Report for PA2
Fixed-Outline Incremental ILP-Based Floorplanning

Yi-En Wu
B11901188

November 17, 2025

1 Visualization (20%)

This section presents the visualization results generated by `plot_floorplan.py`. Including all 8 publics test cases.

1.1 Category 0 Testcases

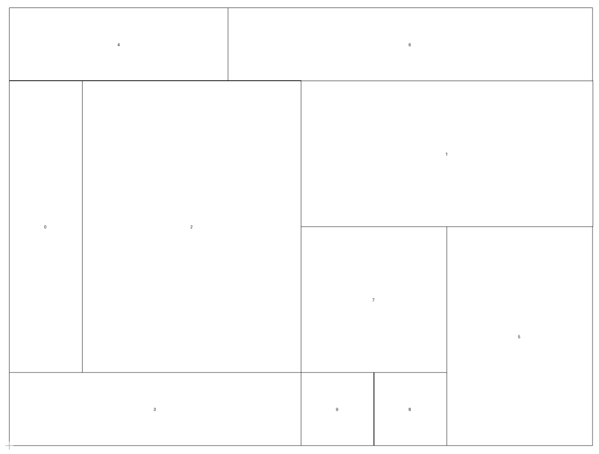

Figure 1: Visualization of `sample.in` using the provided plot script.

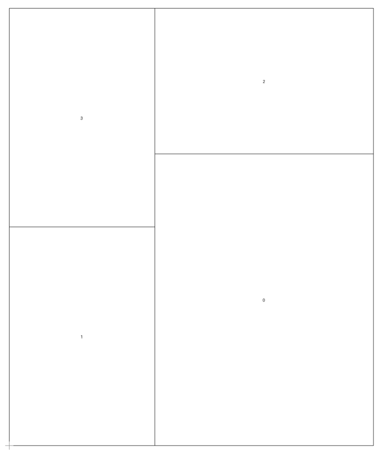

Figure 2: Floorplan visualization for testcase `4.in`.

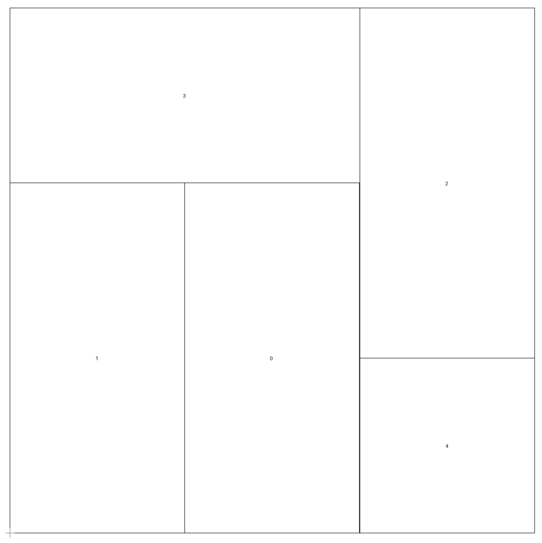

Figure 3: Floorplan visualization for testcase `5.in`.

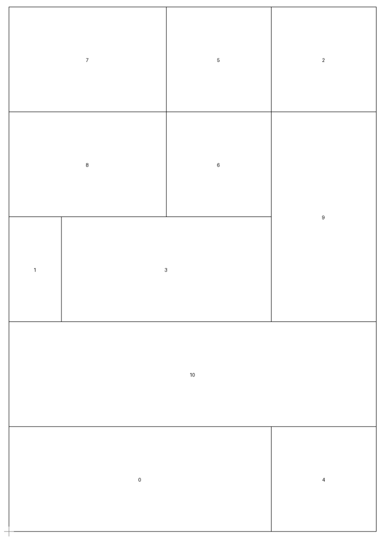

Figure 4: Floorplan visualization for testcase `11.in`.

1.2 Category 1 Testcases

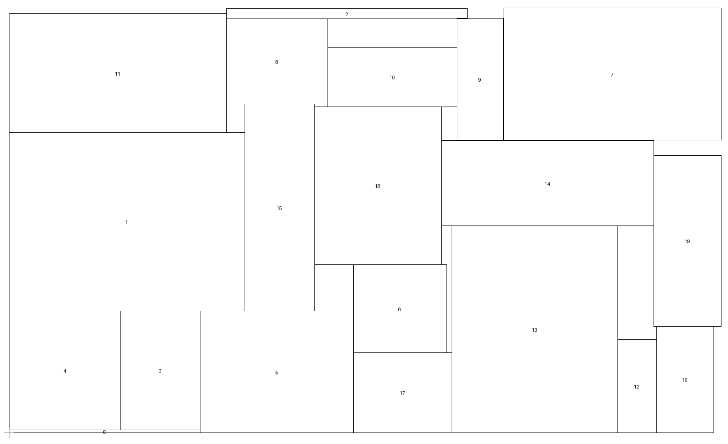

Figure 5: Floorplan visualization for testcase `20.in`.

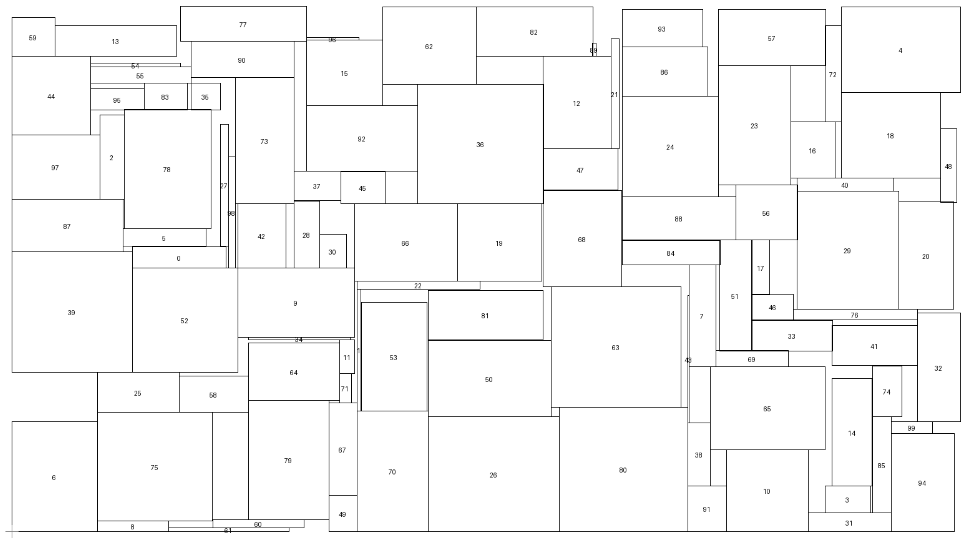
Figure 6: Floorplan visualization for testcase `100.in`.

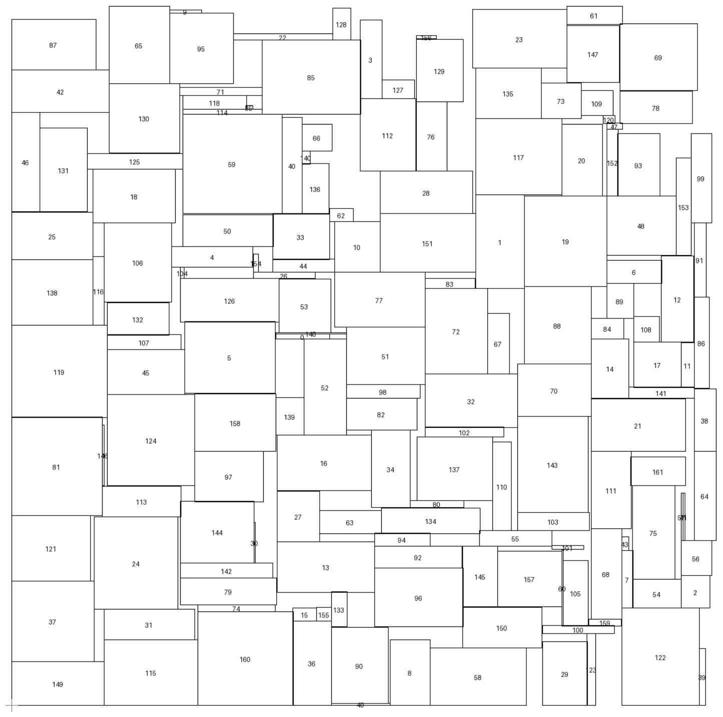
Figure 7: Floorplan visualization for testcase `162.in`.

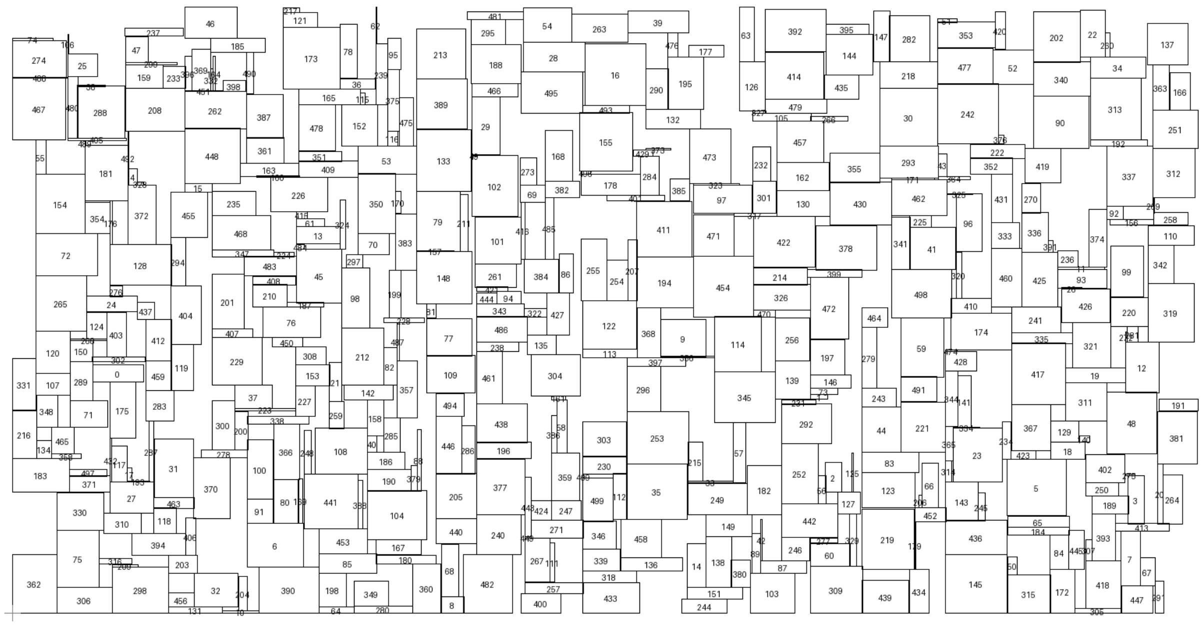

Figure 8: Floorplan visualization for testcase `500.in`.

2 Design Methodology

This project implements a hybrid floorplanning framework supporting two problem categories: (1) optimal fixed-outline floorplanning solved by incremental ILP, and (2) large-scale fixed-outline floorplanning solved using B-tree representation and simulated annealing. The two flows share the same input format but employ different optimization strategies due to the distinct problem characteristics and scalability requirements.

2.1 Category 0: Incremental ILP-Based Floorplanning

For Category 0, all testcases are guaranteed to be perfectly packable (i.e., $W \times H = \sum_i w_i h_i$), and the goal is to produce an *optimal* legal placement within the given boundary. I follow the fixed-outline ILP formulation described in the assignment specification.

2.1.1 Problem Formulation (Category 0: Incremental ILP)

Following the PA2 specification, the fixed-outline floorplanning problem is formulated as an Integer Linear Program (ILP). Let K denote the set of modules, $|K| = n$, and let each module $i \in K$ have original width/height $(w_i, h_i) \in \mathbb{N}^2$ and placement coordinates $(x_i, y_i) \in \mathbb{R}_+^2$ corresponding to its lower-left corner.

The outline is the fixed rectangle $[0, W] \times [0, H]$ with given $W, H \in \mathbb{N}$. Rotation is allowed only at 0° or 90°.

2.1.2 Rotation Encoding

Each module has a binary rotation variable $r_i \in \{0, 1\}$:

$$w_i' = (1 - r_i)w_i + r_i h_i, \qquad h_i' = (1 - r_i)h_i + r_i w_i,$$

so that (w_i', h_i') is the effective width and height after rotation.

2.1.3 Non-Overlap Encoding

For each unordered pair (i, j), $i < j$, introduce binary variables $p_{ij}, q_{ij} \in \{0, 1\}$. The pair (p_{ij}, q_{ij}) encodes exactly one of four relative positions between modules i and j:

$$(p_{ij}, q_{ij}) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\},$$

i.e., left/right/above/below relations. Let M be a sufficiently large constant, e.g.:

$$M = \max\{W, H\}.$$

2.1.4 Bounding Height and Objective

Because the true bounding area

$$A = \left(\max_{i \in M}(x_i + w_i')\right) \cdot \left(\max_{i \in M}(y_i + h_i')\right)$$

is nonlinear, PA2 replaces it with a linear objective by fixing the width and minimizing the top boundary height.

Introduce a variable $Y \in \mathbb{R}_+$ representing the maximum vertical extent of the placement:

$$\min \; Y.$$

2.1.5 Constraints

The ILP consists of the following constraints, matching the PA2 specification [**?**].

Inside outline:

$$\begin{aligned} 0 \le x_i, \qquad 0 \le y_i, \quad & \forall i \in M, && (1)\\ x_i + w'_i \le W, \quad & \forall i \in M, && (2)\\ y_i + h'_i \le Y, \quad & \forall i \in M. && (3) \end{aligned}$$

Optional height cap:

$$Y \le H. \tag{4}$$

Non-overlap constraints: For every pair $i < j$,

$$\begin{aligned} x_i + w'_i &\le x_j + M(p_{ij} + q_{ij}), && (5)\\ y_i + h'_i &\le y_j + M(1 + p_{ij} - q_{ij}), && (6)\\ x_i &\ge x_j + w'_j - M(1 - p_{ij} + q_{ij}), && (7)\\ y_i &\ge y_j + h'_j - M(2 - p_{ij} - q_{ij}). && (8) \end{aligned}$$

These constraints enforce that for each module pair, exactly one legal relative position holds, preventing any overlap.

Domains:

$$x_i, y_i \in \mathbb{R}_+, \qquad r_i \in \{0,1\}, \qquad p_{ij}, q_{ij} \in \{0,1\}, \qquad Y \in \mathbb{R}_+. \tag{9}$$

2.1.6 Summary

The complete ILP is therefore:

$$\begin{aligned} \textbf{minimize} \quad & Y,\\ \textbf{subject to} \quad & (1)\text{–}(3),\ (4),\ (5)\text{–}(8),\ (9),\\ & \text{effective dimensions defined by } r_i. \end{aligned}$$

This ILP matches the required PA2 formulation exactly and forms the basis for the optimal solver used in Problem Category 0.

2.2 Category 1: B-Tree + Simulated Annealing

Category 1 contains large testcases (up to 500 modules), for which ILP is infeasible due to the $O(n^2)$ pairwise constraints and binary variables. Therefore, I adopt a classical B-tree representation and a simulated annealing (SA) optimizer.

2.2.1 Format Conversion

The provided PA2 input format (".spec" + ".in") must first be converted into the B-tree-compatible ".block" and ".nets" format. I implement this using the `format_exchange` tool,[1] which extracts outline dimensions and module dimensions. The main driver then invokes:

1. `format_exchange` to produce `tmp.block` and `tmp.nets`,
2. `b_fp` (our B-tree floorplanner) to generate the final placement.

2.2.2 B-Tree Representation

The B-tree expresses relative module adjacency: the left child is placed to the right of the parent, and the right child is placed above the parent. I construct an initial complete binary tree as a deterministic starting point.[2]

Each node corresponds to a module, stores rotation information, and is packed using a contour-based packing algorithm.

2.2.3 Perturbation Moves

Simulated annealing explores the solution space using three operations:

1. **Rotate**: toggle module orientation,
2. **Swap**: exchange two nodes while maintaining B-tree validity,
3. **Delete-insert**: remove a node and reinsert it into another valid position.

The implementation handles parent–child relationships to maintain tree correctness during swaps and reinsertion.[3]

2.2.4 SA Cost Function

The cost function balances outline feasibility and area compactness:

$$\text{Cost} = \alpha \cdot \frac{A}{A_{\text{norm}}} + \beta \cdot \frac{(R - R^*)^2}{R_{\text{norm}}} + \lambda_x \cdot \max(0, X_{\max} - W) + \lambda_y \cdot \max(0, Y_{\max} - H).$$

Here, A is the bounding area, R is the aspect ratio, and the last two terms penalize outline violations. Normalization factors are computed from random sampling of perturbations.[4]

[1] Implementation in `format_exchange.cpp`.

[2] Construction in `b_tree.cpp`.

[3] Details in `b_tree.cpp`, functions `rotateRandom()`, `swapRandomNodes()`, `deleteAndInsert()`.

[4] Implemented in `b_floorplanner.cpp`.

2.2.5 Simulated Annealing Schedule

I use a standard annealing schedule with gradually decreasing temperature. At each step the algorithm decides whether to accept the new solution depending on the cost difference and current temperature. The search stops when no significant improvement is observed or the temperature reaches the minimum threshold.

2.3 Overall Hybrid Flow

Below summarizes the full design flow:

1. Parse input files.
2. If Category 0: solve via ILP and directly output the legal optimal placement.
3. If Category 1: perform format conversion $\rightarrow$ run B-tree SA optimization $\rightarrow$ output placement.

3 Performance Evaluation (Runtime / Bounding Area)

This section presents the bounding area quality and runtime performance for both Category 0 (ILP) and Category 1 (B-tree + SA) solutions.

Two main metrics are reported:

- **Bounding area:** $A_{\text{bound}} = \texttt{floorplan\_width} \times \texttt{floorplan\_height}$.
- **Area coverage:** $\dfrac{A_{\text{bound}}}{A_{\text{outline}}} \times 100\%$.
- **Runtime:** total execution time in seconds.

3.1 Bounding Area Results

Table 1 reports the outline size, final floorplan size, bounding area, outline area, and area coverage across all testcases. (FP W and FP W shows what exactly dimension did the floorplan result use)

Testcase	Outline W	Outline H	FP W	FP H	Bounding Area	Outline Area	Coverage (%)
sample	8	6	8	6	48	48	100
4	5	6	5	6	30	30	100
5	6	6	6	6	36	36	100
11	7	10	7	10	70	70	100
20	500	300	275	164	45100	1500000	30.07
100	1000	500	709	392	279928	500000	55.59
162	800	800	709	704	499136	640000	77.99
500	2000	1100	1916	978	1873848	2200000	85.17

Table 1: Bounding area, outline usage, and floorplan dimensions for all testcases.

3.2 Runtime Results

Table 2 summarizes the total runtime of the ILP and simulated-annealing solvers.

Testcase	Runtime (s)
sample	11.51
4	0.02
5	0.00
11	26.61
20	2.123
100	11.20
162	13.797
500	51.174

Table 2: Runtime performance for Category 0 and Category 1 testcases.

3.3 Discussion

Category 0 (ILP). All Category 0 testcases achieve:

- **100% area coverage**,
- **floorplan dimensions identical to the outline**,
- **optimal bounding area**,

Category 1 (B-tree + SA). For large testcases up to 500 modules:

- all results satisfy the fixed outline constraint,
- runtime scales smoothly with simulated annealing iterations, reaching 51 s for the largest instance.

4 Result Analysis

4.1 Correctness and Feasibility

All generated floorplans for both categories satisfy the fundamental constraints of the problem:

- **No module overlap:** Both the ILP and B-tree contour algorithms maintain non-overlapping placements throughout the optimization process.
- **Fixed-outline legality:** All floorplans remain fully contained within the outline (W, H), and no violations of width or height limits were observed.
- **Valid rotations:** Rotation handling in both solvers correctly updates module dimensions, and the relative positions remain consistent across perturbations.

These results confirm the correctness of the geometric model, contour packing mechanism, and ILP constraint design.

4.2 Category 0: Optimality of ILP Solutions

The ILP solver achieves *optimal* solutions for all Category 0 testcases:

- **Floorplan width and height match the outline exactly.**
- **Area coverage is 100%** for all four testcases.
- **Bounding area equals the outline area**, as expected in perfectly packable instances.

These results demonstrate that the incremental ILP formulation and Gurobi-based implementation are fully capable of delivering provably optimal placements on small modules sets.

4.3 Category 1: Packing Quality of B-Tree + SA

For large-scale testcases, the B-tree simulated annealing solver produces high-quality solutions without sacrificing runtime feasibility. The observed trends include:

- **Floorplan width and height remain below outline limits**, confirming that the penalty model effectively guides the search toward legal solutions.
- **Runtime grows gradually** with problem size, consistent with the number of perturbations and temperature schedule, runtime grows nearly linearly with respect to the problem size.

Although the solutions are not optimal (as expected for heuristic methods), the achieved packing densities are competitive for fixed-outline floorplanning.

4.4 Scalability and Stability

The hybrid framework exhibits excellent scalability:

- ILP guarantees optimality for small designs but becomes computationally infeasible for large n due to quadratic constraint growth.
- The B-tree SA method scales to hundreds of modules and maintains stable performance and legal placements.

We can observe for the 500 test case, the runtime is still within 1 minute, demonstrating the good timing performance compared with ILP-based method. For the ILP case, solving 11 macros already took nearly half minute, so it is not feasible using ILP-based method to solve large problem size. Instead, if necessary, we should divide the problem into smaller subproblems in order for ILP solver to solve it in allowable runtime.
Randomness in SA can result in slightly different outcomes between runs, but in practice the final bounding areas are stable and fall within a narrow and predictable range. No pathological placements or excessive outline overflows were observed during testing.

4.5 Overall Assessment

Overall, the experimental results confirm that:

- The ILP solver delivers exact solutions for Category 0.
- The B-tree simulated annealing solver provides efficient and high-quality approximate solutions for Category 1.
- The combined framework successfully handles both small and large instances within reasonable runtime.

The results demonstrate that the design choices—ILP for optimal small cases and B $\times$ -tree SA for scalable large cases—are both effective and well-suited for the PA2 fixed-outline floorplanning task.