# EXAM 2

# Floorplanning Based on B*-Tree and Fast Simulated Annealing

Akshaya Sharma Kankanhalli Nagendra

## 0. Abstract

The goal of this project is to build a floorplanner for circuit blocks that is aimed at minimizing area and wirelength.

The approach taken was using a binary tree to represent relationship between blocks and using fast simulated annealing to zero down on a good solution of arrangements of blocks with minimum area. Then based on the configuration a greedy approach places pins closest to the blocks it is connected to.

## 1. Introduction

Floorplanning is a very interesting problem in the VLSI field. Given a set of blocks(rectilinear hard blocks in the current project), floorplanning is a geometric problem of finding the best arrangements of these blocks such that the overall area enclosing the blocks and the wirelength of nets connecting blocks or blocks and a pin is minimized. A good arrangement allows less chip area and less wirelength that reduces delay and power.

From the B*-Tree and Fast SA based floorplanning a concise arrangements of blocks with very little deadspace was obtained. The wirelength was optimized with a greedy approach that gave reasonably good results.

## 2. Algorithm and Data Structures

### 2.1 B*-Tree

An Ordered Binary Tree is used to maintain the relations of blocks relative to each other.

- The root of the tree represents the block at the origin, that is the block whose leftmost corner is at co-ordinate (0,0).
- The left child corresponds to the block to the right of the parent block.
- The right child corresponds to the block above the parent block.

The x-coordinate of each block is determined using the relations from the tree starting from the root node.

If a node $n_j$ is the left child of $n_i$, then the block $b_j$ is located to the right hand side of block $n_i$ and has a x-coordinate $x_j=x_i+w_i$ where $w_i$ is the width of block $b_i$.

If a node $n_j$ is the right child of $n_i$, then the block $b_j$ is located above block $n_i$ and has a x-coordinate $x_j=x_i$.

In the code, the tree is saved as a vector of module structures that contain properties of each block and the parent<->child relation is maintained through indexes as parent(i)=floor(i/2), left(i)=(2*i) and right(i)=((2*i)+1).

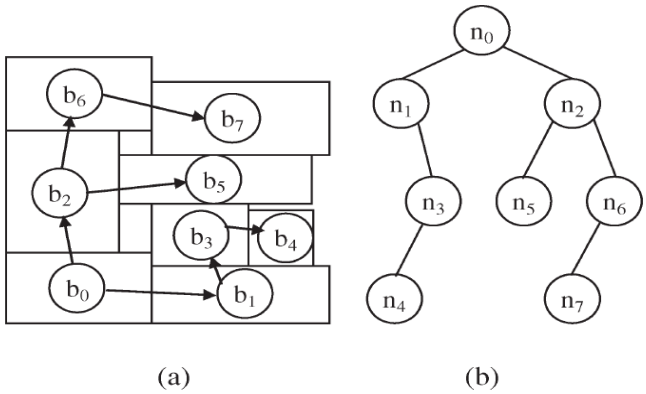


Fig 2.1 (a) Block configuration (b) Corresponding B*-Tree representation

To find the y-coordinate a data structure called contour is used, which is explained in the next section.

**2.2 Contour**

This is a data structure that maintains the contour of the current configuration which can be used to pick the y-coordinate for the next block without violating the no-overlap constraint.

The data structure is a vector of coordinates that maintain the outermost coordinates enclosing the current block configuration. As the new block whose coordinates are to be determined lies beyond the current configuration in relation to an already present block, we can pick the y-coordinate from the contour structure in O(1) time.

From the visual representation of the contour shown below, the edges where the contour is defined are the coordinate elements in the contour data structure.

| Module No. | Width | Height |
|---|---|---|
| 1 | 9 | 6 |
| 2 | 6 | 8 |

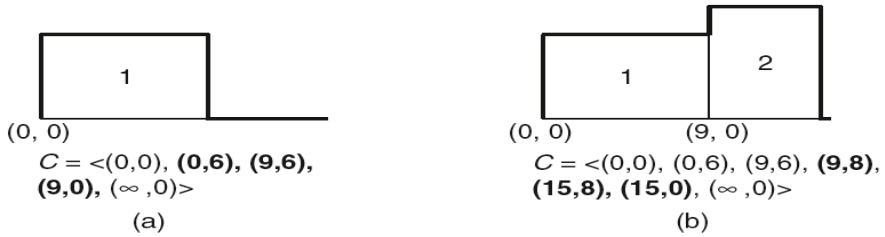


Fig 2.2.1 Example of contour vector for 2 blocks

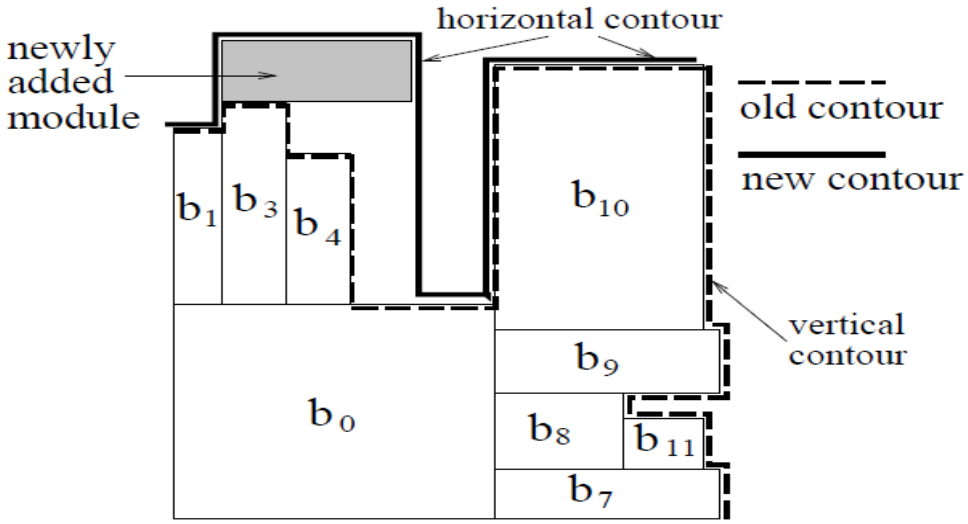


Fig 2.2.2 Visual representation of the contour

### 2.3 Fast Simulated Annealing Algorithm

Simulated Annealing is one of the most used algorithms in floorplanning. It involves starting with an initial solution and a high temperature. The temperature is then cooled over iterations when many random possibilities of configurations are explored and accepted with probability 1, if there is a cost reduction and accepted with a degrading probability if the cost gets worse. The problem with conventional SA is that initially a lot of time is spent on exploring solutions that are usually worse. Hence, a fast simulated annealing algorithm is used to overcome this drawback.

A fast simulated annealing algorithm has three stages:

1) high-temperature random search; for few initial iterations

At this stage, the temperature is set to a very high value and some random solutions are explored for the best cost configuration. This is done for a very few iterations as the likeliness of bad solutions in this stage are very high. An initial probability of accepting inferior solutions is set for this stage.

2) Pseudo-greedy local search stage; between a set of user defined iteration

At this stage the temperature is drastically dropped to a very low temperature to accept very few inferior solutions.

3) Hill-climbing search stage; for the rest of iteration

Here the temperature is again raised back up to facilitate uphill moves for significant amount of iterations so that a minimum cost solution can be found.

$$T_n = \begin{cases} \frac{\Delta_{\text{avg}}}{\ln P} & n = 1 \\ \frac{T_1 \langle \Delta_{\text{cost}} \rangle}{nc} & 2 \le n \le k \\ \frac{T_1 \langle \Delta_{\text{cost}} \rangle}{n} & n > k. \end{cases}$$

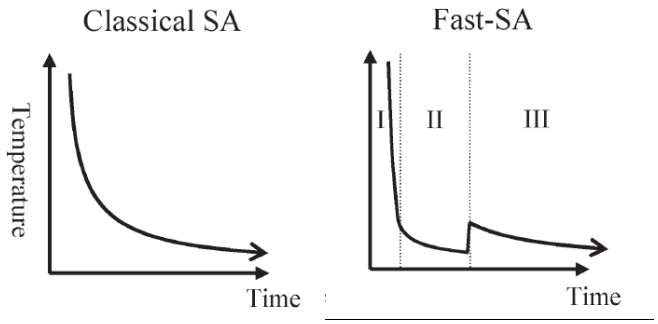


Fig 2.3.1 (a) Temperatures at the three different stages (b) Temp v/s Time for classical SA (c) Temp v/s Time for Fast SA

The Fast-SA Algorithm involves searching random solution space to get different configurations and look for the best configuration. This involves a perturbation method that changes the current configuration to something new and the new configuration's cost is calculated to see if it is better than the previous solution or worse.

The perturbation has two possibility for our hard rectilinear blocks:

1) Swap parent and child node: For this operation we randomly select a node and swap out the properties of the node and its child node (left or right is again chosen randomly or depending on whether it has a left or right child)

2) Delete and Insert node: In this operation we randomly select a node and delete it from its current position. We then select a new position and add the node at that position. Deleting a node changes for a leaf node(simply delete it), a node with one child(delete node and replace it by its child), a node with two children(delete node and replace it by either its left or right child and do this all the way down that branch of tree till a leaf node is found).

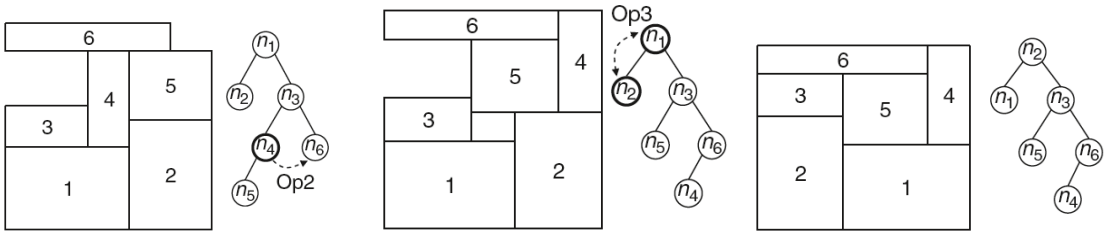


Fig 2.3.2 (a) Delete and insert operation (b) Swapping of two nodes (c) Final result

## 3. Results Table

| Block Size | Area | Wirelength | Execution Time(secs) |
|---|---|---|---|
| 10 | 239540 | 23837 | 0.15 |
| 30 | 222712 | 68628.5 | 3.59 |
| 50 | 204828 | 118346.5 | 9.59 |
| 100 | 184960 | 209833 | 21.13 |
| 200 | 183967 | 464325.5 | 65.97 |
| 300 | 289836 | 670680.5 | 497.879 |

## 4. Floorplan Figures

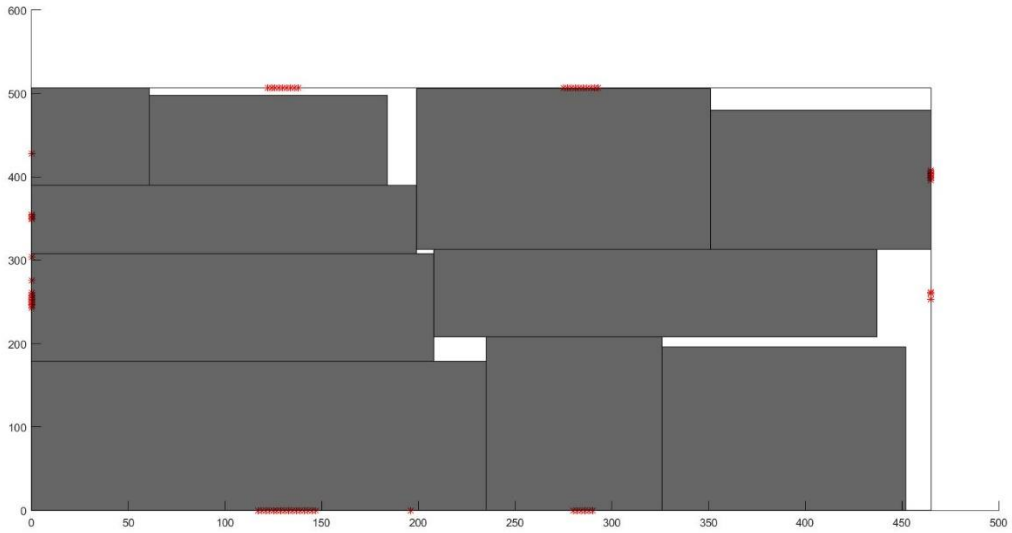


**Fig 4.1 Floorplan for B10**

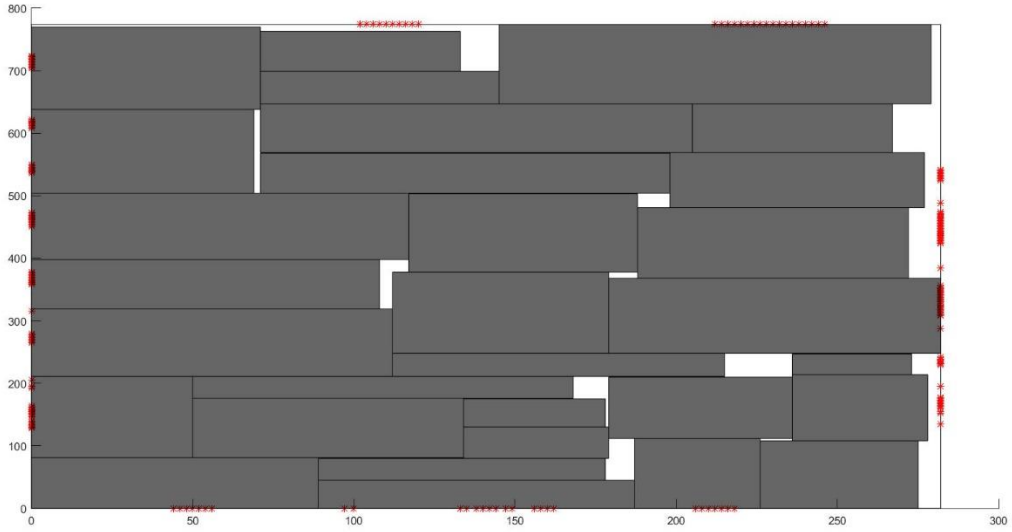


**Fig 4.2 Floorplan for B30**

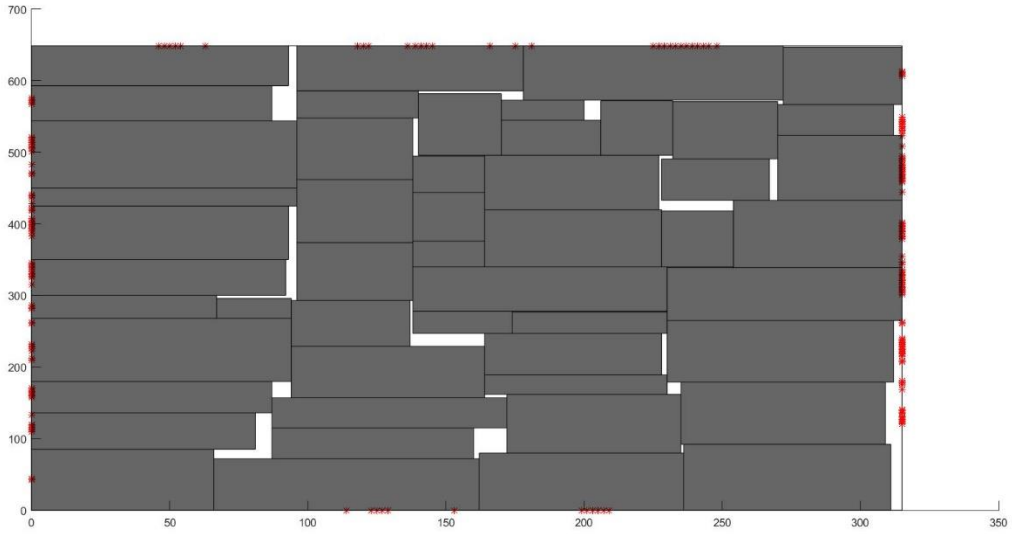


**Fig 4.3 Floorplan for B50**

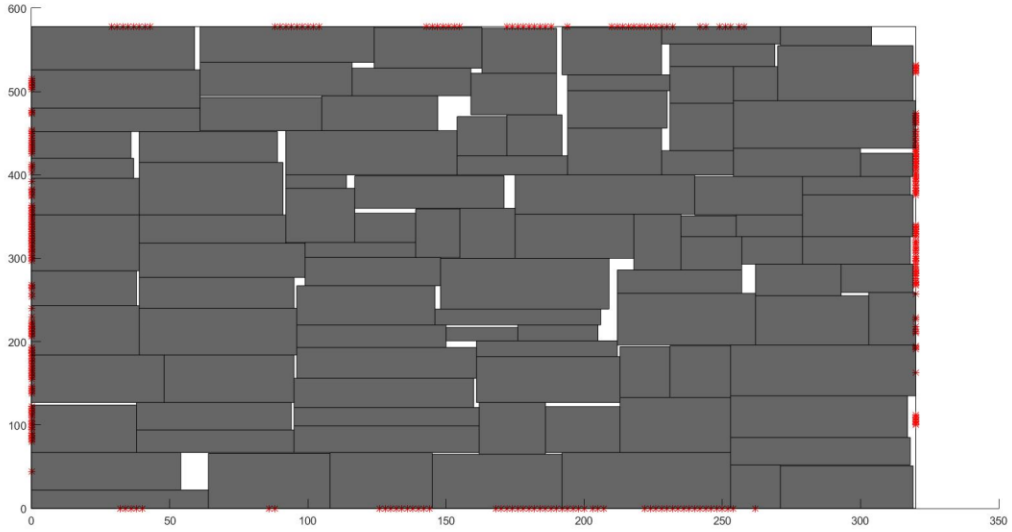


**Fig 4.4 Floorplan for B100**

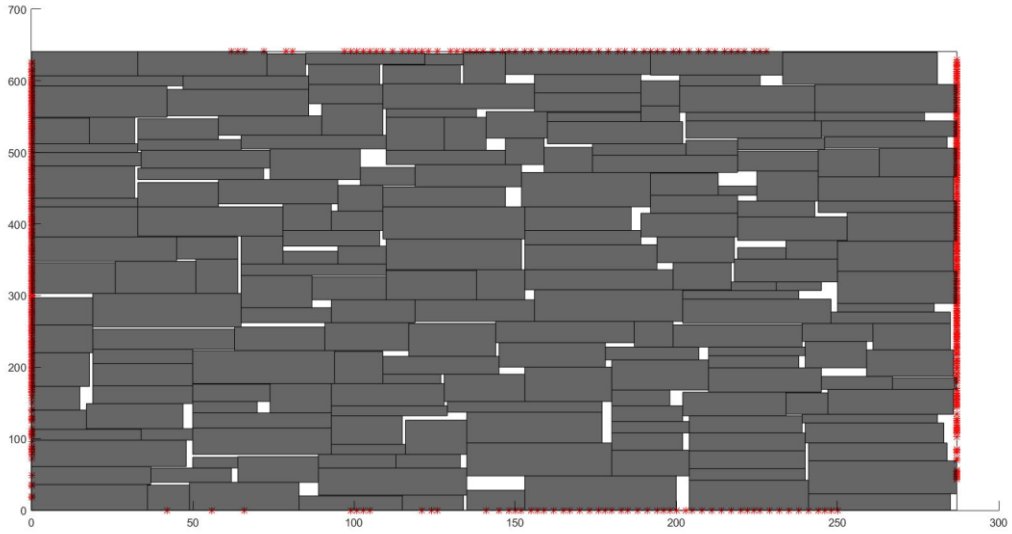


**Fig 4.5 Floorplan for B200**

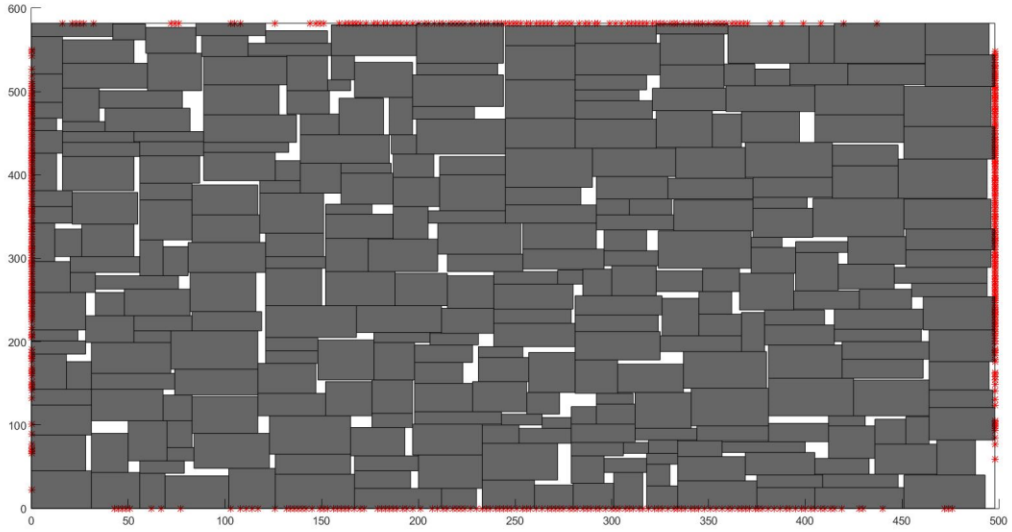


**Fig 4.6 Floorplan for B300**

## 5. Conclusion

A floorplanning algorithm based on B*-Tree and Fast SA was implemented in C++ and good packing results were obtained with optimized area and wirelength.

This can be further improved by combining the factor of wirelength in the process of simulated annealing and having a weight factor that lets us decide which must be optimized more. This gives a better wirelength solution than what is currently implemented with a reasonable increase in area.

## 6. References

[1] "*Modern Floorplanning Based on B*-Tree and Fast Simulated Annealing*" Tung-Chieh Chen, Student Member, IEEE, and Yao-Wen Chang, Member, IEEE

[2] http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD/EDA_floorplanning.pdf

[3] "*B*-Trees: A New Representation for Non-Slicing Floorplans*" Yun-Chih Chang, Yao-Wen Chang, Guang-MingWu, and Shu-Wei Wu

[4] Header files developed by Jer-Ming Hsu and Hsun-Cheng Lee under Yao-Wen Chang.