Assignment 2 Report

Dingqing Yang 38800141

2020 Feburary

1 Discussion on Aspects of the Marking Criteria

- *Basic Simulated Annealing Algorithm*: The simulated annealing algorithm is implemented in `simulatedAnnealer` method in `placer` class. I implement the `proposeTwoPoint` method that randomly pick 2 points to swap. A delta cost is evaluated in `evalDeltaCost` method. The delta cost is used to determine how likely we will actually perform the swap. The temporature will scaled down every `cooling_period` iteration. The search will terminate either:
  - we cannot improve for a consecutive `early_stop_iter`, or
  - we hit the maximum number or iteration of a search
- *Cost Function Calculation*: I implement a function called `evalCost` in `utils.py` for this part. It takes the coordinates of n blocks in a net as an n by 2 numpy array. The min and max across rows give the 2 coordinates that specifies the bounding box, the half perimeter bounding box is evaluate as L1 distance between them.
- *Efficiency*: I am trying to improve efficiency in many parts. First, during `proposeTwoPoint` that randomly pick 2 points to swap, I directly skip the case that 2 proposed coordinates cooresponding to 2 empty cells. Therefore the 2 proposed coordinates contains at least a block. Second, since delta cost is only determined by blocks that gets swapped, I only need to evaluate delta cost for all nets that contains the swapped block. Mapping from block to all nets is saved after initial input parsing to speedup the process of accessing all related nets. Finally, if 2 swapped blocks are in the same net for some net, the delta evalution is also skipped to save execution time.
- *Resuls*: The annealing schedule tuning is implemented in `hyperparameter_search.py`. In this script, I implement the grid search for tuning different initial temperature, cooling period, and cooling coefficient that controls the speed of temperature reduction. The average cost across all provided benchmarks is 3420.3.

State	**Init T**	**Cooling Period**	**Beta**	**Final Cost**
C880	10	1000	0.95	1026
alu2	1	1000	0.9	959
apex1	10	1000	0.9	6550
apex4	1	1000	0.9	13046
cm138a	1	1000	0.8	35
cm150a	1	1	0.95	64
cm151a	1	1	0.95	34
cm162a	1	10	0.8	80
cps	1	1000	0.95	6067
e64	10	1000	0.95	2083
paira	100	1000	0.9	5393
pairb	10	1000	0.95	5707
Average Cost	-	-	-	3420.3

Table 1: Obtained Final Cost on Each Benchmarks with the Corresponding Scheduling

- *Code Quality*: I implement a `Placer` class that contains methods that work for GUI and the routing algorithm. Each method in the class is documented.
- *Initiatives*: The hyperparameter tuning stage is seperated into 2 phases:
  - Initial fast exploration
  - Fine-tune best found result

  In the first phase, the termination condition (i.e. number of consecutive non-improving iterations) is set low to quickly explore the hyperparameter space. The observation is that ranking among different instances in the hyperparameter space can be determined much early to avoid full annealing. In the second stage, the termination condition is set much higher to fine tune the found optimal scheduling for each benchmark.

2 Table of Final Cost

Table 1 shows final cost of all benchmarks with its corresponding schedule.

3 How to run it

Examples:

- To launch GUI and show final result for *stdcell.infile*, run `python routing.py --infile benchmarks/st`
- Test all benchmarks without launching GUI, run `python test_all.py`

See README for more.