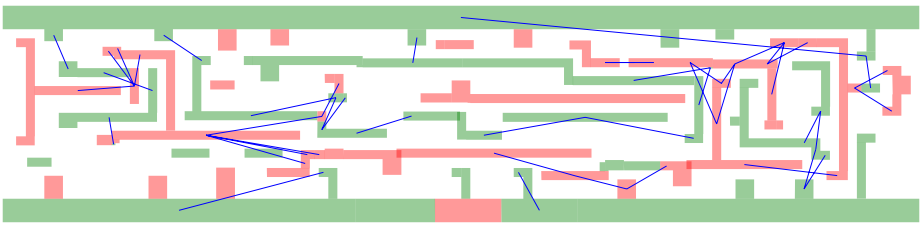
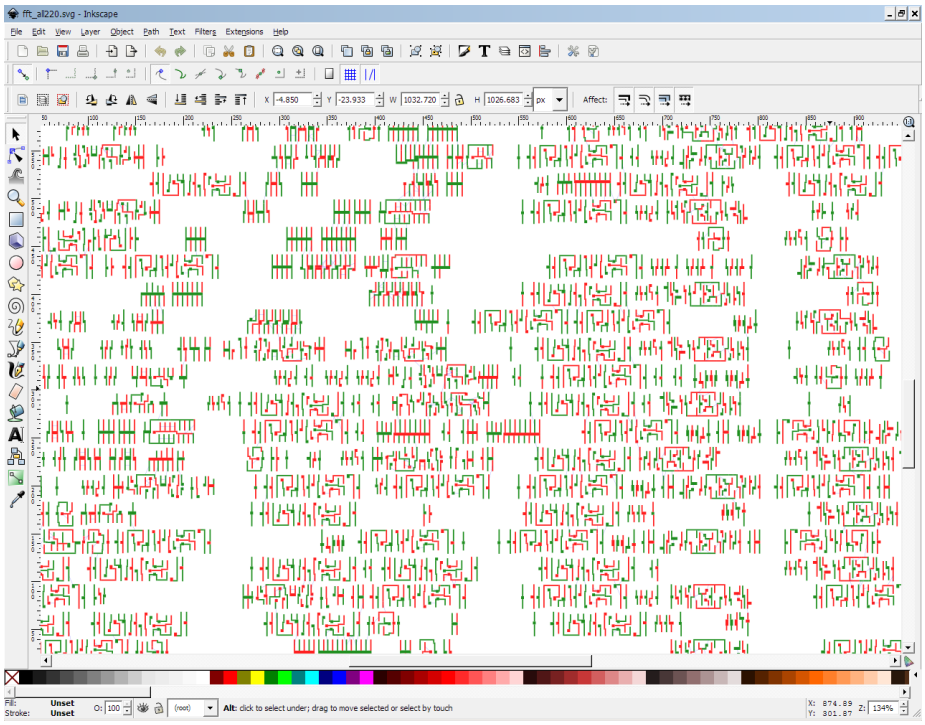


Figure 20: image



fft_all, 320K polygons

Experimental Results

Table 1: Experimental results of the runtime and cost reduction (with minimizing the number of stitches)

#poly	#nodes/#edges	w/ spqr	w/o spqr	time	cost
3631	31371/52060	13.29	38.25	65.3%	4.58%
9628	83733/138738	199.94	2706.12	92.6%	2.19%
18360	159691/265370	400.43	4635.14	91.4%	1.18%
31261	284957/477273	1914.54	9964.18	80.7%	1.61%
49833	438868/738759	3397.26	15300.9	77.8%	1.76%
75620	627423/1057794	3686.07	17643.9	79.1%	2.50%