EDA大实验_floorplan

提交：

1. PPT汇报；
2. 实验内容或报告（word或PDF）；
3. 实验源代码和必要的简单说明(比如readme, 如何执行, 输入输出含义)。

报告

题目介绍

问题描述

在经过芯片划分后，整个芯片被分为若干个**Block**。在简单情况下，这些Blocks均可视为宽高比固定的可移动矩形。芯片上同样摆放着若干**Terminal**。不同Block之间，Block与Terminal之间会存在相互连接关系，称之为**Net**。

> "Terminal"（终端）是指在芯片布局中用于连接不同模块（Block）或与外部环境进行交互的接口

衡量芯片Floorplan的质量优劣往往采用面积（Area）与线长（Wirelength）两个指标。

- 芯片的面积（A）为所有Block接后的图形的上、下、左、右边界围成的面积，即能够包裹住Floorplan后所有Block的最小矩形面积。
- 而芯片的线长（W）为所有Net的半周长线长之和，即：

$$W = \sum_{n_i \in N} HPWL(n_i)$$

其中，N为所有网络的集合，n_i为N中的一个网络。
最终以加权求和的形式来计算芯片的📍总Cost

$$\text{Cost} = \alpha \frac{A}{A_{\text{norm}}} + (1-\alpha)\frac{W}{W_{\text{norm}}}$$

其中α为面积所占的权重，应在0~1范围内。Anorm、Wnorm分别为归一化面积与归一化线长。为简化计算，令Anorm为所有Block面积之和；而Wnorm为所有网络中的每个Block平均边长之和。

题目要求

1.遵循给定的输入输出格式，使用C/C++、python或matlab中一或多种语言编程实现一个简易Floorplanner，算法不限。

2.功能要求：
a)根据输入文件.block中的所有macro信息，将所有block均摆放在给定的Outline范围内，且不允许重叠；
b)在满足功能要求a）的基础上，根据输入文件.block与.net，计算芯片的Cost，并使得芯片Cost最

小；
c)在满足功能要求b）的基础上，设计一种方法能够尽可能使得每个网络中相邻的blocks更多，且相邻的边长更长。

3.功能要求中的a)，b)为必做内容，功能要求c) 为选做内容。

4.在报告中需说明程序运行方法与项目目录结构，在文件读写与脚本文件中使用相对路径，不要使用绝对路径。

输入输出

输入：.block与.net文件
•Block包含有芯片尺寸要求与所有Block与Terminal的输入信息
•Outline表示芯片的边界信息，最终floorplan后的结果不得超出Outline范围
•Block 输入信息包含其名称、宽度与高度
•Terminal 的输入信息包括其名称与坐标

•Net包含有芯片内所有互连关系的要求。一个Net中可能包含若干个Blocks和Terminals。

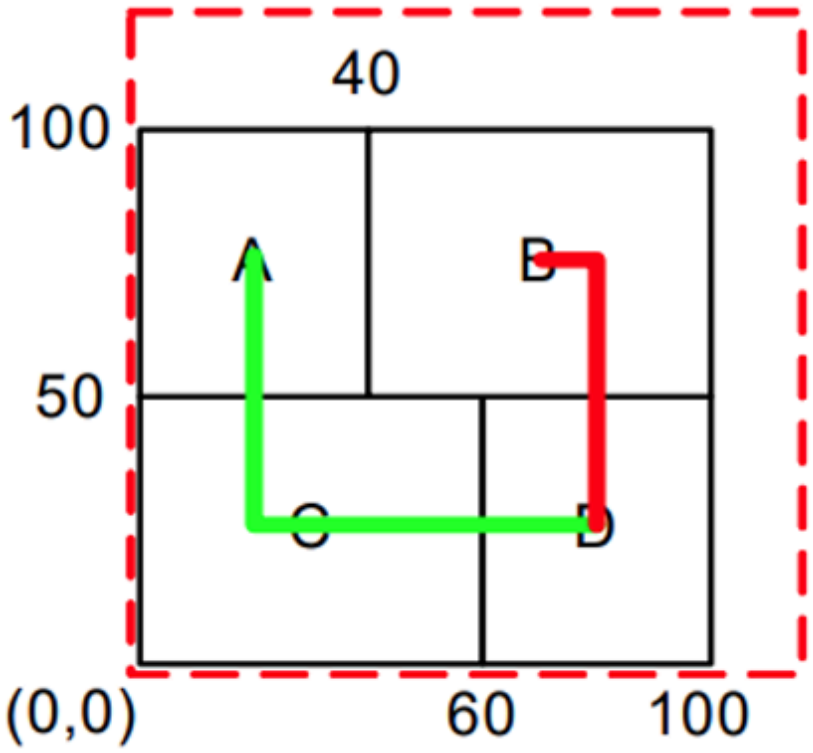
输出：.output文件

- 最终的Cost
- 总半周长线长HPWL
- 芯片面积
- 芯片的宽与高
- 程序运行时间（秒）
- 模块摆放信息

eg.

Input files (input.block):

```
Outline: 120 120
NumBlocks: 4
NumTerminals: 0
A    40    50
B    60    50
C    60    50
D    40    50
```



(input .nets)

```
NumNets: 2
NetDegree: 3
A
C
D
NetDegree: 2
B
D
```

```
5085
170
10000
100  100
0.24
A    0    50    40    100
B    40   50    100   100
C    0    0     60    50
D    60   0     100   50
```

Output files (output.rpt)

模拟退火

模拟退火（Simulated Annealing，SA）是一种受冶金退火过程启发的概率优化技术，常用于在大型搜索空间中找到全局最小值。

关键概念：

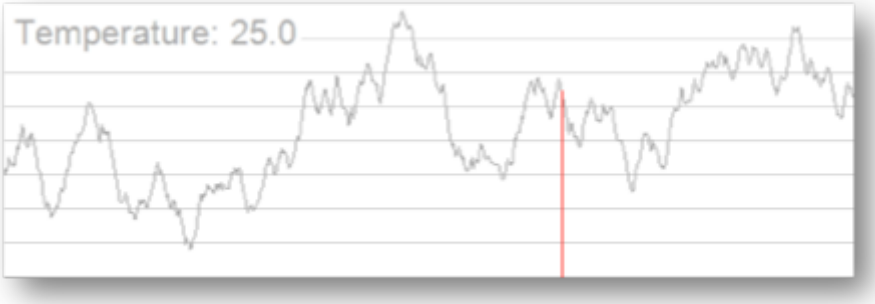
- 温度：控制接受较差解的概率。
- 冷却计划：逐渐降低温度以细化搜索。
- 扰动：应用随机变化以探索解决方案空间。

SA 算法步骤：

- 初始化：以初始解和高温度开始。

•迭代：在每个温度下：
•应用扰动生成新解。
•计算成本差异。
•根据接受概率决定是否接受新解。
•冷却：根据冷却计划降低温度。
•终止：当温度足够低或达到预定迭代次数时停止。

eg.(from wiki)



实现

建模

- Blocks（块）：存储在 Blocks 容器中，便于访问和操作。
- Terminals（终端）：存储在 Terminals 容器中。
- Nets（网络）：存储在 Nets 容器中。
- Outline（外形轮廓）：表示芯片的边界约束。

PYTHON

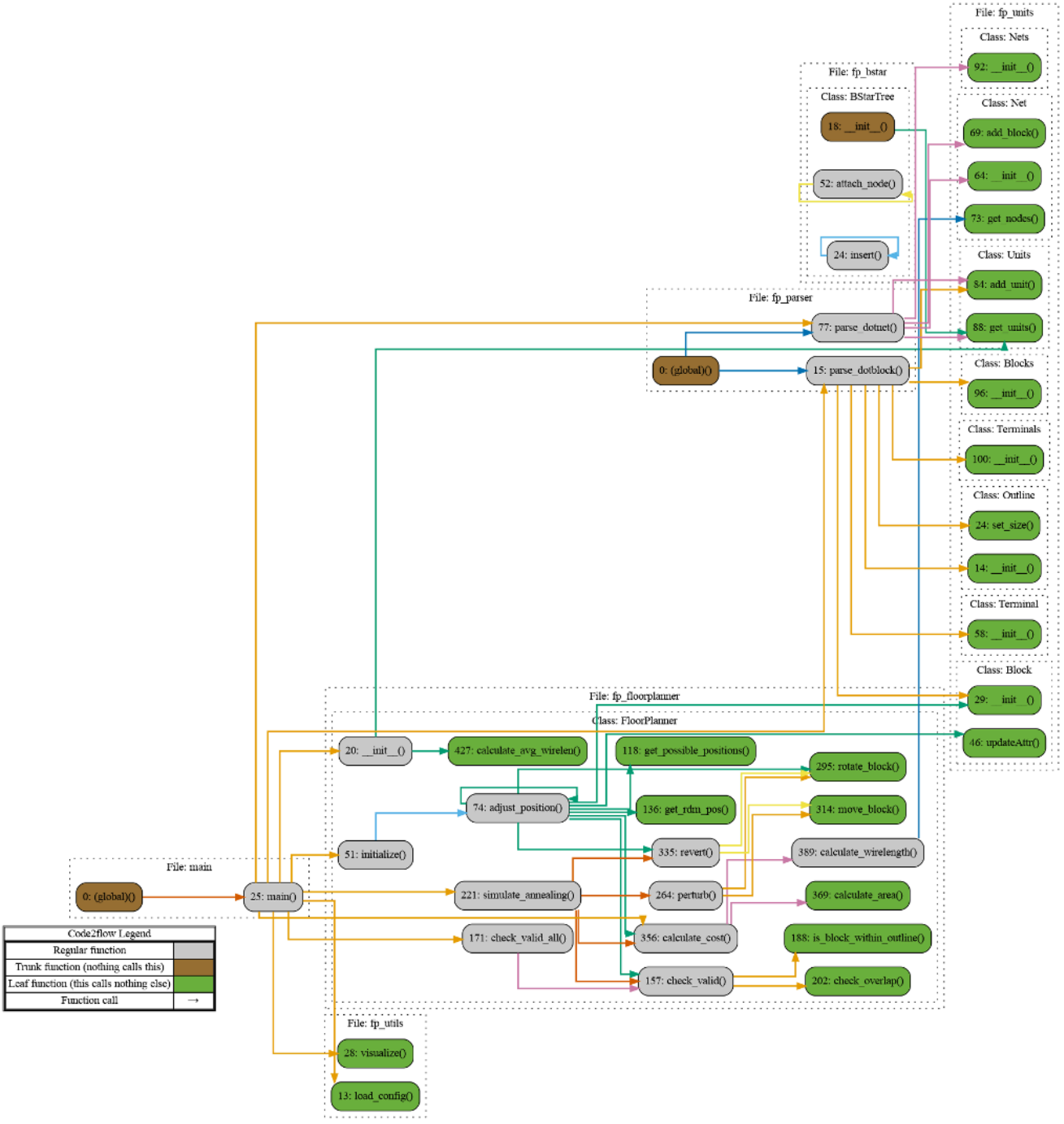
```python
class Outline:
    def __init__(self, width:int=0, height:int=0) -> None:
        self.w = width
        self.h = height

    def set_height(self, height:int) -> None:
        self.h = height

    def set_width(self, width:int) -> None:
        self.w = width

    def set_size(self, width:int, height:int) -> None:
        self.w = width
        self.h = height

class Block:
    def __init__(self, name=None, width=None, height=None, x=0,
y=0) -> None:
        self.name = name
        self.width = width
        self.height = height
        self.x = x
        self.y = y

        # 标记模块是否旋转
        self.rotated = False
        self.rotate_point = 0 # 0: 左下角, 1: 左上角, 2: 右上角, 3:
右下角
        self.placed = False

        # B*-树相关指针
        self.parent = None
        self.left = None
        self.right = None

    def updateAttr(self, block) -> None:
        if isinstance(block, Block):
            self.name = block.name
            self.width = block.width
            self.height = block.height
            self.x = block.x
            self.y = block.y
            self.rotated = block.rotated
            self.rotate_point = block.rotate_point
            self.placed = block.placed
```

```
class Terminal:
    def __init__(self, name, x, y) -> None:
        self.name = name
        self.x = x
        self.y = y

class Net:
    def __init__(self, name:str, degree:int=0) -> None:
        self.name = name
        self.nodes = []
        self.degree = degree

    def add_block(self, block) -> None:
        self.nodes.append(block)
        self.degree += 1

    def get_nodes(self) -> list:
        return self.nodes

class Units:
    def __init__(self, units:list=[], num_units:int=0) -> None:
        if len(units) != num_units:
            raise ValueError(f'Length of units {len(units)} does
not match num_units {num_units}')

        self.units = units
        self.num_units = num_units

    def add_unit(self, unit) -> None:
        self.units.append(unit)
        self.num_units += 1

    def get_units(self) -> list:
        return self.units

class Nets(Units):
    def __init__(self, nets:list=[], num_nets:int=0) -> None:
        super().__init__(nets, num_nets)

class Blocks(Units):
    def __init__(self, blocks:list=[], num_blocks:int=0) -> None:
        super().__init__(blocks, num_blocks)

class Terminals(Units):
    def __init__(self, terminals:list=[], num_terminals:int=0) ->
```

```
    None:
        super().__init__(terminals, num_terminals)
```

flow

1. 输入解析：从输入文件读取块和网络列表信息。
2. 数据结构初始化：为块、终端、网络和外形轮廓创建对象。
3. 初始 Floorplan 构建：构建初始初始可行解。
4. 模拟退火优化：使用 SA 优化 Floorplan。
5. 结果输出：将最终 Floorplan 写入文件并进行可视化。



⭐初始构建

优先放置大模块，根据每个块的面积（宽度 * 高度）对块进行排序，更好地利用空间并避免后续放置时出现空间不足的问题
第一个从坐标原点开始放置；
后续从已经放置的右边或上边放置 get_possible_positions(block)
若右边上边冲突，则 get_rdm_pos

⭐模拟退火优化

扰动操作（perturb）：
随机应用以下操作之一：

- 旋转：交换块的宽度和高度。
- 移动：将一个块移动到树中的不同位置(向下/向左)。

成本计算（calculate_cost）：

- 面积：块占用的总面积。
- 线长：所有网络的半周长总和。
- 成本函数：面积和线长的加权和（总cost公式）。

接受准则：
接受具有更低成本的新解。
以概率 P = exp(-ΔCost / Temperature) 接受较差的解。

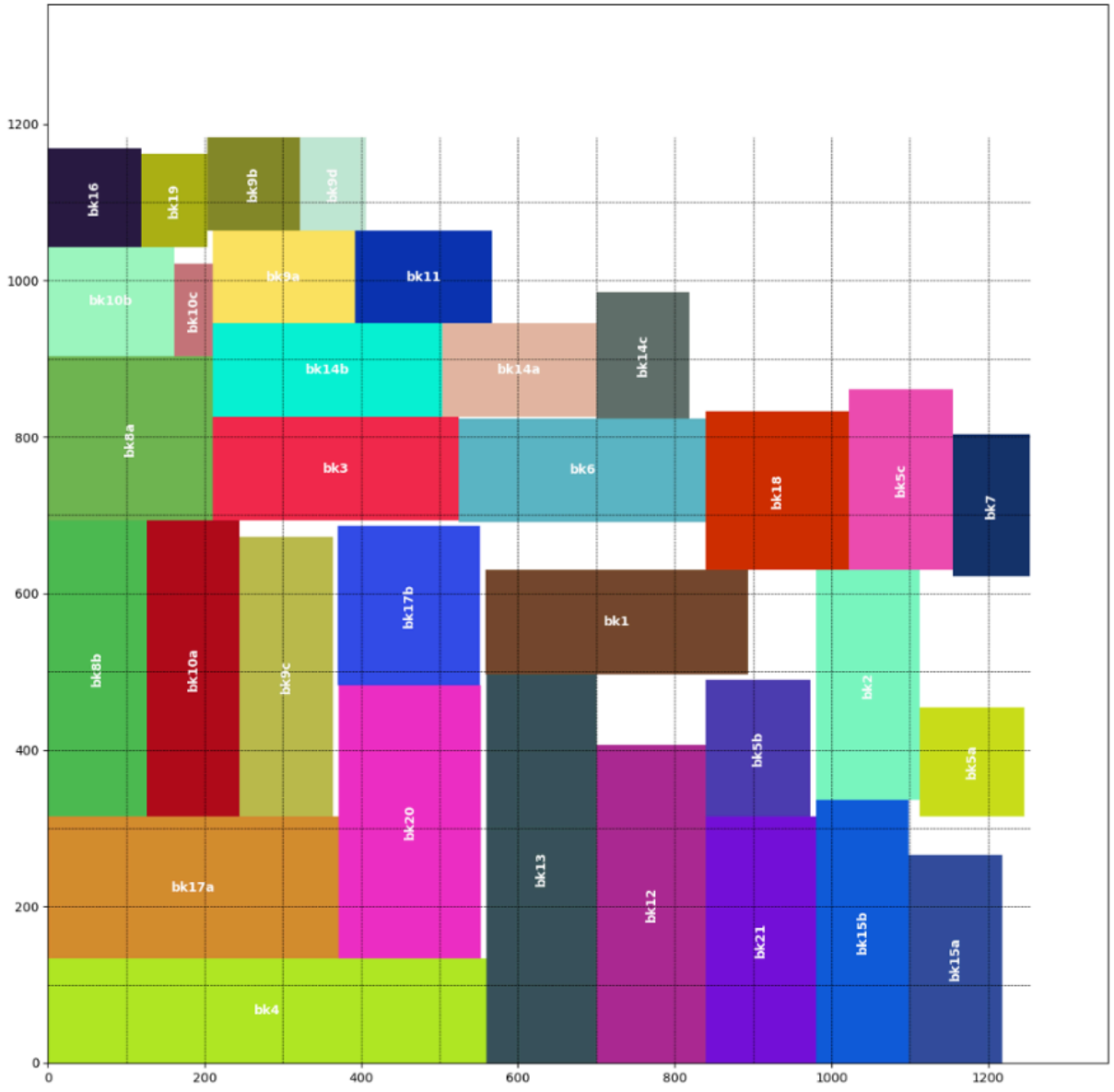
冷却计划：
使用 Temperature = α * Temperature 更新温度，其中 α 为 0 到 1 之间的冷却率。

实验结果

•总 Cost 中 α 的取值为 0.5
•布图优化迭代次数 $iteration$ 的取值为 100000
•模拟退火中 temperature 取 1000

Name	Cost	Wirelength	Area	Width	Height	RunTime(s)
ami33	12.13	148237	1482299	1253	1183	0.402
ami49	24.13	1869070	40740560	5320	7658	12.284
test	0.85	140	10000	100	100	0.0004
xerox	40.77	1115624	24416700	6342	3850	0.041

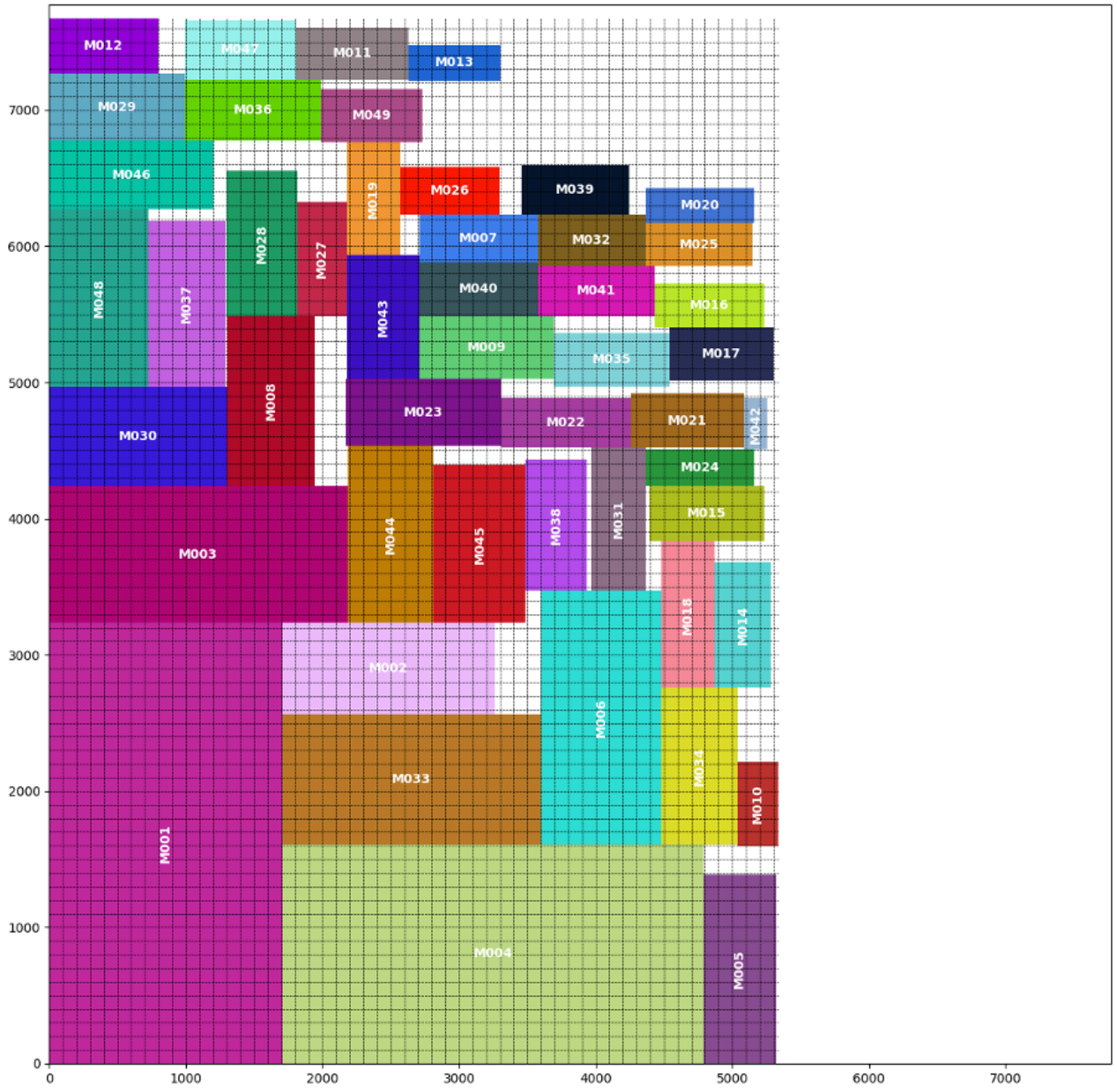
ami33


bk16
bk19
bk9b
bk9d
bk10b
bk10c
bk9a
bk11
bk14b
bk14a
bk14c
bk8a
bk3
bk6
bk18
bk5c
bk7
bk8b
bk10a
bk9c
bk17b
bk1
bk2
bk5a
bk5b
bk17a
bk20
bk13
bk12
bk21
bk15b
bk15a
bk4
0
200
400
600
800
1000
1200

src > output > floorplan_2024-12-26-20:26:41.output

```
Cost 12.132124170930872
Wirelength 148237
Area 1482299
Width 1253
Height 1183
RunTime 0.40212035179138184
bk4 0 0 560 133
bk13 560 0 700 497
bk17a 0 133 371 315
bk20 371 133 553 483
bk12 700 0 840 406
bk8b 0 315 126 693
bk10a 126 315 245 693
bk1 558 497 894 630
bk21 840 0 980 315
bk8a 0 693 210 903
bk9c 245 315 364 672
bk3 210 693 525 826
bk6 525 691 840 824
bk15b 980 0 1099 336
bk2 980 336 1113 630
bk17b 370 483 552 686
```

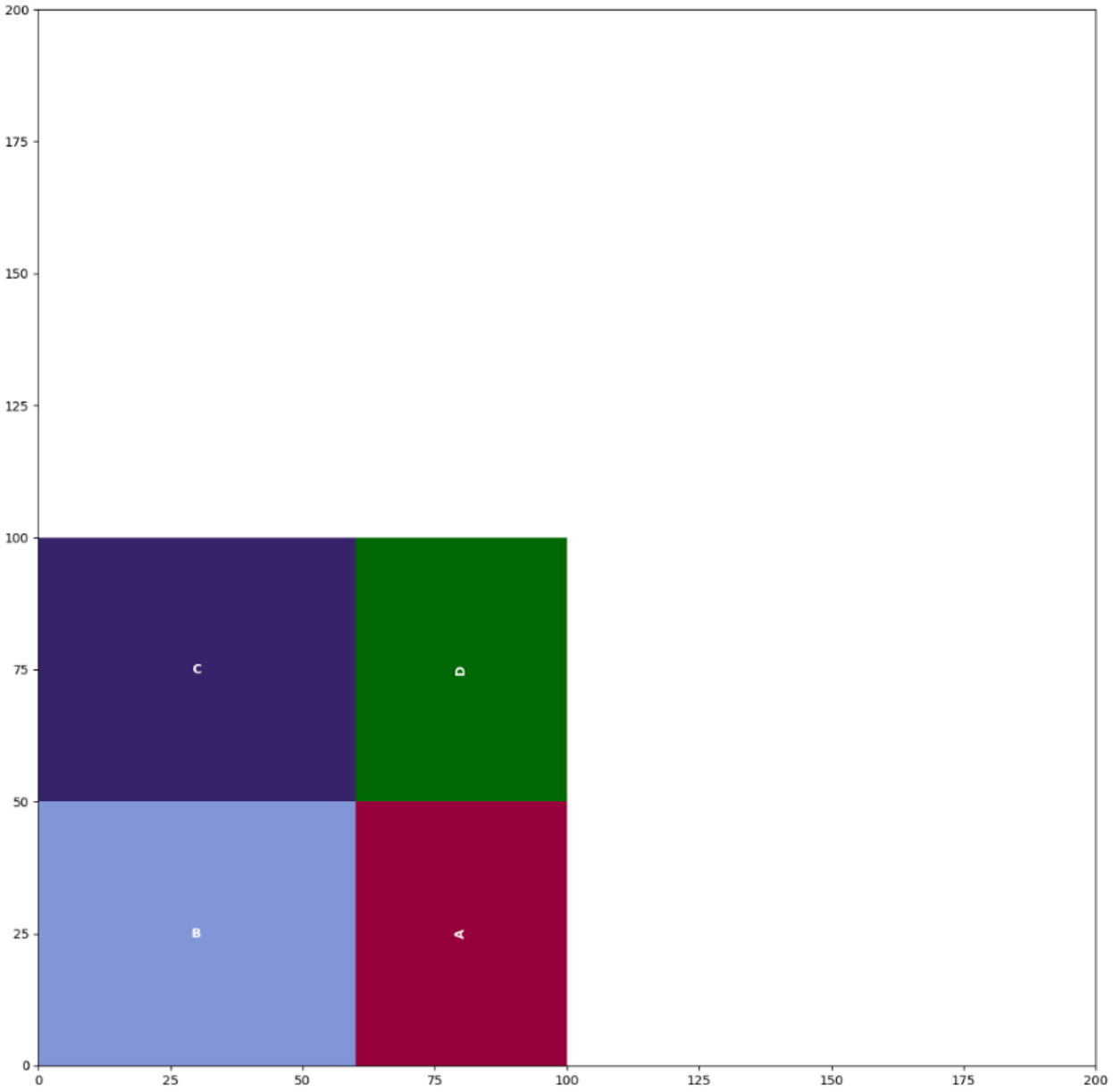
ami49


M012
M047
M011
M013
M029
M036
M049
M046
M019
M026
M039
M020
M028
M027
M007
M032
M025
M048
M037
M043
M040
M041
M016
M009
M035
M017
M008
M023
M022
M021
M042
M030
M024
M003
M044
M045
M038
M031
M015
M018
M014
M002
M006
M033
M034
M010
M001
M004
M005
0
1000
2000
3000
4000
5000
6000
7000

src > output > floorplan_2024-12-26-20:24:47.output

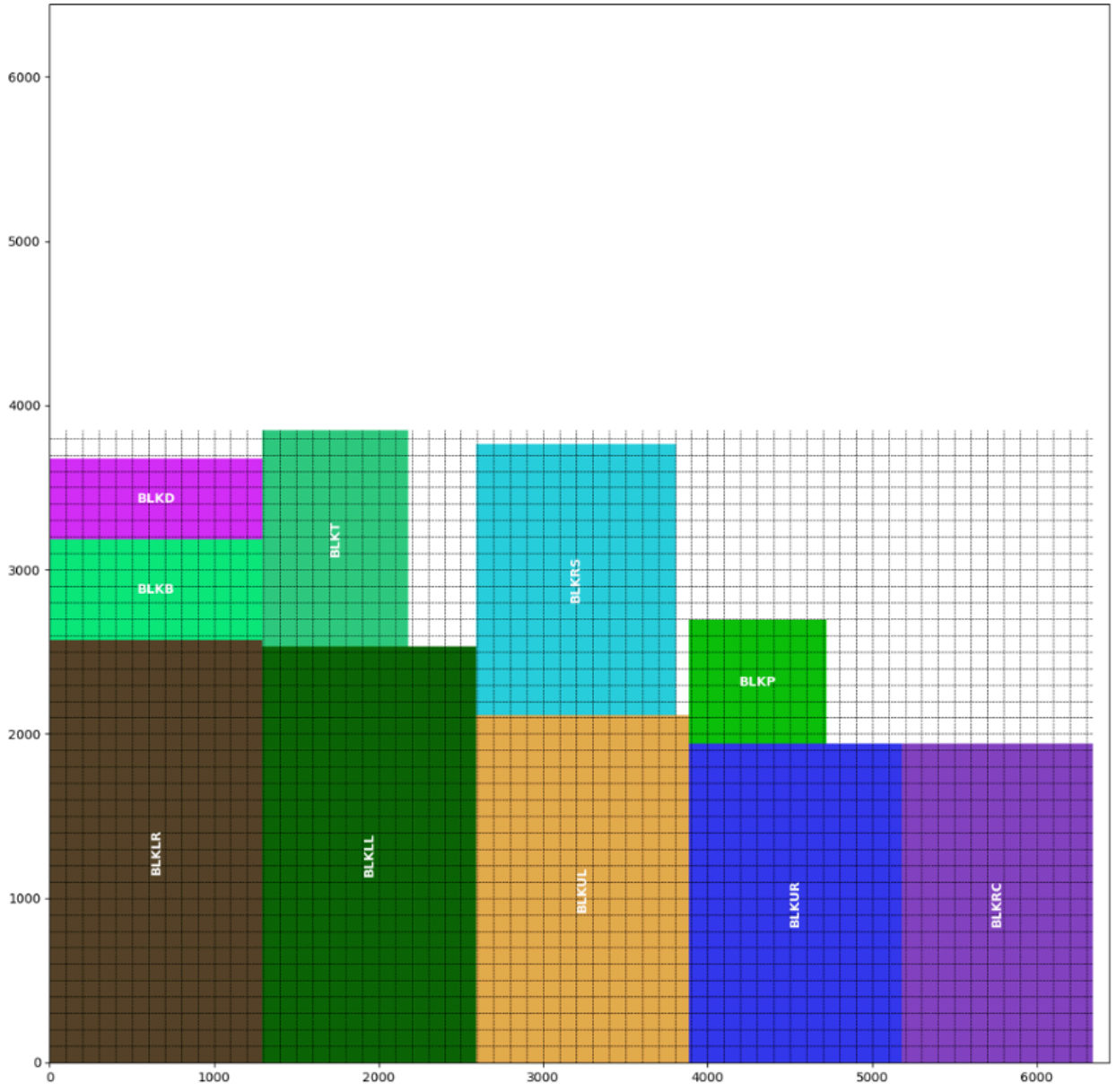
```
Cost 24.133014310403393
Wirelength 1869070
Area 40740560
Width 5320
Height 7658
RunTime 12.283810377120972
M001 0 0 1708 3234
M004 1708 0 4788 1610
M003 1708 1610 3892 2618
M033 0 3234 1890 4186
M006 3892 1610 4774 3472
M002 1890 3178 2562 4732
M030 0 4186 1302 4914
M048 0 4914 1302 5642
M044 2562 3178 3192 4480
M008 3192 3150 3836 4396
M045 2562 4480 3234 5642
M005 4788 0 5320 1386
M037 1708 2618 2926 3178
M034 1302 4186 1862 5334
M046 0 5642 504 6846
M023 4788 1386 5278 2520
M028 504 5642 1022 6706
M029 1302 5334 1792 6328
M043 2926 2618 3836 3150
M009 3836 3472 4816 3934
M036 1862 4732 2310 5726
M038 0 6846 966 7294
```

test



```
Cost 0.85
Wirelength 140
Area 10000
Width 100
Height 100
RunTime 0.0004827976226806406
B 0 0 60 50
C 0 50 60 100
A 60 0 100 50
D 60 50 100 100
```

xerox


6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
BLKD
BLKB
BLKLR
BLKT
BLKLL
BLKFS
BLKJL
BLKP
BLKUR
BLKRC

src > output > floorplan_2024-12-26-20:30:32.output

```
Cost 40.76704752508682
Wirelength 1115624
Area 24416700
Width 6342
Height 3850
RunTime 0.040709733963012695
BLKLR 0 0 1295 2569
BLKLL 1295 0 2590 2534
BLKUL 2590 0 3885 2114
BLKUR 3885 0 5180 1939
BLKRC 5180 0 6342 1939
BLKRS 2590 2114 3808 3766
BLKT 1295 2534 2177 3850
BLKB 0 2569 1295 3185
BLKP 3885 1939 4725 2695
BLKD 0 3185 1295 3675
```

总结

收获

- 熟悉了 Floorplan 的任务目标和基础流程。
- 学会了使用朴素数据结构和优化算法处理 Floorplan 任务
- 可视化 Floorplan 有助于调试并验证算法的正确性
- 积累了自己的编码经验

参考

[1] S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning through Better Local Search," International Conference on Computer Design (ICCD), 2001.
[2] N. Sherwani, Algorithms for VLSI Physical Design Automation, Springer, 2002.
[3] 丛京生，萨拉费扎德，"芯片布局设计与优化"，IEEE Design & Test of Computers，第14卷，第2期，页码12–25，1997年。

README

如何运行

```shell
cd ./src
# 修改config.json中 参数和input file位置
python main.py
```

输入解析

块文件解析（`parse_dotblock`）

——解析`.block文件`

- **目的**：读取块的尺寸、块和终端的数量，以及外形轮廓的大小。
- **过程**：
  - 提取外形轮廓尺寸并初始化 `Outline` 对象。
  - 解析块信息，创建具有相应宽度和高度的 `Block` 对象。
  - 解析终端信息，创建具有固定位置的 `Terminal` 对象。

网络列表文件解析（`parse_dotnet`）

——解析`.net文件`

- **目的**：读取块和终端之间的网络连接。
- **过程**：
  - 解析网络，每个网络包含连接的块和终端列表。
  - 创建 `Net` 对象以表示 Floorplan 中的连通性。

关键方法

- `initialize`：
  - 构建初始 B*-树 Floorplan。
  - 在块之间分配父节点和子节点关系。
- `pack`：
  - 根据 B*-树计算每个块的位置。
  - 确保初始配置中块之间没有重叠。
- `perturb`：
  - 随机选择并应用一种扰动操作（旋转、交换、移动）。
  - 修改 B*-树以探索解决方案空间。
- `calculate_cost`：
  - 计算块占用的总面积。
  - 根据网络计算线长。
  - 返回用于 SA 优化的综合成本。
- `simulate_annealing`：
  - 控制 SA 优化循环。

    - 调整温度并应用接受准则。
    - 记录找到的最佳解决方案。
- `check_out_of_outline`：
    - 验证所有块是否在芯片的外形轮廓内。
    - 如果布局有效，返回 `True`。
- `check_overlap`：
    - 检查块之间是否存在重叠。
    - 确保布局可行。
- `revert`：
    - 如果当前扰动未被接受，则恢复到先前的 Floorplan 状态。
    - 维护 B*-树的完整性。

`visualize`

- **功能**：
    - 从输出文件读取最终的 Floorplan。
    - 将每个块绘制为具有相应尺寸和位置的矩形。
    - 为块分配随机颜色以便区分。
- **目的**：
    - 提供 Floorplan 的可视化表示。
    - 有助于验证布局的正确性。
- **输出**：
    - 将生成的图像保存为文件（如 `floorplan_output.png`）。