# Simulated Annealing for Unit-Size Placement

2019011112 无95 郑名宸

## 任务

本次的任务是使用退火算法来进行placement

## 数据输入格式

The problem definition is taken from a file that obeys the following syntax:

- The first line gives the length *L* of the area available for placement.
- The second line gives the area's width *W*.
- The third line gives the number of cells *m* (*m* is less than or equal to *LW*).
- The fourth line gives the number of nets *n*.
- The fifth line is empty.
- The next *n* lines describe the netlist: the first line lists all the cells that are connected to Net 1, the second the cels connected to Net 2, etc.

## 数据结构

首先输入的是placement的大小和net和cell数量

摆放场地使用2维矩阵，大小为L*W

net使用二维动态数组list

## 算法流程

### 1 初始化

初始化采用随机摆放unit

```
def initialize(L, W, M,grid):
    for element in range(1,M+1):
        assigned = False
        while(not assigned):
            i = np.random.randint(L)
            j = np.random.randint(W)
            if(grid[i][j]==0):
                grid[i][j] = element
                assigned = True
    return grid
```

## 2 衡量cost

接着是要衡量cost，这边的cost采用semi-perimeter len来计算，公式为

cost = 包围net的最大方框的W+L

代码为：

```
def Cost(pos, connections,size):
    cost = 0
    max_x = 0
    max_y = 0
    min_x = size
    min_y = size
    for i in connections:
        for j in i:
            #print(j)
            [x,y] = pos[j-1]
            if(x>max_x):
                max_x = x
            if(x<min_x):
                min_x = x
            if(y>max_y):
                max_y = y
            if(y<min_y):
                min_y = y
        cost = cost + max_x +max_y - min_x - min_y

    return cost
```

## 3 邻居状态

接着是要实现求出邻居状态，也就是要怎么新增一个扰动。

我采用50%的几率让任意两个单元互换

50%的几率让一个单元随机动

```
def NeighborState (pos,grid,M):
    # We will follow these two protocols, each 50% of time
    # 1. 50%的几率让任意两个单元互换
    # 2. 50%的几率让一个单元随机动
    tempGrid = copy.deepcopy(grid)
    temppos = copy.deepcopy(pos)
    if(np.random.random()<0.5):
        element1 = np.random.randint(0, M)
        element2 = np.random.randint(0, M)
                # to ensure element2 != element1
        while(element2==element1):
            element2 = np.random.randint(0, M)
        [x1,y1] = temppos[element1]
        [x2,y2] = temppos[element2]
        tempGrid[x1][y1] = element2+1
        tempGrid[x2][y2] = element1+1
        temppos[element1] = [x2,y2]
        temppos[element2] = [x1,y1]
    else:
```

```
        element1 = np.random.randint(0, M)
        [x1,y1] = temppos[element1]
        # find an empty place
        empty = np.where(tempGrid == 0)
        num = empty[0].size
        ran = np.random.randint(0, num)
        x2 = empty[0][ran]
        y2 = empty[1][ran]
        tempGrid[x1][y1] = 0
        tempGrid[x2][y2] = element1 +1
        temppos[element1] = [x2,y2]
    return temppos,tempGrid
```

## 4 退火算法

核心为如果温度大了话，可以接收delta为正的情况（爬坡），随着温度的下降，接收的程度越来越小，最后稳定下来

```
while(T>threshold):
        temppos,tempgrid = NeighborState(pos,grid,M)
        tempcost = Cost(temppos, connections,size)
        delta = tempcost - cost
        if(delta < 0):
            grid = tempgrid
            pos = temppos
            cost = tempcost
        else:
            p = np.exp(-delta / T)
            if(np.random.random()<p):
                grid = tempgrid
                pos = temppos
                cost = tempcost
        if(tempcost<mincost):
            mincost = tempcost
            mingrid = tempgrid
        T = T*alpha
        # print(cost)
        storedcost.append(cost)
```

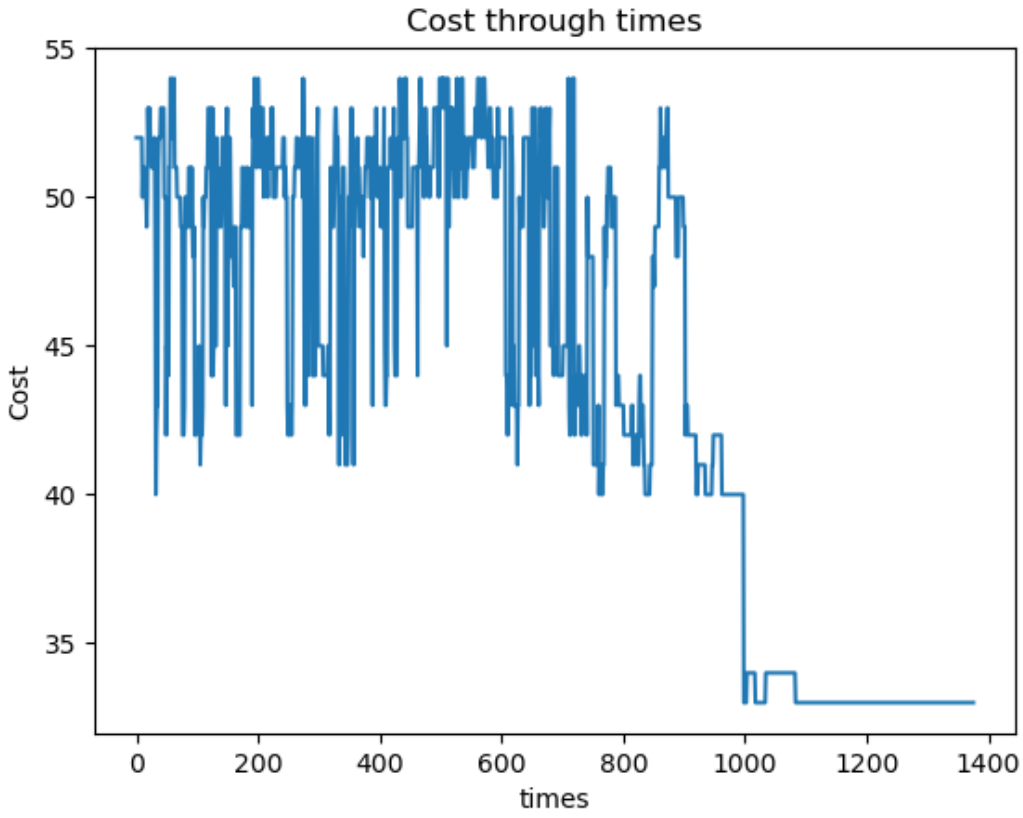
## 4 结果

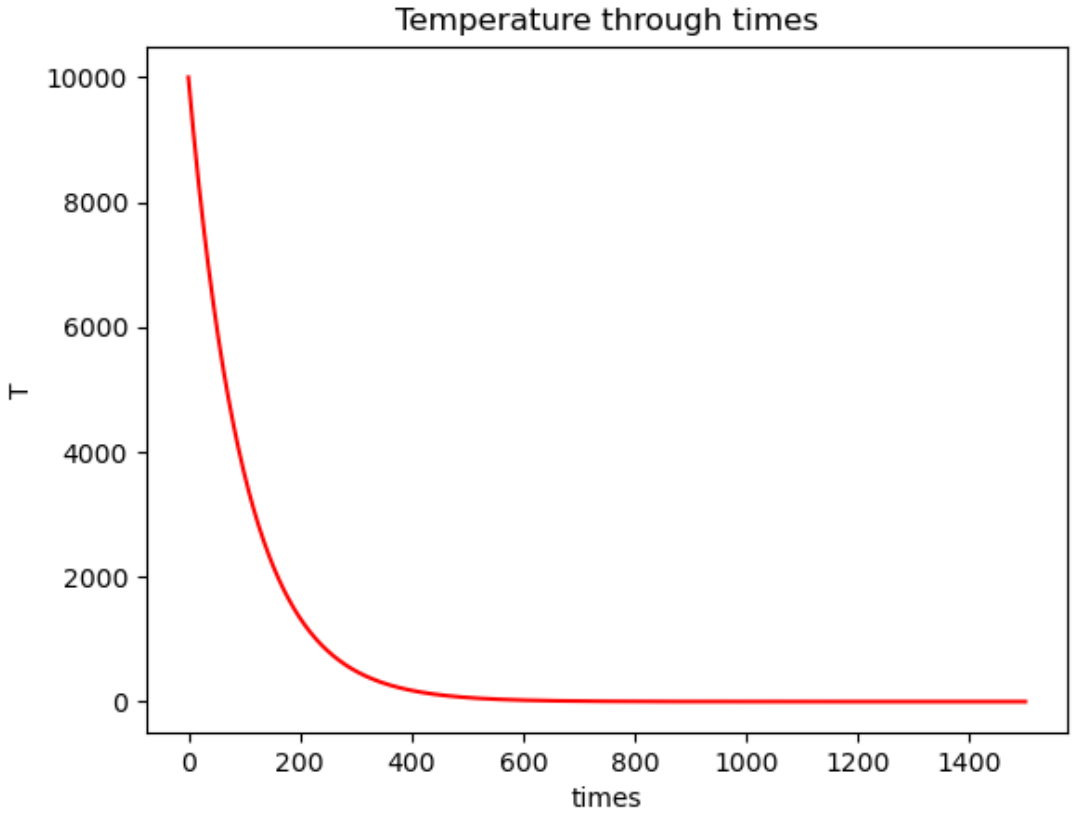
测试的数据为附件的data.txt。由下面的数据和图可以看到，最后收敛的final cost和找到的min cost 是相等的，成功收敛了。温度大致在800次左右收敛。可以画出阈值P和温度T，P和T的关系为

$p = e^{-\Delta/T}$

```
initial grid
 [[5. 0. 0. 0. 8.]
 [0. 0. 0. 4. 7.]
 [1. 6. 2. 3. 0.]]
initial cost 52
final grid
 [[0. 6. 3. 4. 0.]
 [0. 0. 1. 5. 0.]
 [0. 7. 8. 2. 0.]]
final cost 33
```

```
mingrid
 [[0. 1. 5. 0. 0.]
 [0. 4. 3. 6. 0.]
 [0. 2. 7. 8. 0.]]
mincost 33
```



# 总结

经过本次的退火算法实验，完整的了解退火算法的流程。经过了一次的代码实现，比在课上单纯的了解算法还要更深刻的体会了这个算法精髓。因为时间有限，所以没有使用很多的数据去测试我实现的代码，但整体来说得到的结果和鲁棒性是很好的。