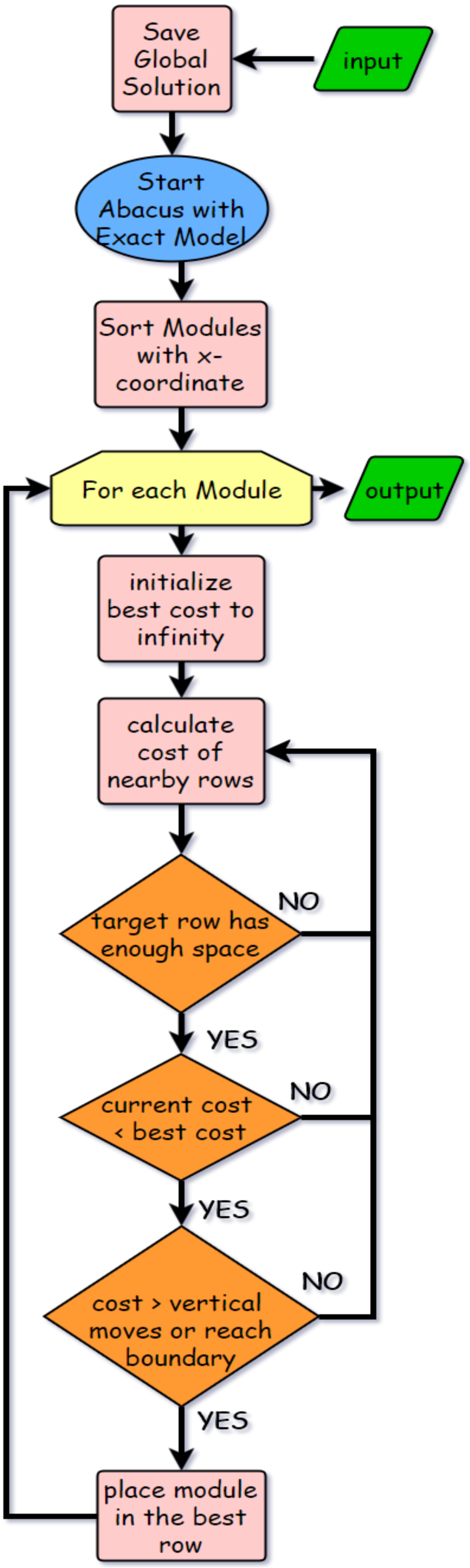
Physical Design for Nanometers ICs, PA 3

B03901056，電機三，孫凡耕

(1) Algorithm flow:


input
Save Global Solution
Start Abacus with Exact Model
Sort Modules with x-coordinate
For each Module
output
initialize best cost to infinity
calculate cost of nearby rows
target row has enough space
NO
YES
current cost < best cost
NO
YES
cost > vertical moves or reach boundary
NO
YES
place module in the best row

(2) Detailed description:

Forward and backward→

Different results will be obtained when the abacus algorithm is run in different directions. In my implementations, I found that forward pass (left to right) will perform better in smaller cases, while backward pass (right to left) will excel at larger cases. Both of which improves the solution by 1~3% compared with only a single pass.

Sorting→

Besides forward and backward pass, I found that different sorting scheme will also result in different solution quality. For forward pass, sort the modules according to center x-coordinate results in better solution, but as for backward pass, sorting according to left x-coordinate is preferred. This improves the solution by 1~3%.

Exact Model→

Originally, the Abacus was based on the approximated quadratic wire-length model in order to perform cost calculations and updates. However, the actual cost should be the absolute different in x and y-coordinate. So I implemented the exact model according to the paper [1]. The paper reformulate the cost function into an process of finding the median of a cluster. While estimating the cost of placeRow(trial), the paper summarize all conditions into three scenario, and the overall complexity is O(3). As for placeRow(final), the complexity is log(n) where n is the number of cells in the last cluster of that row. This not only speed up my 2 times but also improves the solution quality by 7%.

Cost function regarding previous placed cell→

Abacus only calculated the displacement of the current as the cost. According to the paper [1], considering the displacement of other cells in the same row made by placing the cell in that row will improve solution quality. Thus the cost function now becomes:

$$\alpha f1(c) + \beta f2(c)$$

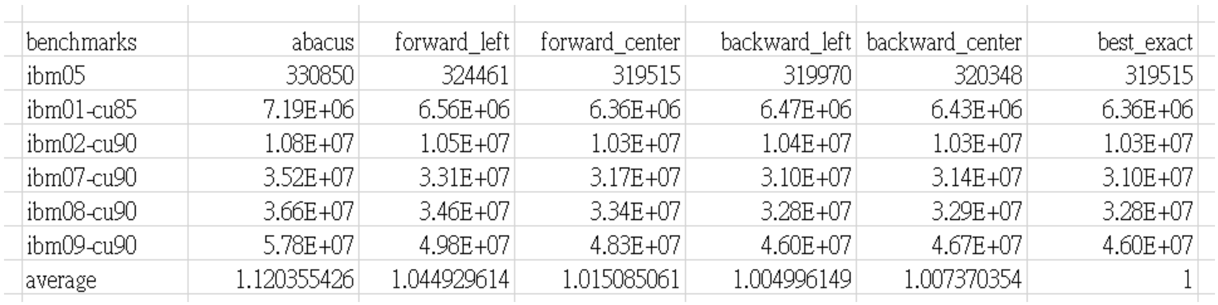
where α, β are use defined parameters.

(3) Data Structure:

```
struct Cluster2 {
  double _wc;
  multiset<double>::iterator _it;
  multiset<double> _xcs;
  vector<int> _modules;
};
```

In every row, there is a vector of Cluster2. In every Cluster2, _wc stores the current total length of all modules, _modules stores all ids of modules, _xcs stores all value of (x - _wc) for every modules and _it points to the median of _xcs.

(4) Problem and Discussion:



benchmarks	abacus	forward_left	forward_center	backward_left	backward_center	best_exact
ibm05	330850	324461	319515	319970	320348	319515
ibm01-cu85	7.19E+06	6.56E+06	6.36E+06	6.47E+06	6.43E+06	6.36E+06
ibm02-cu90	1.08E+07	1.05E+07	1.03E+07	1.04E+07	1.03E+07	1.03E+07
ibm07-cu90	3.52E+07	3.31E+07	3.17E+07	3.10E+07	3.14E+07	3.10E+07
ibm08-cu90	3.66E+07	3.46E+07	3.34E+07	3.28E+07	3.29E+07	3.28E+07
ibm09-cu90	5.78E+07	4.98E+07	4.83E+07	4.60E+07	4.67E+07	4.60E+07
average	1.120355426	1.044929614	1.015085061	1.004996149	1.007370354	1

P.s.:

Abacus: only forward pass and sort according to center x-coordinate

Forward: only forward pass with exact model

Backward: only backward pass with exact model

Left: sort according to left x-coordinate with exact model

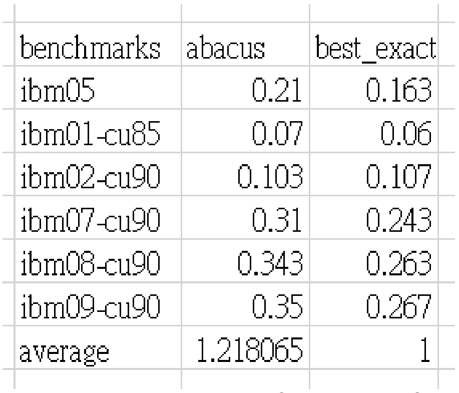
Center: sort according to center x-coordinate with exact model

Best_exact: max(forward_center, backward_left)

The actual result is different from above, since for comparison, the sorting are stable for stats in this table, but generally, unstable sorting results in better solution quality.

Improvement over solution quality→

As shown above, we can see that exact model performs better that Abacus for about 7~8%, comparing the Abacus and forward_center. Also, we can observe that sorting with center x-coordinate performs better for forward pass, but sorting with left x-coordinate performs better for backward pass. However, the exact reason is unknown. Another interest result is that forward_center is good at smaller cases while backward_left excels at larger cases. Combining the advantage of both forward_center and backward_left, the best_exact model can be obtained.



benchmarks	abacus	best_exact
ibm05	0.21	0.163
ibm01-cu85	0.07	0.06
ibm02-cu90	0.103	0.107
ibm07-cu90	0.31	0.243
ibm08-cu90	0.343	0.263
ibm09-cu90	0.35	0.267
average	1.218065	1

Improvement over solution quality→

From the above table, we can see that the best_exact model is faster that Abacus for about 20%. However, note that the Abacus only executes one pass, but best_exact do two passes. Thus, if ignoring other overhead, it is confident to say that exact_mode is more than two times faster than the original Abacus.