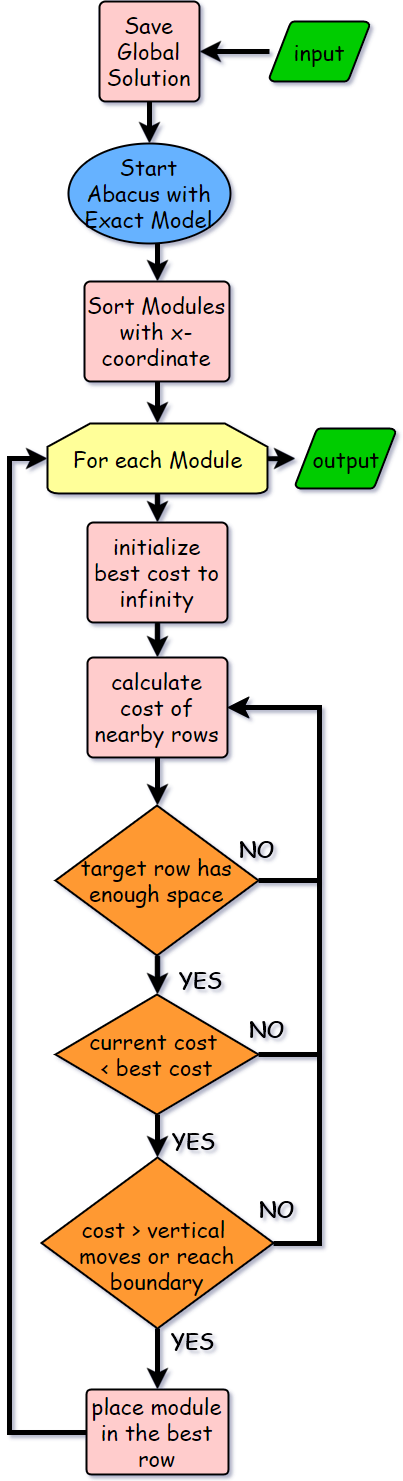
Physical Design for Nanometers ICs, PA 3

­B03901056, 電機三, 孫凡耕

1. Algorithm flow:

1. 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%.

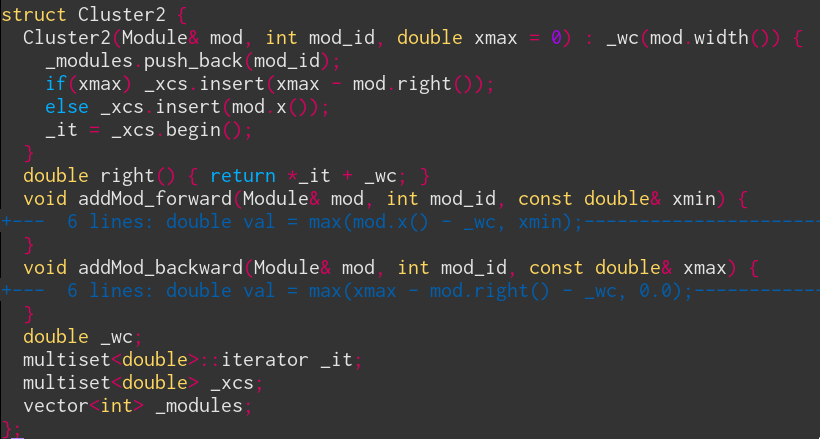
**Exact Model→**

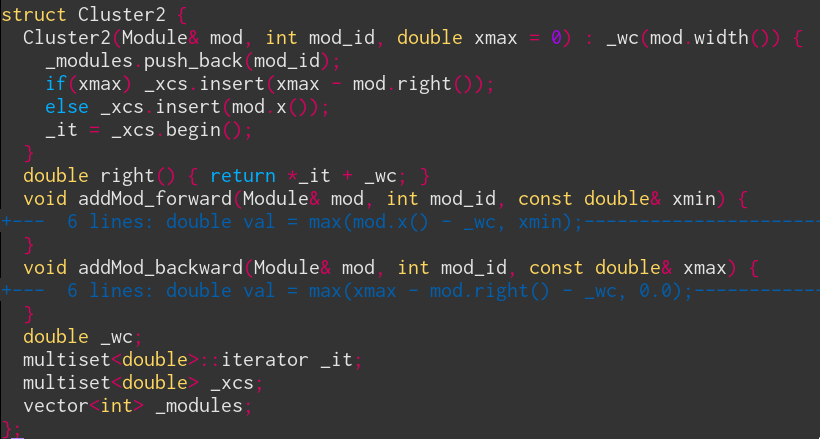
Originally, the Abacus was based on the approximated quadratic wire-length model in order to perform cost calculations and updates. However, the actual cost should be the absolute different in x and y-coordinate. So I implemented the exact model according to the paper [1]. The paper reformulate the cost 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:

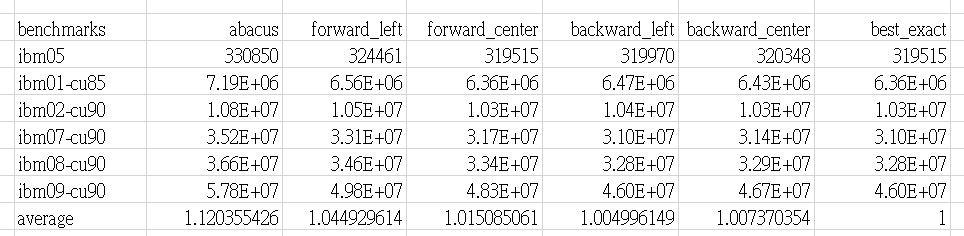
where α,β are use defined parameters.

1. Data Structure:



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.

1. Problem and Discussion:



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

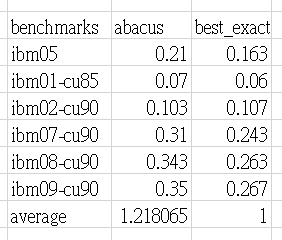
Center: sort according to center x-coordinate with exact model

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.



**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.