

Algorithms

Pseudo code

```

/* Initiation */ // O(N) – N: #Blocks
For each Blocki in Block array
    Block0 ← root
    Blocki ← Left child of Block(i-1)/2
    Blocki ← Right child of Block(i/2)-1

/* Calculate norm factor Areanorm & Wirelenthnorm */ // O(P) for each iteration – B* tree
Perturb Blocki randomly for T times ( T is a self-defined number )
Pack Blocks // O(N) – B* tree
Record Areai & Wirelenthi in each iteration // O(1) – contour line / O(P) – P: #pins
Areanorm ←  $\sum_i^T \text{Area}_i / T$ 
Wirelenthnorm ←  $\sum_i^T \text{Wirelenth}_i / T$ 

/* Calculate UphillCostavg */ // O(P) for each iteration
Perturb Blocki randomly till UphillCosti happens for U times ( U is a self-defined number )
Pack Blocks // O(N)
Evaluate Costi // O(1) / O(P)
UphillCostavg ←  $\sum_i^U \text{UphillCost}_i / U$ 

/* Fast Simulated Annealing */ // O(P) for each iteration
Stage 1: High-temperature stage ( k = 1 )
Tempinit ← UphillCostavg / ln P ( here acceptance probability P is set to 0.85 )
Perturb Blocki randomly for T times (same as stage 3) // O(P) for each iteration

Stage 2: Greedy local search ( k = 2 to 7 )
For k = 2 to k = 7
    Tempstage2 ← Tempinit × (  $\sum_i^T (\text{Cost}_i - \text{Cost}_{i-1}) / T$  )k-1 / k / c ( here c is set to 100 )
    Perturb Blocki randomly for T times (same as stage 3) // O(P) for each iteration

Stage 3: Hill climbing stage ( k > 7 )
While Tempstage3 > Tempstop
    Tempstage3 ← Tempinit × (  $\sum_i^T (\text{Cost}_i - \text{Cost}_{i-1}) / T$  )k-1 / k
    Perturb Blocki randomly for T times // O(P) for each iteration
    Pack Blocks // O(N)
    Evaluate Costi // O(1) / O(P)

    Set acceptance probability  $P = \min(1, e^{-\frac{\text{Cost}_i - \text{Cost}_{i-1}}{\text{Temp}_{\text{stage3}}}})$ 

    Modify weight  $\alpha = \alpha_{\text{base}} + (1 - \alpha_{\text{base}}) \times \frac{n_{\text{feasible}}}{n}$  (  $\alpha_{\text{base}}$  is set to 0.5; n set to 500 )

Return the best solution

```

Data structure

B*-tree

B*-tree supports floorplan abstract convert in **linear time**. Conveying with level ordering tree expression, the root is set to $a[0]$, and the adjacency relation between blocks is inferred as parent-and-child. That is, for Block_i stored in $a[i]$, its left child would be stored in $a[i*2+1]$, and its right child would be stored in $a[(i+1)*2]$. In this program, I mapped index to blocks through unordered map instead of using array to save memory space usage to **$O(N)$** , also, with one-to-one hash, searching time could still keep as **$O(1)$** . For the Block structure itself, I double linked parents and children to reduce time complexity of searching relatives, deleting and inserting Blocks to **$O(1)$** . Thus, the time complexity to rebuild a floorplan through its abstract convey is **$O(N)$** .

Contour line

Another important feature in this program would be contour line. It supports the calculation and update of area to be done in constant time. Contour line is stored by double linked list of nodes storing outline coordinates (x_i, y_i) . For updating, first, we iterate through the list to the closest node $x_i \leq x_{\text{target_start}}$ which takes **$O(N)$** . However, **N here is relatively small**, and thus we could say that the searching takes only **constant time**. Then we remove the latter nodes $x_{i+1} x_{i+2} \dots x_{i+j} \leq x_{\text{target_end}}$. The deleting takes **$O(1)$** with double linked list structure. Last, Insert $x_{\text{target_start}} \leftrightarrow x_{\text{target_end}}$ into the list, which takes **$O(1)$** to complete. Thus, it takes **constant time** for updating contour line. As for area calculation, what we will have to do is to iterate through the whole linked list and record the largest x_{Max} and y_{Max} . Then, we could calculate the area of the floorplan. The iteration takes **$O(N)$** with a **relatively small N** . We could say that it takes only **constant time** to get x_{Max} and y_{Max} for area calculation.

Time complexity

Perturbation

Either three of the operations, rotate, delete and insert, and swap, is implemented in one step. Rotate is to search for a specific Block and then rotate it. It takes **$O(1)$** to find the specific Block with unordered map, since it is a hash table maps one index to one Block. For “delete Block_i and insert under Block_j ”, if the chosen to be deleted Block_i is the root, or Block_i inserting will lead to renewing the whole tree index, it takes **$O(N)$** in worst case. However, if the chosen nodes are leaves, the time complexity is only **$O(1)$** . As for swap, finding two specific nodes takes **$O(1)$** . Then exchange their index and relatives also takes **$O(1)$** with double linked tree structure.

Packing

As mentioned in Data structure section, abstract convert and contour line update take **$O(N)$** to be accomplished. Therefore, Perturbation and Packing together takes **$O(N)$** .

Cost evaluation

Area evaluation takes only **$O(1)$** as mentioned above, but **wire length evaluation** takes **$O(P)$** . We evaluate total wire length with HPWL strategy, which is an efficient way to get the estimated wire length. However, iterate through the pins in each netlist is still necessary.

Fast simulated annealing

Total time the algorithm takes, is mainly **based on the progressed iterations**, here **k** . The main difference between conventional version and fast version is the dynamically adjusted temperature and cost function. Conventional SA has fixed temperature adjustment in each iteration and cost evaluation function, while fast

SA gave high temperature only in the beginning. In conventional SA, accepting probability always updates with $\text{Temp}_{\text{init}}$. On the other hand, in fast SA, when $k=1$, acceptance probability P is set high for accepting any kinds of possibility. When $k=2$ to 7 , temperature is set to low to accept only a small number of inferior solutions. When $k>7$, it becomes conventional SA with adaptive cost function. Adaptive cost function allows cost function to be modified while accepting portion changes. The origin cost is evaluated as $\text{cost} =$

$$\beta \times \frac{\text{Area}_{\text{cost}}}{\text{Area}_{\text{norm}}} + (1-\beta) \times \frac{\text{Wirelength}_{\text{cost}}}{\text{Wirelength}_{\text{norm}}}, \text{ while adaptive cost function adds in a modify weight } \alpha \text{ as}$$

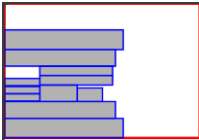
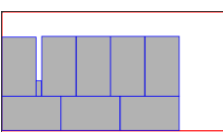
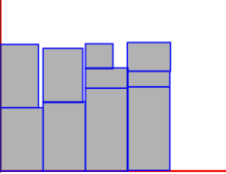
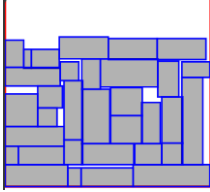
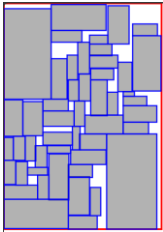
$$\text{cost} = \alpha \times \left(\beta \times \frac{\text{Area}_{\text{cost}}}{\text{Area}_{\text{norm}}} + (1-\beta) \times \frac{\text{Wirelength}_{\text{cost}}}{\text{Wirelength}_{\text{norm}}} \right) + (1-\alpha) \times (\text{panalty function}). \text{ While more}$$

floorplans are accepted in recent n steps, α rises and let the adaptive cost function get much closer to the origin function. On the contrary, the more floorplans are rejected, penalty rate rises to avoid SA to go for the wrong direction. This could help guiding SA, and effectively reduce the iterations, k , that it would take.

Results and findings

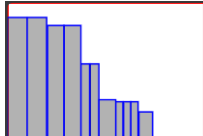
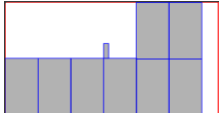
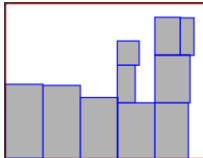
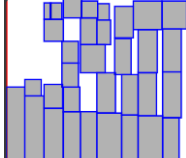
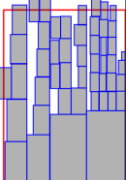
Result

$$\text{Cost function} = \alpha \times \text{Area}_{\text{cost}} + (1 - \alpha) \times \text{Wirelength}_{\text{cost}}, \alpha = 0.5$$

Dataset	hp	apte	xerox	ami33	ami49
Area	9852528	47313280	20497141	1290856	40484780
Wirelength	251846.0	748244.0	521607.5	75981.5	1139362.0
White space	10%	2%	6%	10%	12%
Iteration	1129	997	7	2970	1039
Runtime	1.05	0.76	0.15	22.70	20.86
Final cost	5052187.00	24030762.00	10509374.25	683418.75	20812071.00
Plot					

Initialization

In the beginning, I thought of a heuristic way to get valid solution while initializing. I simply check the aspect of the outline and decide whether the blocked should be placed vertically or horizontally as priority choice. Then I place it in DFS order flowing B*-tree criteria and get the results as follows.

Dataset	hp	apte	xerox	ami33	ami49
Area	13182960	69086160	32150664	1584464	45058440
Wirelength	335992.0	1026735.0	770282.5	145363.5	1637545.0
White space	33%	33%	40%	27%	21%
Valid	valid	valid	valid	valid	invalid
Plot					

I found out that compact initial solution would be hard to SA. SA will reject most of the perturbation and lead to cooling down fast without trying. Therefore, splattering the blocks will be a better initialization.

Penalty

I referred to TUNG-CHIEH 's paper and set my penalty function to $\text{penalty} = \text{outflow} + \text{white_space}$.

$$\text{outflow} = \left(\frac{\max(\text{chip_x}, \text{outline_x}) \times \max(\text{chip_y}, \text{outline_y})}{\text{Area}_{\text{outline}}} \right)^2$$

I squared the outflow penalty to make severer punishment on invalid solutions.

$$\text{white_space} = \left(1 - \frac{\sum_i \text{Block}_i}{\text{Area}} \right)^2, \text{ if white_space} < 15\%, \text{ then white_space} = 0$$

I set a 15% threshold. On one hand, I want to bonus the solutions with fewer white spaces, and on the other hand, to not over restrict the constraint and ignore the importance of wirelength. Most of the better results has low white space. The white_space bonus leads to find optimal solutions faster while weight α is still low.

References

TUNG-CHIEH, Chen; YAO-WEN, Chang. Modern floorplanning based on B*-tree and fast simulated annealing. IEEE Transactions on computer-aided design of integrated circuits and systems, 2006, 25.4: 637-650.

ADYA, Saurabh N.; MARKOV, Igor L. Fixed-outline floorplanning through better local search. In: Proceedings 2001 IEEE International Conference on Computer Design: VLSI in Computers and Processors. ICCD 2001. IEEE, 2001. p. 328-334.