HW4 Global Placement

資工 19 104062302 洪旻爹

Algorithm

The algorithm used in this homework, instead of “Simulated Annealing”, is “Analytical Approach”. By listing some objective functions, we can use conjugate method to solve these unconstrained optimization problems. In this case, the objective is to minimize wirelength while being able to be legalize after the global placement. Therefore, we need a differentiable function for wirelength and a smooth constraint for density. Instead of using MAX function which isn’t differentiable, I use Log-Sum-Exponential (LSE):

$$\eta \sum_{e \in E} (\log \sum_{v_k \in e} \exp(x_k/\eta) + \log \sum_{v_k \in e} \exp(-x_k/\eta) + \log \sum_{v_k \in e} \exp(y_k/\eta) + \log \sum_{v_k \in e} \exp(-y_k/\eta))$$

And for density function, I use Bell-Shaped Function with extension for large modules introduced in class:

- Let $d_x = |x_i - x_b|$

$$\tilde{\Theta}_x(b,i) = \begin{cases} 1 - 2 \times d_x^2/\omega_b^2 & \text{if } 0 \le d_x \le \omega_b/2 \\ 2 \times (d_x - \omega_b)^2/\omega_b^2 & \text{if } \omega_b/2 \le d_x \le \omega_b \\ 0 & \text{if } \omega_b \le d_x \end{cases}$$

$$D_b(x, y) = \sum_{i \in V} C_i \times \tilde{\Theta}_x(b,i) \times \tilde{\Theta}_y(b,i)$$

- Extension for large modules:

$$\tilde{\Theta}_x(b,i) = \begin{cases} 1 - a \times d_x^2 & \text{if } 0 \le d_x \le w_b/2 + w_i/2 \\ b \times (d_x - w_b - w_i/2)^2 & \text{if } w_b/2 + w_i/2 \le d_x \le w_b + w_i/2 \\ 0 & \text{if } w_b + w_i/2 \le d_x \end{cases}$$

where $a = 4/((w_b + w_i)(2w_b + w_i))$

$b = 4/(w_b(2w_b + w_i))$

Unit 5 61

p.s. The equations of a and b in Bell-Shaped Function are slightly different in the ppt of references. I follow the one introduced in class.

Implementation

Most of the program is already coded including file I/O, conjugate solver, legalization and detail placement except for the objective functions and their gradients. My goal is to calculate the objective value and the gradients for each x and y coordinate of each module.

At first, calculating partial derivative of each x and y coordinate is really hard because there are a lot of chain rules to apply. I had to discuss with my classmate to make sure every equation is correctly differentiated. If I gave the solver the wrong objective value or the wrong gradients, it may not be able to solve the equation and thus pop out "!" all the time or it may descend in the wrong direction and not able to minimize the objective. Fortunately, I was able to implement the objective value and the gradients correctly at the end.

The density term in objective function introduced in class not only penalize the overflow bins but also the underflow bins (i.e. It tries to make sure every bin is as close to the target bin density as possible.) But in other paper, I've seen the objective function without penalizing the underflow bins. Therefore, I've implemented it but the result didn't get better. Instead of discarding this change with the unimproved wirelength, I keep it because it seems more reasonable not to penalize the underflow bins.

Base on the ideas from some papers, I run the solver for several times with different beta of the density term. The idea is that I should try to minimize the wirelength as small as possible in the first round by setting beta to 0 and try to minimize the density term at the later rounds by setting beta larger along with the rounds. At first, I let solver run for 4 round with increasing beta, 0, 2500, 5000, 7500. These numbers are some magical numbers base on experience to make the density term large enough. The wirelength after global placement is really small and is able to get smaller if the solver runs for more rounds. But no matter how small the wirelength is or how many rounds the solver runs for, the wirelength after detail placement is always around the same number. Therefore, I let the solver only run for 2 rounds while minimizing the wirelngth in the first round and minimizing the density term in the second round. Although the wirelength after global placement isn't as small as the one running for many rounds and spending less effort on minimizing the density term, the result is still good enough and able to be legalized.

Results

	Runtime (s)	Wirelength
ibm01	21	218102823
ibm05	76	22427520

Compared to the top 5 in last year, mine is a lot worse on wirelength but competitive on runtime. There is an interesting thing I've found in implementation. Although a mistake in calculating objective value or gradients would lead to failure while solving, some particular mistake combinations, instead of causing failure while solving, may come with a better solution with smaller wirelength. Even though the wrong derivative calculation could have better solution, I insist coding it with the right form.

Compile & Execute

Compilation

```
$ make
```

Remove executable file

```
$ make clean
```

Execution

```
$ ./place –aux <path/to/aux/file>
```

e.g.

```
$ ./place –aux benchmark/ibm01/ibm01-cu85.aux
```