VLSI-DFM_Final Project

---

(1) 洪偉勛 109065527

(2) How to compile and execute my program:

$ make

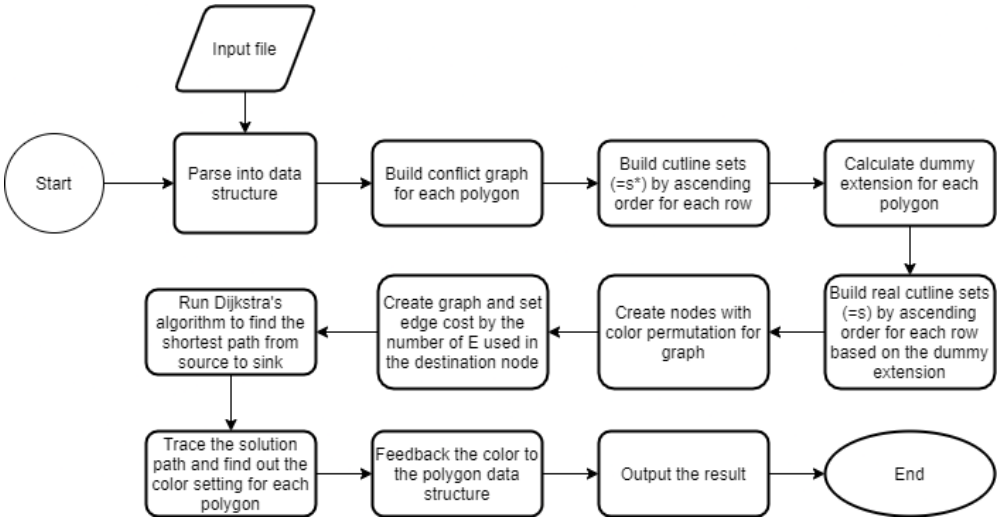
$ ../bin/main 0.0128 ../testcase/M0.out ../output/M0_00128.out

$ ../bin/main 0.0128 ../testcase/aes.out ../output/aes_00128.out

$ ../bin/main 0.064 ../testcase/M0.out ../output/M0_0064.out

$ ../bin/main 0.064 ../testcase/aes.out ../output/aes_0064.out

(3) Algorithm flow and explanations:



When detecting the conflict graph, each polygon has a vector to record the conflicted polygons. I extend the rects in each polygon and see if they are conflicting with any of the rects in other polygons. If 1 rect is found in conflict, the for-loop ends and the conflicted polygon is added to the conflict graph of the current polygon.

The cutline was built based on the min_x of each polygon. One cutline could have more than 1 polygon. The first cutline vector is s* mentioned in the class. After that the dummy extension is set for each polygon. A tolerance delta is given as an offset distance from the rightmost conflicting cutline. After dummy extension is finished, the real cutline set can be done by examine if any polygons and their dummy extension is passing the cutline.

```
#define delta 0.000001; //for dummy extension
```

```
if (flag) { this->dummy_x = rightmost_x - delta; }
else { this->dummy_x = this->max_x; }
```

The permutation process is done for at most 4 polygons by the fact that the maximum number of polygons in a cutline set is always less then 4 in all testcases. I list all the possible color combinations and create a node for each regardless of the conflict issue, which is dealt when creating edges. Besides, there are 4 colors, A, B, C and E.

When creating edges, one needs to examine if there are (1) in-node conflict and (2) inter-node conflict. If all checks pass, we can add this edge to the adjacent list by creating an adjNode with edge weight according to the numbers of 'E' used in the destination node.

For the starting node, edges are built to link all nodes in the first set in the cutline. For the sink node, the nodes in the last set in the cutline are linking to it with zero weight.

After all edges and nodes are ready. We can run the Dijkstra's algorithm and find out the path with minimum cost, which shall permit us the optimal solution for minimum usage of E-beam. We trace back this solution path and find out the color to use for each polygon.

There is a trick. In case of using very small dmin such as 0.0128, there is actually no conflict between any polygon pairs for two given testcases. So I simply assign A color to all polygons. The runtime is only the I/O time and the use of E-beam is zero.

The same treatment is done for those "isolated" polygons which is removed from the SG.

```
if (this->polygons[i]->CG.size() == 0) { //if poly not in any CG, assign it with color A and don't add to SG
    this->polygons[i]->set_to_A();
    this->polygons[i]->inSG = false;
    numAs++;
}
```

(4) Verification Results:

Environment: tested on NTHUCAD-server: ic55

dmin = 0.064

[M0.out] # of E-beam: 111

```
[wshung20@ic55 src]$ ./Verify 0.064 ../testcase/M0.out ../output/M0_out.out
Layout checking pass !
Mask checking pass !
Conflict checking pass !

[ Number of E-beam  ]   111
```

[aes.out] # of E-beam: 163

```
[wshung20@ic55 src]$ ./Verify 0.064 ../testcase/aes.out ../output/aes_out.out
Layout checking pass !
Mask checking pass !
Conflict checking pass !

[ Number of E-beam  ]   163
```

dmin = 0.0128

[M0.out] # of E-beam: 0

```
[wshung20@ic55 src]$ ./Verify 0.0128 ../testcase/M0.out ../output/M0_out.out
Layout checking pass !
Mask checking pass !
Conflict checking pass !

[ Number of E-beam  ]   0
```

[aes.out] # of E-beam: 0

```
[wshung20@ic55 src]$ ./Verify 0.0128 ../testcase/aes.out ../output/aes_out.out
Layout checking pass !
Mask checking pass !
Conflict checking pass !

[ Number of E-beam  ]   0
```