National Taiwan University
Department of Electrical Engineering
Intro. to EDA, Fall 2025

October 15, 2025
Iris Hui-Ru Jiang

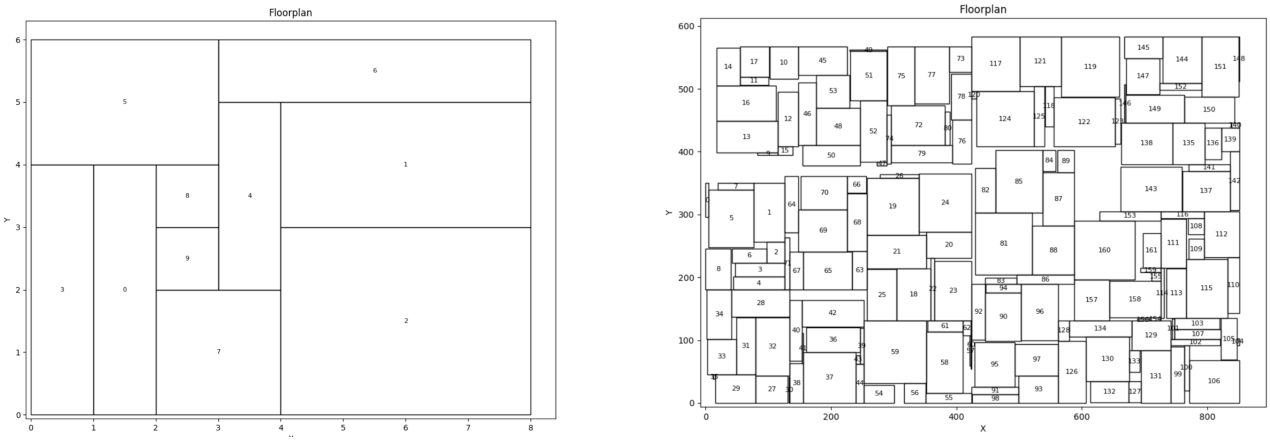
Programming Assignment #2 (due on-line, 23:59, November 19, 2025)

Problem: Fixed-Outline Incremental ILP-Based Floorplanning

As the complexity of modern integrated circuits continues to grow, floorplanning has become a critical step in the VLSI design flow. Floorplanning determines the relative positions and shapes of functional modules within a chip outline, providing early feedback on system architecture, chip area, and potential routing congestion. A well-designed floorplan not only reduces design iterations but also ensures that physical constraints are satisfied in downstream placement and routing stages

One of the key challenges in floorplanning is the fixed-outline constraint, which arises because the chip dimensions are typically predetermined at early design stages. Unlike unconstrained floorplanning, where the overall chip shape may expand to minimize area, fixed-outline floorplanning requires that all modules must be legally placed within a given rectangular boundary. This reflects practical manufacturing requirements and packaging limitations. Without handling this constraint properly, even floorplans with minimized area may fail to map onto the actual chip outline

In this work, we focus on the ILP-based Floorplanning method. The primary objective is to minimize the bounding area of all modules within the given outline. By applying ILP in an incremental and recursive manner, the floorplanner can refine partial solutions step by step, ensuring that the final placement respects the fixed-outline constraint while approaching minimum bounding area.



(a) Floorplanning with minimum bounding area (Problem Category 0)

(b) Floorplanning with optimized result (Problem Category 1)

Figure 1: Floorplanning results.

Objective and Specification

Let $\mathcal{M}$ denote the set of modules with $|\mathcal{M}| = n$. Each module $i \in \mathcal{M}$ has an original width $w_i \in \mathbb{N}$ and height $h_i \in \mathbb{N}$ and is placed at coordinates $(x_i, y_i) \in {\mathbb{R}_+}^2$ corresponding to its *lower-left corner*. Let the fixed-outline (boundary) be the axis-aligned rectangle $[0, W] \times [0, H]$ with $W, H \in \mathbb{N}$ and lower-left corner fixed at $(0, 0)$.

Each module is rotatable by 90°. We encode rotation with a binary 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/height of module i after rotation.

Bounding area. Define the bounding area

$$A \;=\; \Big(\max_{i\in\mathcal{M}}\big(x_i + w_i'\big)\Big) \;\cdot\; \Big(\max_{i\in\mathcal{M}}\big(y_i + h_i'\big)\Big).$$

Objective.

$$\min\; A$$

Subject to:

(Inside outline)	$0 \le x_i,\ 0 \le y_i,\ \ x_i + w_i' \le W,\ \ y_i + h_i' \le H,$	$\forall i \in \mathcal{M},$	(1)
(Non-overlap)	for all $i \ne j,$ rectangles i and j do not intersect,	$\forall i \ne j \in \mathcal{M},$	(2)
(Integrality and rotation)	$x_i, y_i, w_i, h_i, W, H \in \mathbb{N},\ \ r_i \in \{0,1\},$	$\forall i \in \mathcal{M}.$	(3)

All coordinates are integer-valued. The boundary's lower-left corner is fixed at $(0,0)$. Rotation is restricted to 0° and 90° only (no arbitrary angles).

ILP Formulation

Variables. Let $\mathcal{M}$ be the set of rigid modules, $|\mathcal{M}| = n$. Each module $i \in \mathcal{M}$ has original width/height $(w_i, h_i) \in \mathbb{N}^2$ and placement coordinates $(x_i, y_i) \in {\mathbb{R}_+}^2$ for its lower-left corner. A free 90° orientation is encoded by a binary $r_i \in \{0,1\}$:

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

so (w_i', h_i') is the effective size after rotation. For each unordered pair (i,j) with $i \ne j$, introduce binaries $p_{ij}, q_{ij} \in \{0,1\}$. The vector $(p_{ij}, q_{ij}) \in \{0,1\}^2$ encodes one of four non-overlap relations between i and j.

The fixed outline is the rectangle $[0,W] \times [0,H]$ with lower-left corner at $(0,0)$, where $W, H \in \mathbb{N}$ are given constants. Let M be sufficiently large constants example,

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

Objective (height minimization with fixed width). Since the area objective is non-linear, we instead fix the width and use a linear ILP with objective min Y (vertical extent). Introduce a variable $y_{\max} \in \mathbb{N}$ for the top bounding height and minimize it:

$$\min\; y_{\max}.$$

Constraints.

Inside outline:	$0 \le x_i, \quad 0 \le y_i,$	$\forall i \in \mathcal{M},$	(4)
	$x_i + w_i' \;\le\; W,$	$\forall i \in \mathcal{M},$	(5)
	$y_i + h_i' \;\le\; Y,$	$\forall i \in \mathcal{M},$	(6)
Optional height cap:	$Y \;\le\; \mathrm{H},$		(7)
Non-overlap:	$x_i + w_i' \;\le\; x_j + M\,(p_{ij} + q_{ij}),$	$\forall i < j,$	(8)
	$y_i + h_i' \;\le\; y_j + M\,(1 + p_{ij} - q_{ij}),$	$\forall i < j,$	(9)
	$x_i \;\ge\; x_j + w_j' - M\,(1 - p_{ij} + q_{ij}),$	$\forall i < j,$	(10)
	$y_i \;\ge\; y_j + h_j' - M\,(2 - p_{ij} - q_{ij}),$	$\forall i < j,$	(11)
Domains:	$x_i, y_i \in \mathbb{R}_+,\ \ r_i \in \{0,1\},\ \ p_{ij}, q_{ij} \in \{0,1\},\ \ Y \in \mathbb{R}_+.$		(12)

Input

The input will consist of two files: a specification file with extension `.spec` and a module list file with extension `.in`.The module list file specifies the modules to be placed. The first line contains an integer n, which denotes the total number of modules. The second line is the header "ID W H". Each of the following n lines contains three integers that describe one module: its identifier ID, its width W, and its height H. All ID values are unique; W and H are positive integers.

The specification file describes the problem category and the boundary outline. The first line contains an integer "0" or "1", which indicates the category of the problem. The second line of the specification file contains two integers W and H, which denote the width and height of the boundary outline. If the value is "0", the task is to find an optimal solution. For this category, all test cases are guaranteed to admit a solution with 100% utilization; in other words, the outline area equals the total module area, that is, $W \times H = \sum_{k=1}^{n} W_k H_k$, and the width and height of the given outline will match exactly the required dimensions (see Figure 1.a and sample input). In this type of problem, the number of modules satisfies $4 \leq n \leq 12$. If the value is "1", the task is to find a solution that satisfies the fixed-outline constraint with the minimum possible bounding area, without requiring optimality. For this category, the number of modules satisfies $20 \leq n \leq 500$.

Output

The output file should contain exactly n lines, each describing one placed module in the form `ID posX posY r`. Here, `ID` is the module identifier from the input(.in), `posX` and `posY` are coordinates of the module's *lower-left corner* in the global outline coordinate system, and $r \in \{0, 1\}$ is a rotation flag where $r = 0$ means no rotation and $r = 1$ means the module is rotated by 90° *counter-clockwise* (i.e., width and height are swapped). The global outline is the rectangle $[0, W] \times [0, H]$ given in the `.spec` file. All modules must lie entirely within the outline and must be pairwise non-overlapping. The n lines can appear in arbitrary order, and no additional headers or trailers are required.

Here are some input examples and valid output examples:

sample.in	**sample.spec**	**sample.out**	**162.in**	**162.spec**	**162.out**
MODULE_SIZE 10	0	0 0 4 1	MODULE_SIZE 162	1	0 549 146 1
ID W H	8 6	1 4 4 0	ID W H	800 800	1 488 247 1
0 1 4		2 0 0 1	0 5 54		$\widehat{2}$ 488 146 1
1 4 2		3 3 3 1	1 49 94		3 524 151 1
2 4 3		4 1 5 0	2 29 33		4 582 173 0
3 1 4		5 3 1 0	3 22 79		5 491 175 1
4 3 1		6 3 0 0	4 21 82		6 691 0 0
5 3 2		7 6 1 0	5 72 91		7 635 46 1
6 5 1		8 0 5 0	6 23 55		8 648 61 1
7 2 2		9 7 3 0	7 58 11		9 682 101 1
8 1 1			.		.
9 1 1			.		.
			.		.

Command-line Parameter:

The executable binary must be named as **'fp"** and use the following command format.

```
./bin/fp <input>.in <input>.spec <output_file_name>
```

For example, if you would like to run your binary for the input file `5.in`,`5.spec` and generate a solution named `5.out`, the command is as follows:

```
./bin/fp 5.in 5.spec 5.out
```

Plotter

After you successfully generate the output files, you can use `plot_floorplan.py` to visualize the result. Assume your testcase input is `200.in` (with `200.spec`) and your program produces `200.out`. Run the following command from the project root:

```
python ./plot_floorplan.py 200.in 200.out
```

Compilation and Gurobi Package

For this assignment, we provide the template code, the Gurobi package, and a corresponding makefile. Since Gurobi requires a license, please obtain a *Web License Service (WLS)* license from:

- Academic eligibility / request: `https://www.gurobi.com/academia/academic-program-and-licenses/`

After you receive the license key, save it as `gurobi.lic` in the project root directory.

Build instructions. We expect that your code can be compiled and run as follows. Type the following command under the <`student_id`>`_pa2/` directory:

```
make
```

Resulting directory structure. After compilation, the directory structure should be as follows:

```
PA2/
├── bin/
├── build/
├── include/
├── input/
├── spec/
├── src/
├── gurobi1203/
└── gurobi.lic
```

Notes on the provided makefile. We provide a makefile. To add files, place your header files (*.h) under `include/` and your source files (*.cpp, *.c) under `src/`. When you need to compile, simply run `make`. This will create two new directories: `bin/` and `build/`. The executable (`fp`) will appear under `bin/`.

Requirements if you modify the template code/makefile:

1. We must be able to build your project by running `make` from the project root.
2. The compiled executable must be placed under `./bin/`.
3. Your program must read the Gurobi license from the project root (pa2/); during evaluation we will place our gurobi.lic there and run your code from that location.

Required Files:

You need to create a directory named <**student_id**>**_pa2/** (*e.g.* b12901000_pa2/) **(the student ID should start with a lowercase letter)** which must contain the following documents:

- A directory named **src/** containing your source codes (*e.g.* `floorplanner.cpp`): only `*.h`, `*.hpp` are allowed in src/, and no directories are allowed in src/;

- A directory named **include/** containing your header files (*e.g.* `module.h`): only `*.c`, `*.cpp` are allowed in include/, and no directories are allowed in include/;
- A directory named **gurobi1203/** if you rely on the provided Gurobi package;
- A directory named **bin/** containing your executable binary named **fp**;
- A pdf file named **report.pdf**;
- A makefile named **makefile** that produces an executable binary from your source codes by simply typing "make": the binary should be generated under the directory <student_id>_pa2/bin/;
- A text readme file named **readme.txt** describing how to compile and run your program.
- A report named **report.pdf** on the data structures used in your program and your findings in this programming assignment.

We will use our own test cases and licence, so you **DO NOT** need to submit the input files and gurobi.lic. Nevertheless, you should have at least the following items in your *.tgz file.

```
src/<all your source code>
include/<all your header files>
gurobi1203/
bin/fp
report.pdf
makefile
readme.txt
```

Submission Requirements:

1. Your program must be compilable and executable on EDA union servers **U12 (port: 40062).**
2. **The runtime limit for each test case is 1 hour.**
3. **Please use the following command to compress your directory into a *.tgz* file:**

   ```
   tar zcvf <filename>.tgz <your directory>
   ```

   The submission file should be named as <student_id>_pa2.tgz (*e.g.* b12901000_pa2.tgz). For example, if your student ID is *b12901000*, then you should use the command below.

   ```
   tar zcvf b12901000_pa2.tgz b12901000_pa2/
   ```

4. **Please submit your *.tgz* file to the NTU COOL system before 13:00, June 12, 2025 (Thursday).**
5. **You are required to run the `checkSubmitpa2.sh` script to check if your *.tgz* submission file is correct. Suppose you are in the same level as the pa2 directory,**

   ```
   bash checkSubmitPA2.sh <your submission>
   ```

   For example,

   ```
   bash checkSubmitpa2.sh b12901000_pa2.tgz
   ```

Language/Platform:

1. Language: C or C++.
2. Platform: Linux.

Evaluation
There are 4 public test cases for *Problem Category 0* and 4 public test cases for *Problem Category 1*. In addition, grading will include 4 hidden test cases for each problem category. Within the same problem category, all test cases carry equal weight.

Grading Policy.

1. **Report** 20%
   - Visualization 20%
   - Design methodology 40%
   - Performance (runtime / bounding area) 20%
   - Result analysis 20%
2. **Category 0 — Optimal-fit instances (guaranteed 100% utilization)**
   - Correctness 50%
3. **Category 1 — Fixed-outline instances (area minimization)**
   - Correctness 15%
   - Performance 15%

   among all *valid* submissions, let rank $\in \{1, 2, \dots\}$ and valid_sub be the number of submissions with valid result. The score is

   $$\boxed{15\% \times \left(1 - \frac{\text{rank} - 1}{\text{valid\_sub}}\right)}$$

   (smaller area $\Rightarrow$ higher rank $\Rightarrow$ higher score).

For any questions, please email Yu-Hsiang Hunag at johnnyhuang1007@gmail.com. Thank you so much for your cooperation.