多模式集成电路通道布线模型与算法研究

摘　　要

本文分别考虑单层金属层、多层金属层、添加通孔约束的多层金属层三种情形下对“通道布线”问题进行优化。

针对问题一，考虑单层金属层的布线优化，布线区域为二维平面，首先建立二维坐标系以表示布线区域中的方格位置，建立相关集合以表示各方格与其他方格间是否可直接进行布线的关系，以保证金属线只能沿着直线或直角放置。本问题旨在最小化总布线长度，因此将该问题与传统最短路问题的异同点进行了分析，并对一般最短路模型进行改进，主要从寻找多个引脚对的最短路（总路径最短）和保证每两个引脚对线路间不相交为切入点建立相关约束，通过分别对上引脚、下引脚、非上下引脚处的方格进出流量关系与流量大小进行约束，构建了基于改进最短路的单层二维布线模型。该模型为混合整数规划模型（MIP），因此通过 Python 调用求解器 Gurobi 进行求解，并对三个测例的结果进行了展示与分析，发现只有测例 1 有解。考虑到这是一种基于线搜索的并行方法，本题还从基于网格搜索的串行方法展开，采用基于改进 A*的多层布线算法对问题进行求解，它采用了阶乘的思想确定引脚对搜寻顺序，发现了和上述方法相同的结论。最后还分析了无解的情况，发现如果引脚对间存在同侧合法边界相交，则两对引脚不能在同一层中连接，此时一层金属通道布线问题无解。

针对问题二，考虑了多层金属层的布线优化，以实现不同金属线可共用一个方格而不会引起短路。首先在问题一建立的模型基础上，首先考虑将各金属层均映射到最底层，通过交叉点的数量与通道的关系建立二维布线模型，但存在一些该模型无法考虑到的情况，因此考虑三维空间，增加金属层维度，考虑各项约束，分别针对下引脚必须从底层连出与可从任意层连出两种假设下构建了两个基于改进最短路的多层三维布线模型，目标函数中考虑通孔的电阻，对布线方案进行优化。依然调用 Gurobi 对三个测例进行求解，并对结果进行了分析，发现在多层布线时，这三个算例都有解，并且模型运算速度较快。另外，还采用了基于改进遗传算法的多层布线算法对问题进行了求解。它是对问题一提出的改进 A*算法的进一步提升，利用了改进 A*算法生成种群并获取适应度，极大的提高了算法的运行效率，最终发现了与上述方法相同的结论。

针对问题三，在问题二下引脚均从底层连出的假设的基础上增加了任意两个通孔的间距必须大于等于 2 个格点的约束，首先对通孔间距进行定义，考虑获取通孔所处的位置，发现所有通孔的位置均能通过中间金属层进行体现，以此为切入点，探讨该金属层上与有通孔方格的距离小于 2 的方格无通孔的约束，构建模型进行求解，发现该约束对共 z 轴通孔无约束，此时，三个测例都有可行解，当针对该情形通过探讨变量间的关系增加了相关约束，结果表明只有测例 1 有解，其他测例无解，也说明了本文相关假设稍有严格。

关键词：最短路；MIP；Gurobi；改进 A*；改进 GA

1 问题重述

集成电路是利用半导体技术把电子元件集成在一起的具有特定功能的电路，已广泛应用于生产生活的方方面面。随着技术的发展，集成电路内部的元器件数目已达到十亿级别，需要借助专用计算机软件才能完成电路设计与实现，该类软件统称为电子设计自动化（Electronic Design Automation, EDA）工具。

集成电路设计由多个阶段组成，其中一个重要阶段称为“物理设计”，先将器件摆放在合适的位置，然后用金属线连接器件实现连接关系。其中，后者称为“布线”，它是 EDA 工具需要解决的重要问题。简单而言，假设可用区域由 n*m 个方格组成，金属线允许沿着直线或直角（方格）放置，连接指定的方格（引脚）而不引起断路或短路，该过程称为“布线”。由于金属线引入的寄生电阻会影响电路性能，所以需要最小化布线长度。本题重点考虑“布线”问题中的一个特例：“通道布线”。“通道”是指一个横向的布线区域，此区域的顶部和底部分布着需要连接的方格，需用金属线将相应的引脚连通起来。

问题一：假设采用一层金属布线，那么已经布线的方格被锁定，不允许其它线路穿过，否则会形成短路。问题一所示为采用一层金属的通道布线例子，布线空间为 4*7，空间上下沿的数字分别对应方格的引脚编号，编号相同的引脚需要连接起来。请针对此一层金属的“通道布线”问题完成建模和求解，并回答如下问题：在何种情况下，一层金属通道布线问题无解。

问题二：可以观察得到，有些测例无法采用一层金属完成布线。实际中，集成电路会采用多个金属层，不同的金属层处在不同的高度，相邻层之间需要用通孔连通，这样不同金属层可共用一个方格而不引起短路。问题二所示为芯片的剖面图，其中网状填充为金属层，点状填充为通孔。问题二所示为一个用两层金属的布线示例，其中蓝色为下层金属，黄色为上层金属，红色为通孔。假设一个通孔的电阻等于 5 个方格的导线，请使用最多 3 层金属对“通道布线”重新建模和求解。

问题三：随着集成电路尺寸缩小，新的通孔制造工艺要求任意两个通孔的间距必须大于等于 2 个格点，请加入此通孔相关的约束后再次求解问题。

2 问题分析

2.1 理论背景

本题重点考虑“通道布线”，它是指一个横向的布线区域，此区域的顶部和底部分布着需要连接的方格，需用金属线将相应的引脚连通起来。通道布线运用的数据结构主要有三类：基于网格的数据结构，基于线的数据结构，基于 Tile 的数据结构：（1）基于网格搜索的算法使用网格点作为布线资源，将占用大量的内存来存放布局，其空间复杂度为长×宽。该算法可能因为搜索空间太大而导致很高的时间复杂度；（2）基于线的搜索使用线段作为布线资源，占用的内存相较使用网格点作为布线资源明显减少。由于路径形状简单性，找到的通路也许不是最短路径，但该算法一般比基于网格点的算法快；（3）基于 Tile 的数据结构以无网格的模式来表示版图，线宽和间距都是可变的，在设计规则上有很强的灵活性。

另外，通道布线采用的基本布线技术有三种。第一种是盲目搜索算法，也就是一个广度优先搜索算法。第二种是 A*算法，与盲目搜索算法不同，A*算法度量从源点到目标点期望路径的长度，并且总是扩展具有最小度量值的网格。第三种是双向搜索算法，双向搜索算法较无方向搜索算法优势是搜索范围更小。

其次，通道布线方法还会考虑串并行布线。串行布线意思是将待布设的线网按某个顺序一条一条地进行布线，一条已布设完的线网可能对下一条待布设线网产生堵塞等影响，因此这类算法对线网的排序相当敏感。而并行布线则不同，它力求克服串行算法中的线网必须排序这一缺点，同时考虑线网集的所有线网布线，并行布线的计算复杂性很高[1,2]。

因此，本研究将根据以上理论背景对各问题进行建模求解。

2.2 问题一分析

根据问题一可知，该问题对象为通道布线中的双边双端线网问题，引脚分布于顶边和底边。要求采用一层金属将编号相同的引脚连接起来，每个方格只能布线一次以避免短路。并回答在何种情况下，一层金属通道布线问题无解。

为了解决该问题，首先需要对题中所给的布线空间建立坐标系，以便于对布线空间中的方格与方格间的路线进行表示，同时建立相关集合以表示各方格与其他方格间的关系，例如是否可以直接用金属线连接连通等，以保证金属线只能沿着水平与垂直方向布线。考虑问题旨在最小化总布线长度，与最短路问题类似，探讨该问题与最短路问题的相似之处与差异点，构建模型，即基于线搜索的并行方法。由此考虑到从基于网格搜索的串行方法，因此还采用了基于改进 A*算法的单层布线算法对问题进行求解。并在模型求解过程中发现了无解时的引脚对的位置关系的规律。

2.3 问题二分析

问题二引入了多层金属层，其间有通孔连。在问题一建立的模型基础上，考虑到多个金属层可以实现不同金属线可共用一个方格而不会引起短路，因此首先考虑将各金属层均映射到最底层，通过交叉点的数量与通道的关系建立二维布线模型，但在模型构建与求解

过程中发现一些问题，因此考虑三维空间，增加金属层维度，考虑各项约束，构建了三维布线模型，目标函数中考虑通孔的电阻，对布线方案进行优化。同时也考虑两种不同的角度对问题求解，分别是基于改进最短路的多层三维布线模型（基于线搜索的并行方法）和基于改进遗传算法的多层布线算法（基于网格搜索的串行方法）。

2.4 问题三分析

由问题三可知，要求任意两个通孔的间距必须大于等于 2 个格点，首先对通孔间距进行定义，考虑获取通孔所处的位置，发现所有通孔的位置均能通过中间金属层进行体现，以此为切入点，探讨该金属层上与有通孔的方格距离小于 2 的方格无通孔的约束，以及各种通孔位置关系下的间距的约束，构建了考虑通孔约束的多层三维布线模型并求解。

3 模型假设

（1）假设每个引脚都在网格中心点上。

（2）假设当有多个金属层时，上引脚均从底层接入。

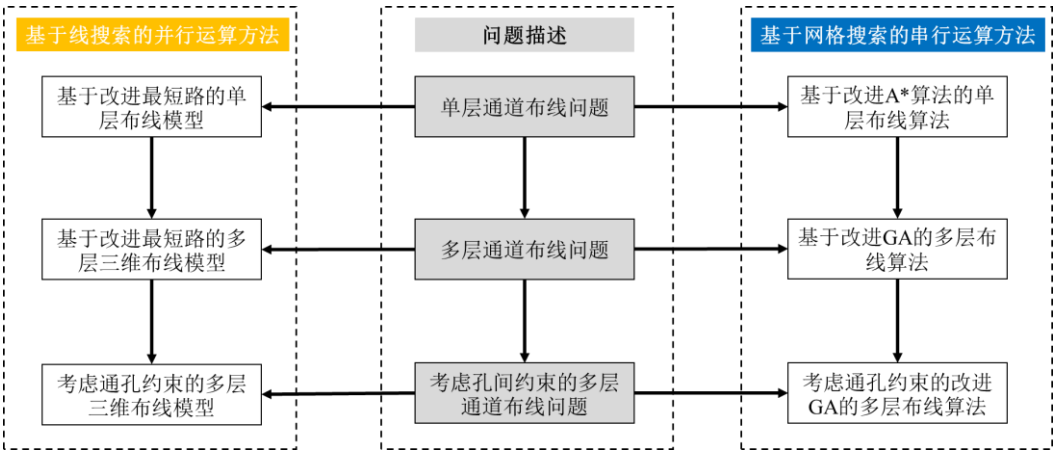
（3）假设金属层厚度足够小，当用三维网络表示方格空间时，任意两通孔上的点间的最短距离表示为这两个通孔间的间距。

4 符号说明

符号	符号说明
(i, j)	二维网络中方格的坐标，其集合为 N
(i, j, k)	三维网络中方格的坐标，其集合为 N'
k	金属层数，值为 0、1、2，其集合为 L
r	引脚对的上引脚坐标，其集合为 R，三维中分别对应为 r'、R'
s	引脚对的下引脚坐标，其集合为 S，三维中分别对应为 r'、R'
rs	引脚对，其集合为 RS，三维中分别对应为 $r's'$、$R'S'$
$x_{ij,kl}$	0-1 变量，二维网络中从网格 (i, j) 至 (k,l) 是否被金属线覆盖
$x_{ijk,lmk}^{r's'}$	0-1 变量，二维网络中从网格 (i, j,k) 至 (l,m,n) 是否被金属线覆盖
N_{ijk}	与网格 (i, j,k) 能直接相连的网格的集合
z_{ij}	引脚对从中间层上的点 $(i, j,1)$ 流出的流量总和
NE_{ij}	单层平面上与方格 (i, j) 相邻（包含对角）的方格的集合

5 建模思路

针对上述三个问题所关注的重点，本研究将它们归纳为单层通道布线问题，多层通道布线问题以及考虑孔间约束的多层通道布线问题。经过对以往研究的梳理了解到多种类型的通道布线方法，本研究分别从两个角度出发，分别是基于先搜索的并行方法和基于网格搜索的串行方法，来建立规划模型和优化算法。随着研究问题的推进，逐步提升模型或算法的适用性，以达到期望要求，具体思路如下图所示。



6 问题一的模型建立与求解

6.1 基于改进最短路的单层布线模型建立与求解

6.1.1 模型建立

为方便模型的建立，首先将布线的方格转化成所需的形式，取布线空间中的每个方格的中心代表该方格，以第一行第一列（左下角）的方格中心为原点(0, 0)，以向右为 x 轴正方向，向上为 y 轴正方向建立布线方格坐标系，如下图所示，得到每个方格中心的坐标以表示该方格，例如第一行第二列的方格的坐标为(0, 1)。

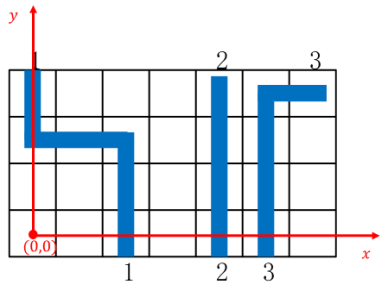


图 1：二维布线方格坐标系

1) 单条路径最短路模型

对应单条路径最短路问题，只有一个起点一个终点，即单对 OD，其模型表达式及示例图如下：

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$subject\ to \sum_{\{j:(i,j)\in A\}} c_{ij}x_{ij} - \sum_{\{j:(i,j)\in A\}} c_{ij}x_{ji} = \begin{cases} 1, i = r \\ -1, i = s \\ 0, \forall\, i \in N \setminus \{r、\ s\} \end{cases}$$

$$x_{ij} \geq 0, \forall (i,j) \in A$$

$$where \quad c_{ij} = cost\ on\ link\ (i,j)$$

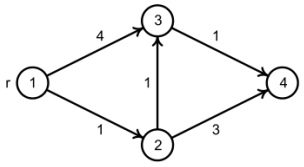


图 2：一般最短路问题网络图示例

上述模型寻找从起点 r 至终点 j 的最短路径，该路径上的流量为一个单位网络流，模型分别对起点、中间点、终点进出的流量进行限制。对于起点，应当有 1 个单位的网络流流出，无网络流流进，因此进入该点的流量减去进入该点的流量应当等于 1 个单位的网络流。对于终点，应当无流量流进，1 个单位的流量的流出，因此进入该点的流量减去进入该点的流量应当等于-1 个单位的网络流。对于非起点与终点的其他中间点，进入该点的流量应当等于流出该点的容量，因此两者之差等于 0。

2) 单层通道布线的改进最短路模型

可将本文的问题描述为，对于一个引脚对，从上引脚（起点）开始沿着相邻方格寻找最短路到达下引脚（终点），与最短路问题具有相似性，因此对上述模型进行改进使之适用于本文需要解决的单层通道布线优化问题。首先将本题的方格映射到寻找最短路的网络中，布线空间中每个方格的中心坐标表示最短路网络中的节点，相邻方格中心的连线（仅包含

相邻网格可保证金属线沿着方格放置不会出现断路）代表最短路网络中的链，即连接可相通的节点的弧，用 $x_{ij,kl}$。由于模型中存在流进与流出方格的流量的概念，所建立的网络为有向网络，相邻方格中心的连线具有方向性，例如 $x_{ij,kl}$ 表示从坐标为 (i,j) 的方格连接至坐标为 (k,l) 的方格的金属线。

与前文单条路径最短路问题不同的是，单层通道布线问题不仅是寻找一个引脚对的最短路问题，在本问题中，每一个引脚对相当于最短路问题中的一个 OD 对，因此单层通道布线问题为同时寻找多个 OD 对的路径问题，同时每两个 OD 路径间无交叉。可将整个问题描述为在前文所建立的二维布线方格网络中，寻找多个引脚对间的路径，在保证引脚对间无交叉的情况下，最小化布线长度。因此，对前文所述的最短路模型进行改进，改进的思路主要包含两个要点：（1）寻找多个引脚对的最短路（总路径最短）；（2）保证每两个引脚对线路间无交叉。基于金属线展开，以相邻方格的连线是否布设金属线为决策变量，以最小化总布线长度为优化目标建立基于改进最短路的 MIP 模型对单层布线问题进行求解。

假设对每个 OD 对 rs，即每个引脚对分配的网络流为 1 个单位流量，模型构建思路如下（相关符号含义见第四章符号说明）：

（1）变量的 0-1 约束

模型中的所有变量 $x_{ij,kl}^{rs}$ 均为 0-1 变量，其中 rs 表示上引脚为 r，下引脚为 s 的引脚对，(i,j) 为网络中任一方格中心的坐标，(k,l) 表示与 (i,j) 相邻的方格中心坐标，$x_{ij,kl}^{rs}$ 表示从方格 (i,j) 至方格 (k,l) 是否对引脚对 rs 进行布线，为 0-1 变量，1 表示布线，0 表示不布线。

（2）对上引脚 r 处流出流量的约束

对任一对引脚对 rs（$rs \in W$）而言，在上引脚 r 处，流出的流量总和应当为 1：

$$\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} = 1 \quad (i,j) = r, \forall\, rs \in RS \tag{1}$$

该约束对任一引脚对 rs 的上引脚 r 进行约束，即当表达式中的 (i,j) 为引脚对 rs 的上引脚 r 时，引脚对 rs 的路径从该点流出的网络流总和为 1。

上述约束仅对引脚对 rs 的上引脚 r 处，对 rs 进行布线的流量进行了约束，为了保证在某一对引脚对 rs 的上引脚 r 处，不会对其他引脚对进行布线，因此在该点处增加如下约束：

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} = 1 \quad (i,j) \in R \tag{2}$$

上述约束表示对任一引脚对 rs 的上引脚 r 处，所有引脚对的路径从该点流出的网络流总和为 1。

（3）对上引脚 r 处流入流量的约束

针对每个引脚对的上引脚而言，无论是对于其所属的引脚对路线还是其他的引脚对路线，都不应当有流量流入，因此只需要增加一个对所有引脚对的总约束即可：

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = 0 \quad (i,j) \in R \tag{3}$$

上述约束表示对任一引脚对 rs 的上引脚 r 处，所有引脚对的路径流入该点的网络流总和为 0，即所有上引脚处均无网络流输入。

（4）对下引脚 s 处流入流量的约束

对任一对引脚对 rs ($rs \in W$)而言，在下引脚 r 处，流入的流量总和应当为 1：

$$\sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = 1 \quad (i,j) = s, \forall\, rs \in RS \tag{4}$$

该约束对任一引脚对 rs 的下引脚 s 进行约束，即当表达式中的 (i, j) 为引脚对 rs 的下引脚 s 时，引脚对 rs 的路线流入该点的网络流总和为 1。

上述约束仅对引脚对 rs 的下引脚 s 处，对 rs 进行布线的流量进行了约束，为了保证在某一对引脚对 rs 的下引脚 s 处，不会对其他引脚对进行布线，因此与上引脚约束类似，同样在该点处增加如下约束：

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = 1 \quad (i,j) \in S \tag{5}$$

上述约束表示对任一引脚对 rs 的下引脚 s 处，所有引脚对的路线流入该点的网络流总和为 1。

（5）对下引脚 s 处流出流量的约束

针对每个引脚对的下引脚而言，无论是对于其所属的引脚对路线还是其他的引脚对路线，都不应当有流量流出，因此只需要增加一个对所有引脚对的总约束即可：

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} = 0 \quad (i,j) \in S \tag{6}$$

上述约束表示对任一引脚对 rs 的下引脚 s 处，所有引脚对的路径从该点流出的网络流总和为 0，即所有下引脚处均无网络流输出。

（6）对非上、下引脚处流入与流出流量的约束

针对非上引脚与下引脚的其他节点（方格）而言，对任一对引脚对 rs 路线，其流入的流量应当与流出的流量相当：

$$\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} - \sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = 0 \quad (i,j) \neq r、s, \forall\, rs \in RS \tag{7}$$

该约束为对任一引脚对 rs 布线时的非引脚处的流量均衡约束，即当表达式中的 (i, j) 为非引脚坐标时，引脚对 rs 的路线从该点流入的网络流与流出的网络流的差值为 0。

对于单层通道布线问题，不允许短路也是一个很重要的问题，在前文约束保证布线不出现断路下增加如下约束（对所有方格均成立），保证引脚对路线间不会短路，即不会交叉：

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} \leq 1 \quad \forall (i,j) \in N \tag{8}$$

该约束对流出非引脚方格总流量进行约束，由于前一个约束中对每个引脚对 rs 的路线流入流量等于流出流量，因此所有引脚对的路线流入这些方格的流量一定与流出流量相等，为保证不短路，只需要对流入的总流量与流出的总流量之一进行约束即可。流出非引脚方格的总流量不超过 1 个单位的流量，即对所有引脚对经过该方格的所有路线不会超过 1 条。

（7）优化目标

根据题目相关要求，本文模型的优化目标为最小化总布线长度，由于决策变量 $x_{ij,kl}^{rs}$ 表示从方格 (i,j) 至方格 (k,l) 是否对引脚对 rs 进行布线，1 表示布线，0 表示不布线，因此决策变量之和即为从上引脚方格连接至下引脚方格的总布线长度，表达式如下：

$$\min \sum_{rs\in RS}\sum_{(i,j)\in N}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} \tag{9}$$

上述所有约束从上引脚、下引脚、非引脚方格展开，分别对流入、流出流量进行约束，可保证金属线的不短路与不断路，上述模型总结如下，为混合整数规划模型（MIP）：

$$\min \sum_{rs\in RS}\sum_{(i,j)\in N}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} \tag{10}$$

s.t.

$$\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} = 1 \quad (i,j)=r, \forall\, rs\in RS \tag{11}$$

$$\sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = 1 \quad (i,j)=s, \forall\, rs\in RS \tag{12}$$

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} = \begin{cases} 1 & (i,j)\in R \\ 0 & (i,j)\in S \end{cases} \tag{13}$$

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = \begin{cases} 0 & (i,j)\in R \\ 1 & (i,j)\in S \end{cases} \tag{14}$$

$$\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} - \sum_{(k,l)\in N_{ij}} x_{kl,ij}^{rs} = 0 \quad (i,j)\neq r、s, \forall\, rs\in RS \tag{15}$$

$$\sum_{rs\in RS}\sum_{(k,l)\in N_{ij}} x_{ij,kl}^{rs} \le 1 \quad \forall (i,j)\in N \tag{16}$$

$$x_{ij,kl}^{rs} \in \{0,1\} \tag{17}$$

6.1.2 模型求解

由于集成电路布线问题与传统的最短路问题存在较大差异，因此无法用一般的最短路算法 Dijkstra's algorithm 以及 Bellman's algorithm 这两种算法进行求解上述布线问题，因此本文采用 MIP 数学规划求解方法对上述模型进行求解。

本文利用 python 调用优化器 Gurobi 对 MIP 模型进行求解。Gurobi 是一个求解速度比较快的大规模优化器，通常可以解决具有数百万变量和约束的线性规划（LP）、混合整数规划（MIP）、二次规划（QP）等模型，应用广泛。

在问题一下，Gurobi 求解 MIP 问题采用的算法为单纯形法，由于该算法较为常见，不再赘述。利用本模型对于测例 1、测例 2 和测例 3 分别进行测试，发现测例 2 和测例 3 在问题一的背景下无解，即若只能采用一层金属将编号相同的引脚连接起来且每个方格只能布线一次，测例 2 和测例 3 无解。测例 1 的结果如下图所示，路段长度为 13，耗时 0.02s。

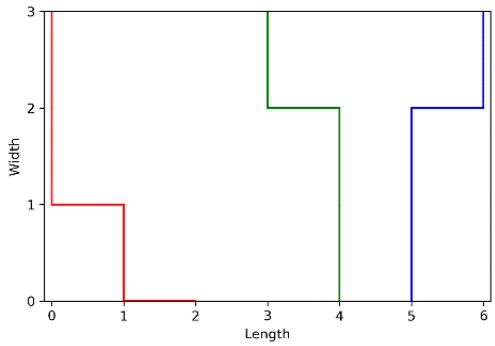


图 3：测例 1 求解结果

6.2 基于改进 A*算法的单层布线算法设计和分析

6.2.1 算法设计

为从算法角度，而非模型角度对问题一求解，本研究利用改进 A*算法对每条路径进行搜索，属于基于网格搜索的串行方法。首先，把搜索区域简化为了 2 维数组。数组的每一项代表一个格子，它的状态就是可走和不可走。通过计算出从 A 到 B 需要走过哪些方格，就找到了路径。一旦路径找到了，线路便从一个方格的中心移动到另一个方格的中心，直至到达目的地。

需要说明的是本研究的代价函数$f(n)$是节点 n 的综合优先级，它由$g(n)$和$h(n)$组成。当我们选择下一个要遍历的节点时，我们总会选取综合优先级最高（值最小）的节点，$g(n)$是节点 n 距离起点的代价，$h(n)$是节点 n 距离终点的预计代价。

如果$h(n)$始终小于等于节点 n 到终点的代价，则 A*算法保证一定能够找到最短路径。但是当$h(n)$的值越小，算法将遍历越多的节点，也就导致算法越慢。如果$h(n)$完全等于节点 n 到终点的代价，则 A*算法将找到最佳路径，并且速度很快。如果$h(n)$的值比节点 n 到终点的代价要大，则算法不能保证找到最短路径，不过此时会很快。在另外一个极端情况下，如果$h(n)$相较于$g(n)$大很多，则此时只有$h(n)$产生效果，这也就变成了最佳优先搜索。考虑到本问题只允许朝上下左右四个方向移动，因此$\hbar(n)$使用曼哈顿距离表示。

$$\hbar(n) = |pos\_n.getX() - pos\_end.getX()| + |pos\_n.getY() - pos\_end.getY()| \quad (18)$$

其中，$pos_n.getX()$为 n 点横坐标，$pos_n.getY()$为 n 点纵坐标，$pos_end.getX()$为终点横坐标，$pos_end.getY()$为终点纵坐标。

算法在运算过程中，每次从优先队列中选取 f(n)值最小（优先级最高）的节点作为下一个待遍历的节点。另外，算法使用两个集合来表示待遍历的节点（$open\_set$），与已经遍历过的节点（$close\_set$）。

由于该问题需要搜寻出多条路径以满足全局的线路规划，在把待布设的路径按某个顺序一条一条地进行布线时，一条已布设完的路径可能对下一条待布设路径产生堵塞等影响，因此该算法对于路径的排序相当敏感。为了解决这一问题，本研究计算了引脚对数的阶乘，并全排列所有排列状况，遍历所有排列状况进行布线设计。这样就能够得到较为客观的通道布线方式。该算法的具体流程如下所示。

```
* 计算引脚对数的阶乘，并全排列所有排列状况；
* 遍历所有排列状况进行布线设计：
    * 遍历当前排列状况搜寻每一对引脚对的布线路径：
        * 初始化 open_set 和 close_set；
        * 将起点加入 open_set 中，并设置优先级为 0（优先级最高）；
        * 如果 open_set 不为空，则从 open_set 中选取优先级最高的节点 n：
            * 如果节点 n 是终点，则：
                * 从终点开始逐步追踪 parent 节点，一直达到起点；
                * 返回找到的结果路径，算法结束；
            * 如果节点 n 不是终点，则：
                * 将节点 n 从 open_set 中删除，并加入 close_set 中；
                * 遍历节点 n 所有的邻近节点：
                    * 如果邻近节点 m 在 close_set 中，则：
                        * 跳过，选取下一个邻近节点；
                    * 如果邻近节点 m 也不在 open_set 中，则：
                        * 设置节点 m 的 parent 为节点 n；
                        * 计算节点 m 的优先级；
                        * 将节点 m 加入 open_set 中；
        * 将为当前引脚对规划的路线设为其他引脚对的障碍；
```

6.2.2 算法分析

利用本算法对于测例 1、测例 2 和测例 3 分别进行计算，发现测例 2 和测例 3 在问题一的背景下无解，即若只能采用一层金属将编号相同的引脚连接起来且每个方格只能布线一次，测例 2 和测例 3 无解。测例 1 的结果如下图所示，路段长度为 13，耗时 0.03s。

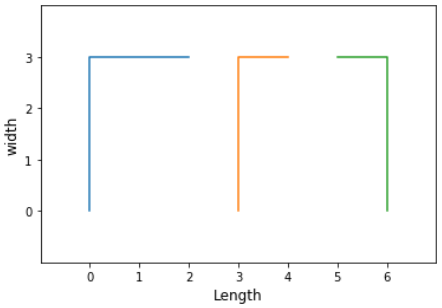


图 4：测例 1 结果示意图

6.3 无解状况分析

由以上两种方法可以总结得出，一层金属通道布线问题无解的状况为：对每一对引脚由左到右尽可能的靠左连接，接着对每一对引脚从右到左尽可能的靠右进行连接。对每一对引脚，得到两条极端路径，称为该引脚对的左合法边界与右合法边界。因此容易看出，如果某个引脚对的右合法边界与另一个引脚对的右合法边界相交或某个引脚对的左合法边界与另一个引脚对的左合法边界相交，则两对引脚不能在同一层中连接，此时一层金属通道布线问题无解[3]。

7 问题二的模型建立与求解

7.1 基于改进最短路的多层三维布线模型建立与求解

本题首先基于第一问中模型，考虑依然基于二维的布线方格，考虑更多的约束建立模型，即对平面中的交叉点数量（用流出该点的总流量和或流入该点的总流量和表示）不超过总金属层数 3 进行限制，在目标函数中用通过交叉点的金属线的数量减 1 表示通道数，用该方法建立了模型并进行了求解（相关代码见附录 3.1）。而后发现在该模型中通道数量与通过交叉点的金属线数量并不存在此关系。因此下文考虑增加金属层维度，建立基于三维网络的布线优化模型。

由于不同的金属层处在不同的高度，因此布线空间可以看作方格组成的三维空间，因此考虑通过三维坐标重新对问题进行建模，金属线间通过不同金属层实现不短路，即可以看作在三维空间中不同金属线不会相交，否则出现短路情况。从三维的角度本题考虑了两种情况：（1）所有的金属线均需要从底层下引脚处连出；（2）金属线可以从任意层的下引脚处连出。下面针对这两种情况分别建模。

7.1.1 基于仅允许底层下引脚处连出假设的模型

1) 模型建立

本题在第一题的模型基础上进行建模，与第一题不同的是，第一题中的方格（节点）由二维坐标表示，本题中增加金属层维度，节点坐标变为三维坐标，因此模型中的决策变量相应变化为 $x_{ijk,lmk}^{r's'}$，其中 $r's'$ 表示上引脚为 r'、下引脚为 s' 的引脚对的三维坐标，(i,j,k) 为网络中任一方格中心的坐标，(l,m,n) 表示与 (i,j,k) 相邻的方格中心坐标，$x_{ijk,lmk}^{r's'}$ 表示从方格 (i,j,k) 至方格 (l,m,n) 是否对引脚对 $r's'$ 进行布线，为 0-1 变量，1 表示布线，0 表示不布线。模型中的所有变量 $x_{ijk,lmk}^{r's'}$ 依然均为 0-1 变量。

除此之外，由于一个通孔的电阻等于 5 个方格的导线，因此在最小化布线长度时需要将通孔的电阻转化为金属线的长度考虑进目标函数中：

$$\min \sum_{k\in L}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(l,m,k)\in N_{ijk}} x_{ijk,lmk}^{r's'} + 5*\sum_{(i,j)\in N}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(i,j,n)\in N_{ijk}} x_{ijk,ijn}^{r's'} \tag{19}$$

其中前半部分中的 $\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(l,m,k)\in N_{ijk}} x_{ijk,lmk}^{r's'}$ 表示在第 k 层中的布线长度和，引起前半部分对各层求和得到所有金属层平面的布线长度和。后半部分即考虑通孔的数量，针对网络中第 k 层的某个方格坐标为 (i,j,k)，从该点流向其他金属层 (i,j,n) 的流量总和为 $\sum_{r's'\in R'S'(i,j,n)\in N_{ijk}} x_{ijk,ijn}^{r's'}$（所有引脚对 rs 之和），由于网络中的每个方格只允许被一条金属线覆盖，即只有 1 个单位的流量流出，因此该值只能等于 1 或 0，也可以表示从该点在垂直方向上连出的线的数量，该线即为通道，其值表示通道的数量。对 (i,j) 进行遍历，求出每层

该点流出的流量总和，即可求出通道的总数量 $\sum_{(i,j)\in N}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(i,j,n)\in N_{ijk}} x_{ijk,ijn}^{r's'}$，并赋予权重5，即得到上述目标函数。

修正后的基于三维方格的模型如下：

$$\min \sum_{k\in L}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(l,m,k)\in N_{ijk}} x_{ijk,lmk}^{r's'} + 5*\sum_{(i,j)\in N}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(i,j,n)\in N_{ijk}} x_{ijk,ijn}^{r's'} \tag{20}$$

s.t.

$$\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = 1 \quad (i,j,k) = r', \forall\, r's' \in R'S' \tag{21}$$

$$\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 1 \quad (i,j,k) = s', \forall\, r's' \in R'S' \tag{22}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = \begin{cases} 1 & (i,j,k)\in R' \\ 0 & (i,j,k)\in S' \end{cases} \tag{23}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = \begin{cases} 0 & (i,j,k)\in R' \\ 1 & (i,j,k)\in S' \end{cases} \tag{24}$$

$$\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} - \sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 0 \quad (i,j,k) \neq r'、s', \forall\, r's' \in R'S' \tag{25}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} \leq 1 \quad \forall (i,j,k)\in N' \tag{26}$$

$$x_{ijk,lmn}^{r's'} \in \{0,1\} \tag{27}$$

与第一问约束建立过程类似（详细原理可参考本文 5.2.1），约束(21)、(23) 的第一个约束、(24)的第一个约束针对底层的上引脚的进出流量进行约束，约束(21)表示引脚对$r's'$的上引脚r'流出的流量为 1，约束(23) 的第一个约束表示所有引脚对在该点流出的流量和为 1，约束(24)的第一个约束表示所有引脚对在该点无流入的流量，即起点只有一条线连出，无金属线连入。

约束(22)、(23) 的第二个约束、(24)的第二个约束针对底层的下引脚的进出流量进行约束，约束(22)表示流入引脚对rs的下引脚s的流量为 1，约束(23) 的第二个约束表示所有引脚对流入该点的流量和为 1，约束(24)的第二个约束表示所有引脚对在该点无流出的流量，即下引脚只有一条线连入，无金属线连出。

约束(25)对非底层的上引脚与下引脚的方格进行约束，每个引脚对在该方格流入与流出的流量相等，同样为了约束金属线在三维空间中不断路，三维中的每个方格依然只能被一条线覆盖，因此所有引脚对在该点流出的流量等于流入的流量，并且不超过 1，即约束(26)。约束(27)为变量的 0-1 约束。

2）模型求解

依然调用 Gurobi 对上述模型进行求解，采用的方法是对偶单纯形法。利用本模型对于测例 1、测例 2 和测例 3 分别进行测试，发现采用多层金属层时，测例 1、测例 2 和测例 3

在问题二的背景下都有解，各测例结果如下图所示。其中，对于测例 1，路径长为 13，耗时 0.02s；对于测例 2，路径长为 63，耗时 3.94s；对于测例 3，路径长为 412，耗时 60.34s。

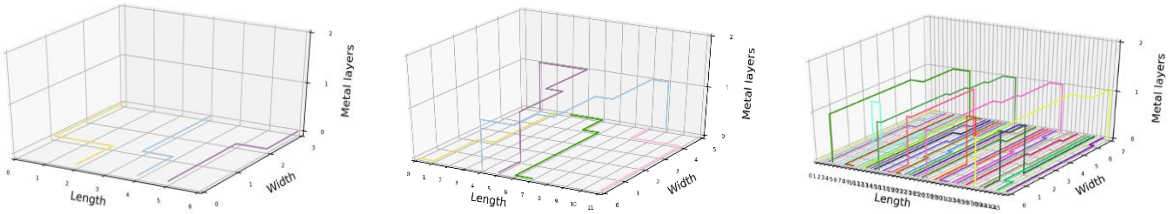


图 5：测例求解结果示意图

7.1.2 基于各层均允许下引脚处连出假设的模型

1) 模型建立

本小节模型在基于仅底层允许下引脚处连出假设的模型的基础上，对模型进行修正，由于这里模型允许金属线从各层连出，这从一定意义上来说可以减少通孔的数量，但仍然假设上引脚对从底层连入，在上述模型中只需要对下引脚的约束进行修改，即将前模型中对下引脚在底层流入流量等于 1，在其他金属层流量流入等于流出的约束，将约束修改为对相应引脚对的下引脚，各层总的流入网络流等于 1、流出网络流为 0，考虑到其他引脚对的金属线可能在与该引脚对的下引脚连出的其他层通过，还需要增加约束：所有引脚对流入该下引脚各层的总流量与流出的总流量的差值为 1，即可保证其他引脚对通过该下引脚各层时流入流量等于流出流量，且总流量不超过 1。于是模型修正后如下：

$$\min \sum_{k\in L}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(l,m,k)\in N_{ijk}} x_{ijk,lmk}^{r's'} + 5*\sum_{(i,j)\in N}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(i,j,n)\in N_{ijk}} x_{ijk,ijn}^{r's'} \tag{28}$$

s.t.

$$\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = 1 \quad (i,j,k) = r', \forall\, r's' \in R'S' \tag{29}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = 1 \quad (i,j,k) \in R' \tag{30}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 0 \quad (i,j,k) \in R' \tag{31}$$

$$\sum_{k\in L}\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 1 \quad (i,j) = s, \forall\, r's' \in R'S' \tag{32}$$

$$\sum_{k\in L}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = 0 \quad (i,j) = s, \forall\, r's' \in R'S' \tag{33}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} - \sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = 1 \quad (i,j) \in S \tag{34}$$

$$\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} - \sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 0 \quad (i,j,k) \neq r', (i,j) \neq s, \forall\, r's' \in R'S' \tag{35}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} \leq 1 \quad \forall (i,j,k) \in N' \tag{36}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} \leq 1 \quad \forall (i,j,k) \in N' \tag{37}$$

$$x_{ijk,lmn}^{r's'} \in \{0,1\} \tag{38}$$

其中约束(29)-(31)是对底层连入引脚对的流入、流出流量约束，与前一个问题设置相同，因此约束也相同，约束(32)-(34)是对引脚对处在各层的流量流入、流出的约束，约束（32）、（33）分别表示各引脚对在相应下引脚处各层总的流入网络流等于 1、流出网络流为 0，约束（34）表示所有引脚对在各下引脚处各层流入的总流量与流出的总流量差值为 1。约束（35）表示对于其他各点，即除底层上引脚、各层下引脚处的其他方格，每个引脚对流入该方格的流量与流出的流量相等，约束（36）、（37）对各层中的每个方格均有流入流量总和不超过 1，流出流量总和也不超过 1 的约束，即保证不短路的情况，最后是对变量的 0-1 约束。

2) 模型求解

依然调用 Gurobi 对上述模型进行求解，采用的方法是对偶单纯形法。利用本模型对于测例 1、测例 2 和测例 3 分别进行测试，发现采用多层金属层时，测例 1、测例 2 和测例 3 在问题二的背景下都有解，各测例结果如下图所示。其中，对于测例 1，路径长为 13，耗时 0.02s；对于测例 2，路径长为 53，耗时 1.73s；对于测例 3，路径长为 372，耗时 78.92s。

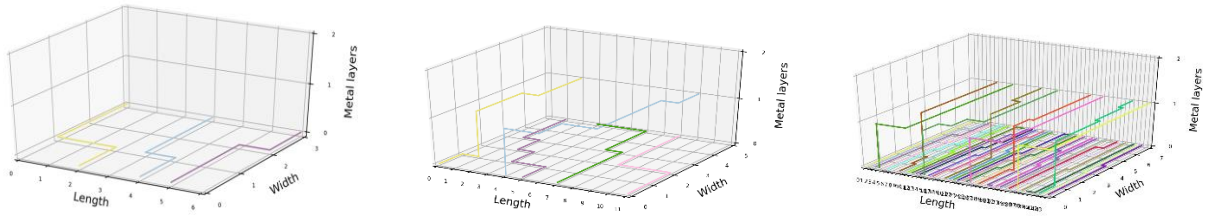


图 6：测例结果示意图

7.2 基于改进遗传算法的多层布线算法设计和分析

7.2.1 算法设计

考虑到面对一些通道布线问题，一层金属通道布线可能存在无解的状况，本研究将问题一的改进 A*单层布线算法拓展至多层，并利用遗传算法提高搜寻效率。遗传算法是基于“适者生存”的一种高度并行、随机和自适应的优化算法，它将问题的求解表示成“染色体”的适者生存过程，通过“染色体”群的一代一代不断进化，包括复制、交叉、变异等操作，最终收敛到“最适应环境的个体”，从而求得问题的最优解。遗传算法的两个显著特点是隐含并行性和全局解空间搜索。

将改进 A*算法与遗传算法结合后，算法的时间复杂度大大降低，该算法具体步骤为：

第一步：随机产生一个种群，作为问题的初代解，该种群的产生是基于解决问题一的改进$A*$单层布线算法拓展而来的，它跳出了面向问题一所建立的二维框架，采用了三维的搜索空间，并让结点的移动可以向上下左右前后六个方向自由移动。同样根据模型结果对于引脚对的排序较为敏感的观点，采用了引脚对数的阶乘的思想，随机的选择一些引脚对

搜索顺序。这样不仅可以提高收敛的速度，还能够确保个体基因的多样性；

第二步：采用二进制编码对种群中的个体进行编码，一个位能表示出 2 种状态的信息量，因此足够长的二进制染色体便能表示所有的特征。它编码、解码操作简单易行，交叉、变异等遗传操作便于实现，符合最小字符集编码原则；

第三步：考虑到适应度函数必须为非负的，本研究适应度是改进 A*的代价函数$f(n)$。改进$A*$的代价函数$f(n)$是节点n的综合优先级，由$g(n)$和$h(n)$组成。当我们选择下一个要遍历的节点时，我们总会选取综合优先级最高（值最小）的节点，$g(n)$是节点n距离起点的代价，$h(n)$是节点n距离终点的预计代价。特别的，当针对一个种群，即本研究中的一种全局不限策略，无解时，会给$f(n)$增加一个很大惩罚项，以便能够搜寻出完整的路径。按照这种方法计算种群中每个个体的适应度，为后续的个体选择提供依据；

第四步：对个体编码串进行解码处理后，可得到个体的表现型。由个体的表现型可计算出对应个体的目标函数值。根据最优化问题的类型，由目标函数值按一定的转换规则求出个体的适应度。根据适应度的高低选择参与繁衍的父体与母体，选择的原则是适应度越高的个体越可能被选中，以此不断淘汰适应度低的个体；

第五步：按轮盘赌选择方法执行遗传算法的选择操作，然后将当前群体中适应度最高的个体结构完整地复制到下一代群体中。并且当前群体中适应度最高的个体不参与交叉运算和变异运算，而是用它来代替掉本代群体中经过交叉、变异等操作后所产生的适应度最低的个体。对被选出的父体与母体执行遗传操作，即复制父体与母体的基因，并采用交叉、变异等算子产生出子代；

第六步：达到指定迭代次数时找出所有子代中适应度最高个体作为解返回并结束程序。

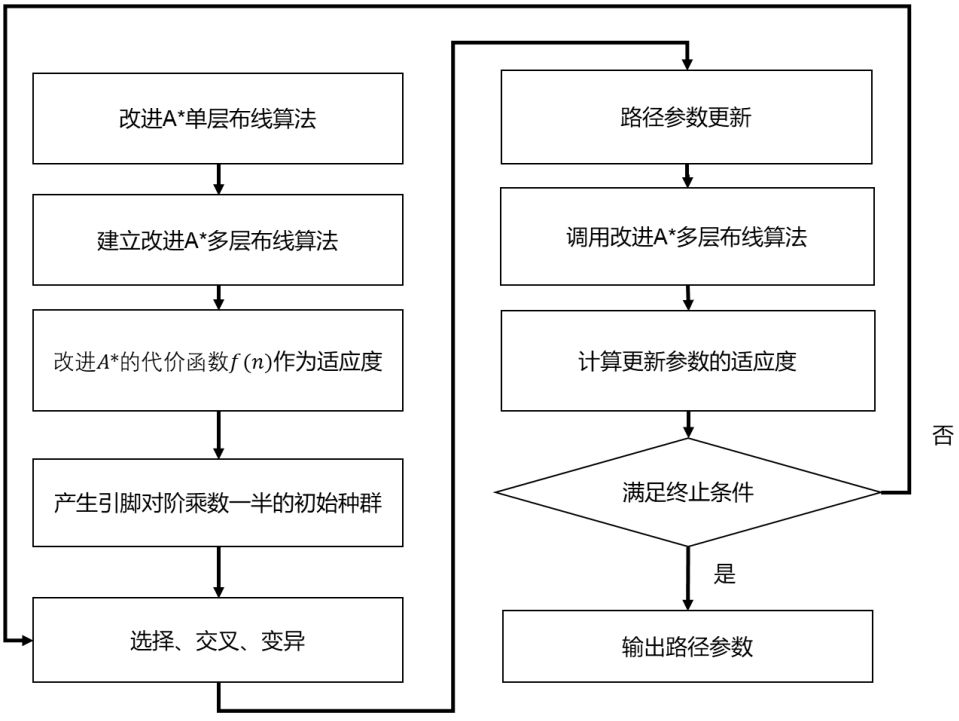


图 7：改进 GA 流程图

7.2.2 算法分析

利用本算法对于测例 1、测例 2 和测例 3 分别进行计算，发现采用多层金属层时，测例 1、测例 2 和测例 3 在问题二的背景下都有解，并且在应用遗传算法后，计算速度显著提升，各测例结果如下图所示。其中，对于测例 1，路径长为 13，耗时 0.03s；对于测例 2，路径长为 72，耗时 13.75s；对于测例 3，路径长为 412，耗时 128.54s。

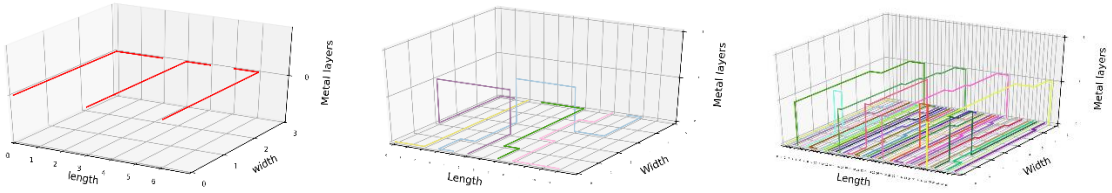


图 8：测例结果示意图

8 问题三的模型建立与求解

8.1 考虑通孔约束的多层三维布线模型建立与求解

8.1.1 模型建立

本题假设下引脚均需要从底层连出，即在问题二的第一个模型，即基于仅允许下引脚在底层连出的假设的模型基础上，加入任意两个通孔的间距必须大于等于 2 个格点的约束构建问题三的模型。同时本题认为任意两通孔包含连接不同金属层的通孔对，用任意两通孔的最短距离表示这两个通孔间的间距

为了找出通道所处的位置进而进行约束，本题引入新的变量 z_{ij} 来实现，由于题目要求最多三层金属层，因此可通过观察中间的金属层判断通孔的位置，因为无论是中间层与底层间或中间层与顶层间有通孔，均会通过中间层的方格，因此遍历中间层各点 $(i,j,1)$（1 表示中间层），若该点有流量流出（原模型已对中间层每个方格的流入量与流出量相等，且不超过一个单位进行了约束，即保证不短路），说明该点有与其他层连接的通孔，因此 $z_{ij}=\sum_{r's'\in R'S'}\sum_{(l,m,n)\in N_{ij1}} x_{ij1,lmn}^{r's'}$ 表示所有引脚对从中间层上的点 $(i,j,1)$ 流出的流量总和，当有流量流出时，即有通孔时，其值为 1，无则为 0。令 NE_{ij} 表示在单层平面上与方格 (i,j) 相邻的 8 个方格，如图 9 所示（当该方格位于布线平面的最外边或者顶角时，仅考虑与之相邻的方格，即不一定存在 8 个与之相邻的方格），这些方格与该方格的间距为 1 或者 $\sqrt{2}$，均小于 2，其他方格与该方格的间距均大于或等于 2。因此遍历中间层的所有点 (i,j)（或表示为 $(i,j,1)$），将其与邻接点的关系进行约束，当 z_{ij}=1 时，说明该点连接有通孔，当其邻接的八个方格 $z_{i'j'}$ 值之和 $\sum_{i'j'\in NE_{ij}} z_{i'j'}=\sum_{i'j'\in NE_{ij}}\sum_{r's'\in R'S'}\sum_{(l,m,n)\in N_{i'j'1}} x_{i'j'1,lmn}^{r's'}$ 为 0 时，说明这 8 个方格均无通孔，当有至少一个通孔时，该值 $\sum_{i'j'\in NE_{ij}} z_{i'j'}\geq 1$，不符合我们的约束条件，因此当 z_{ij} 为 1 时，

$\sum_{i'j' \in NE_{ij}} z_{i'j'}$ 必须为 0，因此其乘积 $z_{ij} * \sum_{i'j' \in NE_{ij}} z_{i'j'} = 0$。当 z_{ij} 为 0 时，不需要考虑其与周围方格的关系，因为该处无通孔，上述约束条件也成立。将该约束条件总结如下：

$$z_{ij} * \sum_{i'j' \in NE_{ij}} z_{i'j'} = z_{ij} * \sum_{i'j' \in NE_{ij}} \sum_{r's' \in R'S'} \sum_{(l,m,n) \in N_{i'j'1}} x^{r's'}_{i'j'1,lmn} = 0 \quad \forall (i,j) \in N \qquad (\ )$$

(1)	(2)	(3)
(4)	(i, j)	(5)
(6)	(7)	(8)

图 9：中间层平面上方格与相邻方格位置关系

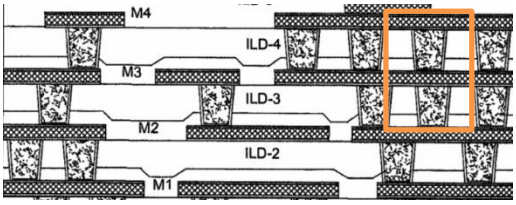


图 10：共 z 轴通道示意图

添加上述约束条件后调用 Gurobi 进行求解，各测试案例得到的结果见附录。

然而上述约束条件遗漏了一种情况：当两个通孔共 z 轴放置，如图 10 所示情况，因为这种情况下，从中间层的方格 $(i, j, 1)$ 流出的流量和依然为 1，因此上述约束对共 z 轴通孔无限制作用，对其他各情形均有限制作用，因此，针对这种情况建立新的约束条件。当出现这种情况时，由于其他约束的存在，只有两种情形：（1）一条金属线从底层通过该方格流入顶层；（2）一条金属线从顶层通过该方格流入底层。当出现情形（1）时，从底层流入该方格的流量和为 1，从该方格流出至顶层的流量和为 1，从顶层流入该方格的流量和为 0，从该方格流出至底层的流量和为 0，四者之和为 2。同理，当出现情形（2）时，从顶层流入该方格的流量和为 1，从该方格流出至底层的流量和为 1，从底层流入该方格的流量和为 0，从该方格流出至顶层的流量和为 0，四者之和仍为 2。因此建立如下约束条件：

$$zup\_out_{ij} = \sum_{r's' \in RS} \sum_{(l,m,n) \in N_{ij1}} x^{r's'}_{ij1,ij2} \quad \forall (i,j) \in N \qquad (39)$$

$$\text{zdown\_out}_{ij} = \sum_{r's' \in R'S'} \sum_{(l,m,n) \in N_{ij1}} x^{r's'}_{ij1,ij0} \quad \forall (i,j) \in N \qquad (40)$$

$$zup\_\text{in}_{ij} = \sum_{r's' \in R'S'} \sum_{(l,m,n) \in N_{ij1}} x^{r's'}_{ij2,ij1} \quad \forall (i,j) \in N \qquad (41)$$

$$zdown\_\text{in}_{ij} = \sum_{r's' \in R'S'} \sum_{(l,m,n) \in N_{ij1}} x^{r's'}_{ij0,ij1} \quad \forall (i,j) \in N \qquad (42)$$

$$zup\_out_{ij} + zdown\_\text{in}_{ij} + \text{zdown\_out}_{ij} + zup\_\text{in}_{ij} \le 1 \quad \forall (i,j) \in N \qquad (43)$$

其中 zup_out_{ij} 表示从 $(i, j, 1)$ 向顶层流出的流量和，zdown_out_{ij} 表示从 $(i, j, 1)$ 向底层流出的流量和，zup_in_{ij} 从顶层流至 $(i, j, 1)$ 的流量和，$zdown\_\text{in}_{ij}$ 从底层流至 $(i, j, 1)$ 的流量和，由于其他约束限制了只能一条金属线从一个方向通过该方格，因此四者之和的只

能为 0（该方格上下均无通孔连接）、1（该方格上下共连接一个通孔）、2（该方格上下均连接了通孔），因此这四个变量之和小于等于 1（或写成不等于 2）均可防止两个通孔共 z 轴放置情形的出现（这里引入的四组新的变量使说明更加直观，为减少变量数，也可直接对求和项相加进行约束，无需引入新变量）。

其他约束与目标函数均与第六章的第一个模型相同，完整模型表达式见附录。

8.1.2 模型求解

调用求解器 Gurobi 进行求解，无解，约束较多，考虑对假设进行松弛，问题二中讨论了下引脚可不从底层连出的情况，本题可同时或分别考虑上引脚可不从底层连入，下引脚可不从底层连出的情况。利用本模型对于测例 1、测例 2 和测例 3 分别进行测试，当下引脚必须从底层连出时，发现只有测例 1 在问题二的背景下有解。而当下引脚不必从底层连出时，发现测例 1、测例 2 和测例 3 在问题二的背景下都有解，结果如下图所示。其中，对于测例 1，路径长为 13，耗时 0.03s；对于测例 2，路径长为 83，耗时 53.83s；对于测例 3，路径长为 538，耗时 106.92s。

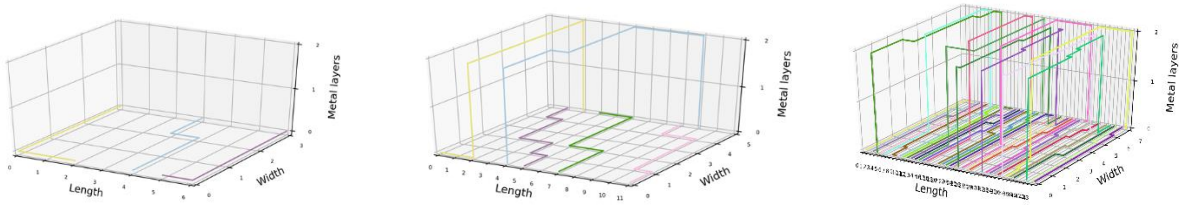


图 11：测例求解结果示意图

9 总结与展望

9.1 总结

本文分别考虑单层金属层、多层金属层、添加通孔约束的多层金属层三种情形下对“通道布线”问题进行优化。

针对问题一，建立二维坐标系以表示布线区域中的方格位置，建立相关集合以表示各方格与其他方格间是否可直接进行布线的关系，以保证金属线只能沿着直线或直角放置。本问题旨在最小化总布线长度，因此将该问题与传统最短路问题的异同点进行了分析，并对一般最短路模型进行改进，主要从寻找多个引脚对的最短路（总路径最短）和保证每两个引脚对线路间不相交为切入点建立相关约束，通过分别对上引脚、下引脚、非上下引脚处的方格进出流量关系与流量大小进行约束，构建了基于改进最短路的单层二维布线模型。通过调用求解器 Gurobi，研究发现只有测例 1 有解。考虑到这是一种基于线搜索的并行方法，本研究还从基于网格搜索的串行方法展开，采用基于改进 A*的多层布线算法对问题进行求解，它采用了阶乘的思想确定引脚对搜寻顺序，发现了和上述方法相同的结论。最后还分析了无解的情况，发现如果引脚对间存在同侧合法边界相交，则两对引脚不能在同一层中连接，此时一层金属通道布线问题无解。

针对问题二，考虑了多层金属层的布线优化，以实现不同金属线可共用一个方格而不会引起短路。首先在问题一建立的模型基础上，首先考虑将各金属层均映射到最底层，通过交叉点的数量与通道的关系建立二维布线模型，但存在一些该模型无法考虑到的情况，因此考虑三维空间，增加金属层维度，考虑各项约束，分别针对下引脚必须从底层连出与可从任意层连出两种假设下构建了两个基于改进最短路的多层三维布线模型，目标函数中考虑通孔的电阻，对布线方案进行优化。发现在多层布线时，这三个算例都有解，并且模型运算速度较快。另外，还采用了基于改进遗传算法的多层布线算法对问题进行了求解。它是对问题一提出的改进 A*算法的进一步提升，利用了改进 A*算法生成种群并获取适应度，极大的提高了算法的运行效率，最终发现了与上述方法相同的结论。

针对问题三，在问题二下引脚均从底层连出的假设的基础上增加了任意两个通孔的间距必须大于等于 2 个格点的约束，首先对通孔间距进行定义，考虑获取通孔所处的位置，发现所有通孔的位置均能通过中间金属层进行体现，以此为切入点，探讨该金属层上与有通孔方格的距离小于 2 的方格无通孔的约束，构建模型进行求解，发现该约束对共 z 轴通孔无约束，此时，三个测例都有可行解，当针对该情形通过探讨变量间的关系增加了相关约束，结果表明只有测例 1 有解，其他测例无解，也说明了本文相关假设稍有严格。

9.2 展望

随着器件关键尺寸进入深亚微米，串扰的问题越来越严重。它是指当信号在传输线上传播时，因电磁耦合对相邻的传输线产生的不期望的电压噪声干扰。在未来的通道布线研究中，应将串扰考虑进去，建立串扰噪声模型，以最大程度的减少电路的寄生效应。

参考文献

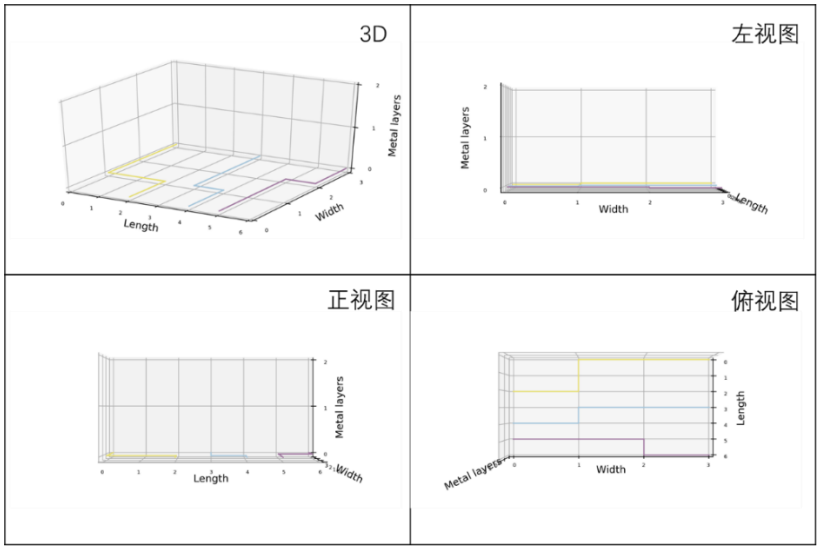
[1] 常晓夏. 超大规模集成电路串扰问题的研究[D].北京邮电大学,2006.

[2] 卢永江.基于拓扑分析的多层通道布线算法[J].电路与系统学报,2003(06):6-9.

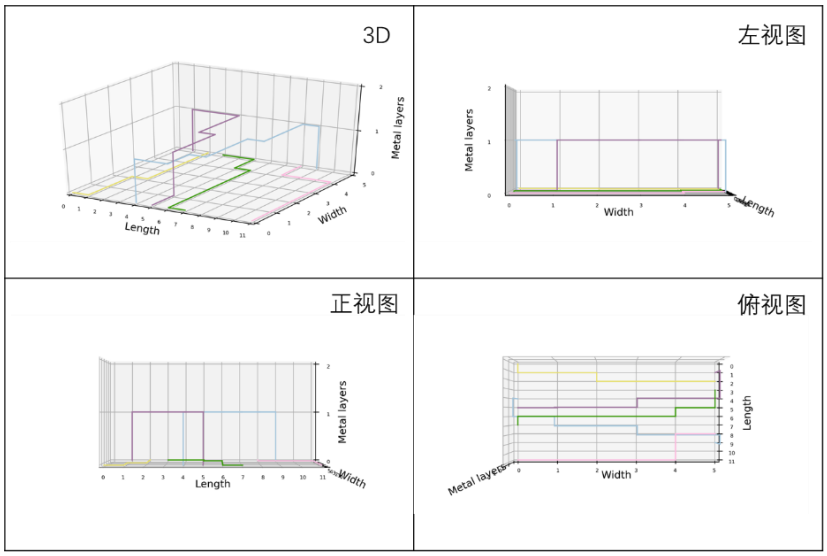
[3] 刘铁英,陈琛.一个具有最小串扰的多层通道集成电路布线算法[J].内蒙古大学学报(自然科学版),2002(06):714-717.

附　录

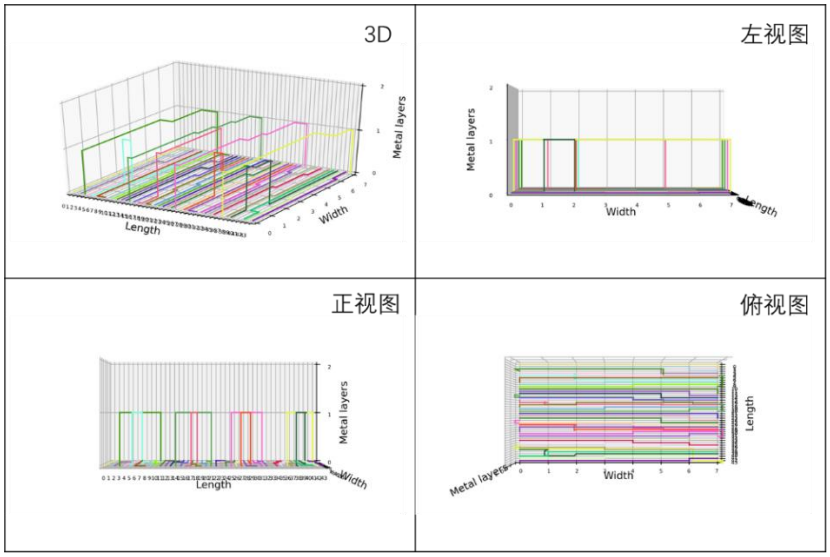
1. 求解结果展示



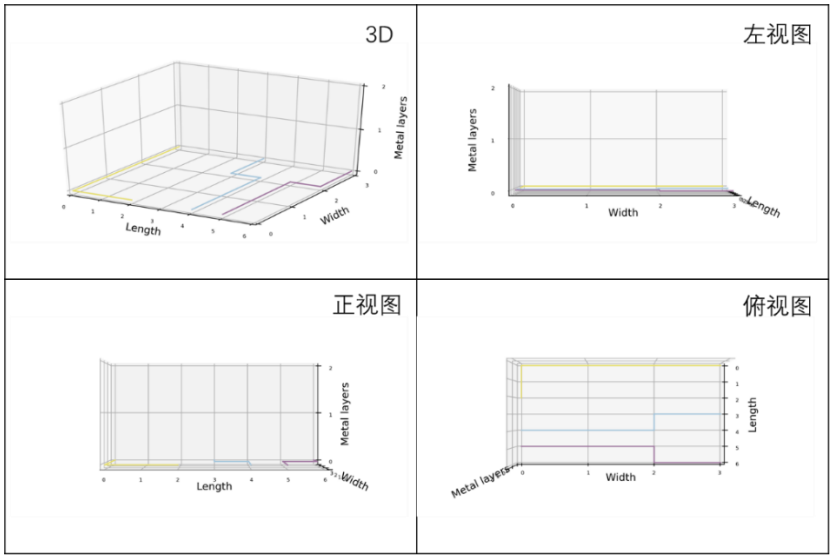
问题二_方法 1_test1



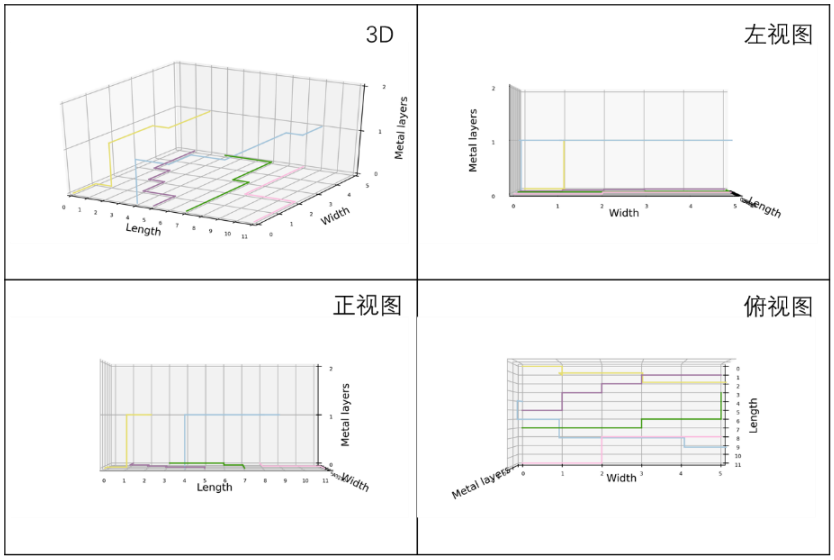
问题二_方法 1_test2



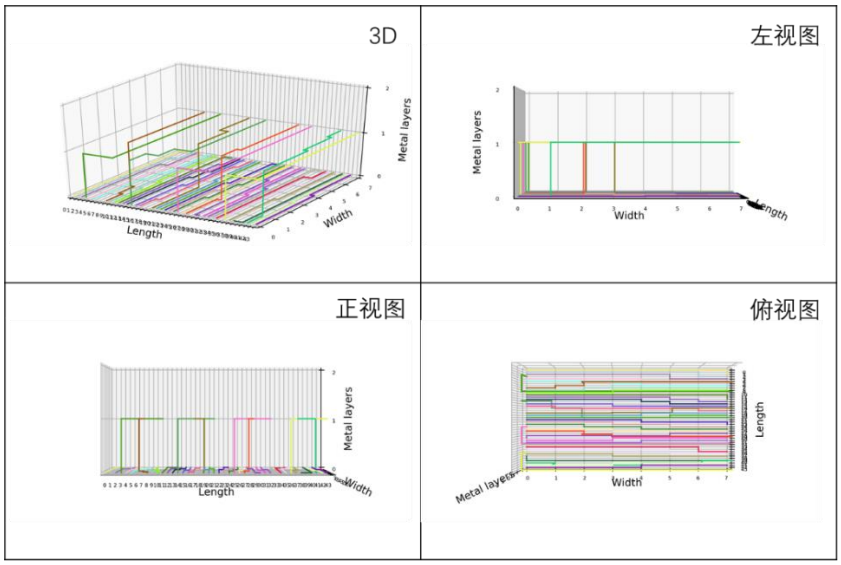
问题二_方法 1_test3



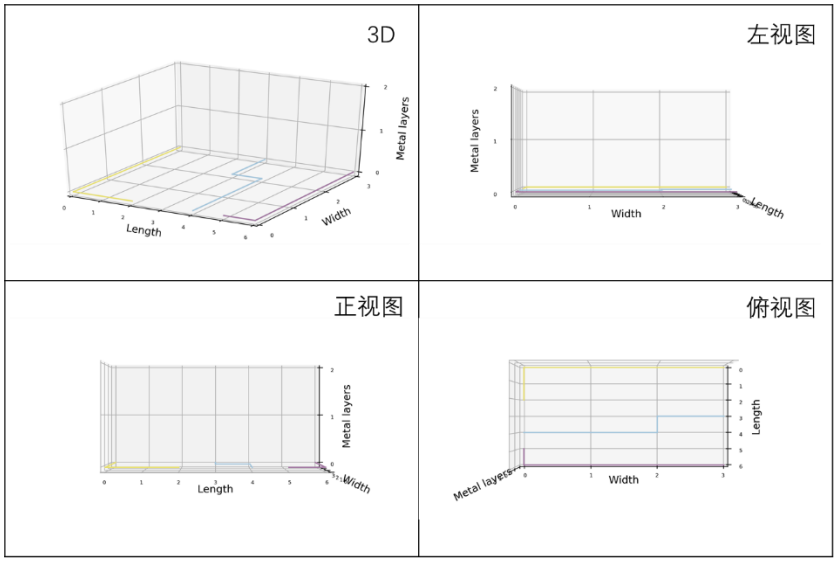
问题二_方法 2_test1



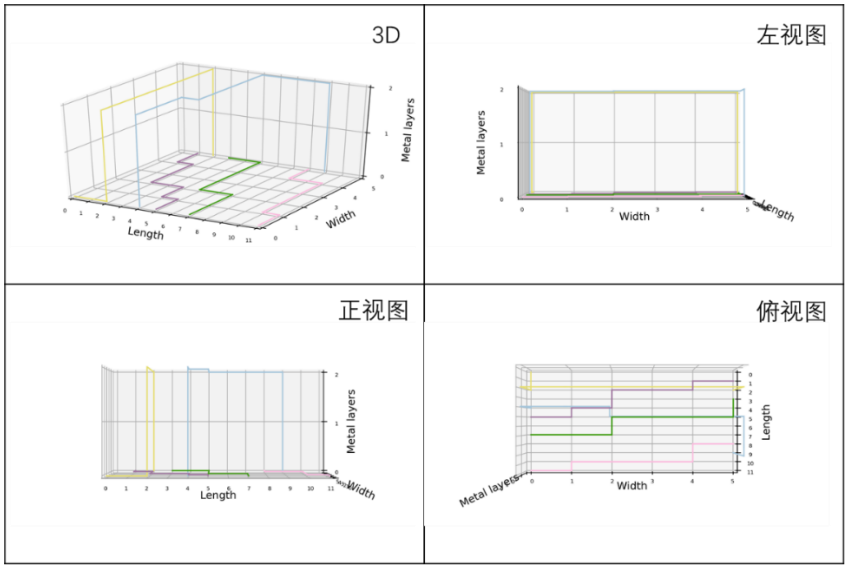
问题二_方法 2_test2



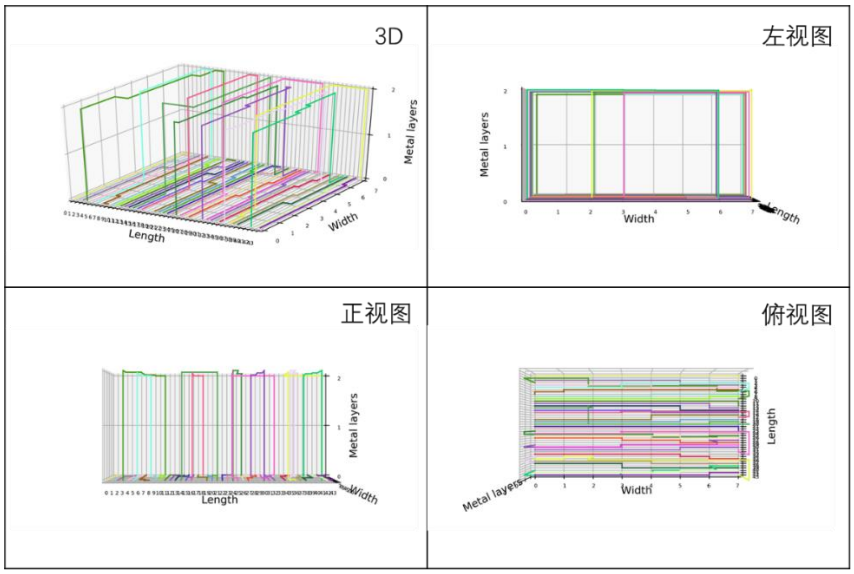
问题二_方法 2_test3



问题三_test1



问题三_test2



问题三_test3

2. 问题一算法程序

2.1 基于改进最短路的单层布线模型（P1）

```
# -*- coding: utf-8 -*-
from gurobipy import *
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

info = pd.read_excel("测例 1.xlsx", "Sheet1",header = None,skiprows=None)
info = info.values
length = info[1][2]
width = info[0][2]
paths_num = info[2][2]

starts = []
ends = []
starts_path = []
ends_path = []
for path in range(paths_num):    #该 width 和 0 的位置
    start = (info[3][path+2]-1, 0)
    end = (info[4][path+2]-1, width-1)
    start_path = (path,info[3][path+2]-1, 0)
    end_path = (path,info[4][path+2]-1, width-1)

    starts.append(start)
    ends.append(end)
    starts_path.append(start_path)
    ends_path.append(end_path)


x_index = {}
for path in range(paths_num):
    for i in range(length):
        for j in range(width):
            if i-1>=0:
                x_index[path,i,j,i-1,j] = 0
            if i+1<= length-1:
                x_index[path,i,j,i+1,j] = 0
            if j-1>=0:
```

```
                    x_index[path,i,j,i,j-1] = 0
                if j+1<= width-1:
                    x_index[path,i,j,i,j+1] = 0

try:
    mo = Model("MIP")

    #x = mo.addVars(point1_index.keys(),obj=1, vtype=GRB.BINARY, name='x')
    x = mo.addVars(x_index.keys(), vtype=GRB.BINARY, name='x')
    obj = 0
    for i in x_index.keys():
        obj = obj + x[i]
    mo.setObjective(obj, GRB.MINIMIZE)

    for i in range(length):
        for j in range(width):
            for path in range(paths_num):
                if (path,i,j) in starts_path:
                    mo.addConstr((x.sum(path,i,j,'*','*') == 1),"1")
                    #mo.addConstr((x.sum(path,'*','*',i,j) == 0),"2")
                else:
                    if (path,i,j) in ends_path:
                        #mo.addConstr((x.sum(path,i,j,'*','*') == 0),"3")
                        mo.addConstr((x.sum(path,'*','*',i,j) == 1),"4")
                    else:   #包含其他 path 经过起终点
                        mo.addConstr((x.sum(path,i,j,'*','*')  -  x.sum(path,'*','*',i,j)  ==
0),"5")
    for i in range(length):
        for j in range(width):
            if (i,j) in starts:
                mo.addConstr((x.sum("*",i,j,'*','*') == 1),"3")
                mo.addConstr((x.sum("*",'*','*',i,j) == 0),"3")
            else:
                if (i,j) in ends:
                    mo.addConstr((x.sum("*",i,j,'*','*') == 0),"3")
                    mo.addConstr((x.sum("*",'*','*',i,j) == 1),"3")
                else:
                    mo.addConstr((x.sum('*',i,j,'*','*') <= 1),"6")
                    mo.addConstr((x.sum('*','*','*',i,j) <= 1),"7")
```

```
    '''
    for i in range(length):
        for j in range(width):
            for path in range(paths_num):
                if (path,i,j) in starts_path:
                    mo.addConstr((x.sum(path,i,j,'*','*') == 1),"1")
                    #mo.addConstr((x.sum(path,'*','*',i,j) == 0),"2")
                else:
                    if (path,i,j) in ends_path:
                        #mo.addConstr((x.sum(path,i,j,'*','*') == 0),"3")
                        mo.addConstr((x.sum(path,'*','*',i,j) == 1),"4")
    for i in range(length):
        for j in range(width):
            if (i,j) in starts:
                mo.addConstr((x.sum("*",i,j,'*','*') == 1),"3")
                mo.addConstr((x.sum("*",'*','*',i,j) == 0),"3")
            else:
                if (i,j) in ends:
                    mo.addConstr((x.sum("*",i,j,'*','*') == 0),"3")
                    mo.addConstr((x.sum("*",'*','*',i,j) == 1),"3")
                else:
                    mo.addConstr((x.sum('*',i,j,'*','*') <= 1),"6")
                    mo.addConstr((x.sum('*','*','*',i,j) <= 1),"7")
            for path in range(paths_num):
                if (i,j) not in starts:
                    if (i,j) not in ends:
                        mo.addConstr((x.sum(path,i,j,'*','*') - x.sum(path,'*','*',i,j) ==
0),"5")
    '''
    mo.optimize()
    bestX = {}
    bestX_1 = []
    for i in x_index.keys():
        bestX[i] = x[i].x
        if bestX[i] ==1:
            print(i)
            bestX_1.append(i)
    #print(bestX.keys())
```

```
        #plt.title('a')
        colors = ['r','g','b', 'c', 'm', 'y']
        for arc in bestX_1:
            for path in range(paths_num):
                if arc[0] == path:
                    #color = np.zeros((100,100))*(path+1)
                    plt.plot([arc[1],arc[3]],[arc[2],arc[4]],c=colors[arc[0]])

        #plt.legend()
        plt.axis([-0.1,length-0.9,0,width-1])
        my_x_ticks = np.arange(0,length, 1)
        my_y_ticks = np.arange(0,width, 1)
        plt.xticks(my_x_ticks)
        plt.yticks(my_y_ticks)
        plt.xlabel('Length')
        plt.ylabel('Width')
        plt.savefig('P1_1', dpi=300, bbox_inches = 'tight')
        plt.show()


except GurobiError as exception:
        print('Error code ' + str(exception.errno) + ": " + str(exception))
```

2.2 基于改进 A*算法的单层布线算法

```
import math
import copy
import time
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# position of the best path search
class Position:
        def __init__(self, x, y, z):
            self.x = x
            self.y = y
            self.z = z
```

```
    def getX(self):

        return self.x

    def getY(self):
        return self.y

    def getZ(self):
        return self.z

class PathData:
    def __init__(self):
        # list of paths that can be continued, containing: a path and Fscore
        self.openList = []
        # List of coordinates already visited
        self.closedList = []

    # function to put a path as openPath in the open list
    def putInOpenList(self, path, F):
        openPath = [path, F]
        self.openList.append(openPath)

    # function to put an element in the closed list
    def putInClosedList(self, pos):
        self.closedList.append(pos)

    # function to check if a position is in the closed list
    def inClosedList(self, position):
        for element in self.closedList:
            if element.getX() == position.getX() and element.getY() == position.getY() and
element.getZ() == position.getZ():
                return True
        return False

    # function to delete element from the open list
    def deleteFromOpenList(self, path):
        counter = 0
        for element in self.openList:
            if path is element[0]:
                del self.openList[counter]
```

```
                counter += 1

    def getLowestFScore(self, endPos):
        lowestFscore = 10000
        H = 10000
        bestPath = self.openList[0]
        for possiblePath in self.openList:
            if possiblePath[1] < lowestFscore:
                lowestFscore = possiblePath[1]
                H = calcDistance(possiblePath[0][-1], endPos)
                bestPath = possiblePath[0]
            elif possiblePath[1] == lowestFscore and calcDistance(possiblePath[0][-1],
endPos) < H:
                lowestFscore = possiblePath[1]
                H = len(possiblePath[0])
                bestPath = possiblePath[0]
        return bestPath

class Grid:
    # dus dit is voor het maken van het grid van 17 bij 12
    def __init__(self, width, height, length):
        self.grid = [[[0 for x in range(length)] for x in range(height)] for x in range(width)]
        self.width = width
        self.height = height
        self.length = length

    def setPointToValue(self, position, value):
        self.grid[position.getX()][position.getY()][position.getZ()] = value

    def printGrid(self):
        for row in self.grid:
            print (row)

    def setStartEnd(self, startPos, endPos):
        self.start = startPos
        self.end = endPos
        self.grid[startPos.getX()][startPos.getY()][startPos.getZ()] = 1
        self.grid[endPos.getX()][endPos.getY()][endPos.getZ()] = 2

    def getStart(self):
```

```
        return self.start

    def getEnd(self):
        return self.end

    # find all possible next paths in the grid
    def getPossibleNextPaths(self, currentPath, pathData):
        nextPaths = []
        x = currentPath[-1].getX()
        y = currentPath[-1].getY()
        z = currentPath[-1].getZ()

        possiblePath = copy.copy(currentPath)
        rightx = x + 1
        if rightx <= self.width-1 and (self.grid[rightx][y][z] == 0 or self.grid[rightx][y][z] ==
2) and not pathData.inClosedList(Position(rightx, y, z)):
            possiblePath.append(Position(rightx, y, z))
            nextPaths.append(possiblePath)

        possiblePath = copy.copy(currentPath)
        leftx = x - 1
        if leftx >= 0 and (self.grid[leftx][y][z] == 0 or self.grid[leftx][y][z] == 2) and not
pathData.inClosedList(Position(leftx, y, z)):
            possiblePath.append(Position(leftx, y, z))
            nextPaths.append(possiblePath)

        possiblePath = copy.copy(currentPath)
        upy = y + 1
        if upy <= self.height-1 and (self.grid[x][upy][z] == 0 or self.grid[x][upy][z] == 2) and
not pathData.inClosedList(Position(x, upy, z)):
            possiblePath.append(Position(x, upy, z))
            nextPaths.append(possiblePath)

        possiblePath = copy.copy(currentPath)
        downy = y - 1
        if downy >= 0 and (self.grid[x][downy][z] == 0 or self.grid[x][downy][z] == 2) and
not pathData.inClosedList(Position(x, downy, z)):
            possiblePath.append(Position(x, downy, z))
            nextPaths.append(possiblePath)
```

```
            possiblePath = copy.copy(currentPath)
            shangz = z + 1
            if shangz <= self.length-1 and (self.grid[x][y][shangz] == 0 or self.grid[x][y][shangz]
== 2) and not pathData.inClosedList(Position(x, y, shangz)):
                  possiblePath.append(Position(x, y, shangz))
                  nextPaths.append(possiblePath)

            possiblePath = copy.copy(currentPath)
            xiaz = z - 1
            if xiaz >= 0 and (self.grid[x][y][xiaz] == 0 or self.grid[x][y][xiaz] == 2) and not
pathData.inClosedList(Position(x, y, xiaz)):
                  possiblePath.append(Position(x, y, xiaz))
                  nextPaths.append(possiblePath)

            return nextPaths

      def drawPath(self, path):
            X =[]
            Y= []
            Z= []
            for i in range(self.width):
                  for j in range(self.height):
                        for k in range(self.length):
                              if self.grid[i][j][k] == 1:
                                    self.grid[i][j][k] = 0
            for step in path:
                  self.grid[step.getX()][step.getY()][step.getZ()] = 1
                  print(step.getX(),step.getY(),step.getZ())
                  X.append(step.getX())
                  Y.append(step.getY())
                  Z.append(step.getZ())
            figure = ax.plot(X, Y, Z, c='r')

def findPath(grid):

      pathData = PathData()

      currentPath = []
      currentPath.append(grid.getStart())
      endPos = grid.getEnd()
```

```
    while not(currentPath[-1].getX() == endPos.getX() and currentPath[-1].getY() == endPos.getY() and currentPath[-1].getZ() == endPos.getZ()):

        pathData.deleteFromOpenList(currentPath)

        # add currentPos to closedList
        pathData.putInClosedList(currentPath[-1])
        # expand the openList
        continuationList = grid.getPossibleNextPaths(currentPath, pathData)

        for possiblePath in continuationList:
            # calculate cost and heuristic
            G = len(possiblePath) - 1
            H = calcDistance(possiblePath[-1], endPos)
            F = G+H
            # put path,cost and heuristic in the openList like [[path], F-score]
            pathData.putInOpenList(possiblePath, F)

        # set currentPath to the lowest F-score path in the openList
        currentPath = pathData.getLowestFScore(endPos)
        grid.setPointToValue(currentPath[-1], 1)
        # time.sleep(0.3)
        # print ""
        # grid.printGrid()
        # print pathData.closedList
    print ("visited:")
    grid.printGrid()
    print ("found path:")
    grid.drawPath(currentPath)
    grid.printGrid()

# function to calculate manhattan distance
def calcDistance(pos1, pos2):
    distance = abs(pos1.getX() - pos2.getX()) + abs(pos1.getY() - pos2.getY()) + abs(pos1.getZ() - pos2.getZ())
    return distance

if __name__ == "__main__":
    time1 = time.time()
```

```
c=7
k=4
# new a figure and set it into 3d
fig = plt.figure(figsize=(8,4))
ax = fig.gca(projection='3d')
# draw the figure, the color is r = red
# set figure information
#ax.set_title("3D")
ax.set_xlabel("length",size=12)
ax.set_ylabel("width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.axis([0,7,0,3])
my_x_ticks = np.arange(0,7,1)
my_y_ticks = np.arange(0,4,1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=8)
#ax.set_title("Test3_Wiring")

newGrid1 = Grid(c,k,3)#长， 宽=width, height
start1 = Position(2,k-1,0) #x-1,y
end1 = Position(0,0,0)
newGrid1.setStartEnd(start1, end1)
findPath(newGrid1)
newGrid2 = Grid(c,k,3)#长， 宽=width, height
start2 = Position(4,k-1,0) #x-1,y
end2 = Position(3,0,0)
newGrid2.setStartEnd(start2, end2)
findPath(newGrid2)
newGrid3 = Grid(c,k,3)#长， 宽=width, height
start3 = Position(5,k-1,0) #x-1,y
end3 = Position(6,0,0)
newGrid3.setStartEnd(start3, end3)
findPath(newGrid3)

'''
newGrid = Grid(5,4,8)
```

```
    start = Position(3,3,0)
    end = Position(4,2,7)

    newGrid.setStartEnd(start, end)
    '''
    # obstruction
    #newGrid.setPointToValue(Position(4,3), 3)
    #newGrid.setPointToValue(Position(1,3), 3)
    #newGrid.setPointToValue(Position(2,3), 3)
    #newGrid.setPointToValue(Position(3,3), 3)
    time2 = time.time()
    print ("time in seconds:")
    print (time2-time1)
    plt.savefig('P2_1_test3_RETURN', dpi=300, bbox_inches = 'tight')
    plt.show()
```

3. 问题二算法程序

3.1 基于改进最短路的多层二维布线模型 (P2_3_2dem)

```
# -*- coding: utf-8 -*-

from gurobipy import *
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random

def randomcolor(path):
    colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
    color = ""
    random.seed(path*5)
    for i in range(6):
        color += colorArr[random.randint(0,14)]
    return "#"+color

info = pd.read_excel("测例 3.xlsx", "Sheet1",header = None,skiprows=None)
info = info.values
```

```
length = info[1][2]
width = info[0][2]
paths_num = info[2][2]



starts = []
ends = []
starts_path = []
ends_path = []
for path in range(paths_num):    #该 width 和 0 的位置
    start = (info[3][path+2]-1, 0)
    end = (info[4][path+2]-1, width-1)
    start_path = (path,info[3][path+2]-1, 0)
    end_path = (path,info[4][path+2]-1, width-1)

    starts.append(start)
    ends.append(end)
    starts_path.append(start_path)
    ends_path.append(end_path)



x_index = {}
y_index = {}
for path in range(paths_num):
    for i in range(length):
        for j in range(width):
            if i-1>=0:
                x_index[path,i,j,i-1,j] = 0
            if i+1<= length-1:
                x_index[path,i,j,i+1,j] = 0
            if j-1>=0:
                x_index[path,i,j,i,j-1] = 0
            if j+1<= width-1:
                x_index[path,i,j,i,j+1] = 0
for i in range(length):
    for j in range(width):
        if (i,j) not in starts:
            if (i,j) not in ends:
                y_index[i,j] = 0
```

```
try:
    mo = Model("MIP")

    #x = mo.addVars(point1_index.keys(),obj=1, vtype=GRB.BINARY, name='x')
    x = mo.addVars(x_index.keys(), vtype=GRB.BINARY, name='x')
    y = mo.addVars(y_index.keys(), lb=0, name='x')
    z = mo.addVars(y_index.keys(), lb=0, name='x')



    obj = 0
    for i in x_index.keys():
        obj = obj + x[i]
    for i in y_index.keys():
        obj = obj + 5*z[i]
    mo.setObjective(obj, GRB.MINIMIZE)

    for i in range(length):
        for j in range(width):
            for path in range(paths_num):
                if (path,i,j) in starts_path:
                    mo.addConstr((x.sum(path,i,j,'*','*') == 1),"1")
                    mo.addConstr((x.sum(path,'*','*',i,j) == 0),"2")
                else:
                    if (path,i,j) in ends_path:
                        mo.addConstr((x.sum(path,i,j,'*','*') == 0),"3")
                        mo.addConstr((x.sum(path,'*','*',i,j) == 1),"4")
                    else:
                        mo.addConstr((x.sum(path,i,j,'*','*') - x.sum(path,'*','*',i,j) ==
0),"5")
    for i in range(length):
        for j in range(width):
            if (i,j) in starts:
                mo.addConstr((x.sum("*",i,j,'*','*') == 1),"3")
                mo.addConstr((x.sum("*",'*','*',i,j) == 0),"3")
            else:
                if (i,j) in ends:
                    mo.addConstr((x.sum("*",i,j,'*','*') == 0),"3")
                    mo.addConstr((x.sum("*",'*','*',i,j) == 1),"3")
                else:
```

```
                    mo.addConstr((x.sum('*',i,j,'*','*') <= 3),"6")
                    mo.addConstr((y[i,j] == x.sum('*',i,j,'*','*')),"6")
                    #mo.addConstr((z[i,j] == max_(0,y[i,j])),"23")
                    #mo.addConstr((z[i,j] >= 0),"23")
                    mo.addConstr((z[i,j] >= y[i,j]-1),"23")
                    mo.addConstr((x.sum('*','*','*',i,j) <= 3),"7")


mo.optimize()
bestX = {}
bestX_1 = []
for i in x_index.keys():
    bestX[i] = x[i].x
    if bestX[i] ==1:
        print(i)
        bestX_1.append(i)
#print(bestX)


colors = ['r','g','b', 'c', 'm', 'y']
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            #plt.plot([arc[1],arc[3]],[arc[2],arc[4]],c=colors[arc[0]])
            #plt.plot([arc[1],arc[3]],[arc[2],arc[4]],c="r")
            plt.plot([arc[1],arc[3]],[arc[2],arc[4]],c=color)

plt.axis([-1,length,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
plt.xticks(my_x_ticks)
plt.yticks(my_y_ticks)
#plt.figure(figsize=(10, 10), dpi=300)
plt.xlabel('x')
plt.ylabel('y')
plt.tick_params(labelsize=6)
plt.title('Test3_Wring')
plt.savefig('P2_2D_test3', dpi=300, bbox_inches = 'tight')
plt.show()
```

```
except GurobiError as exception:
    print('Error code ' + str(exception.errno) + ": " + str(exception))
```

3.2 基于改进最短路的多层三维布线模型

（1）假设下引脚均从底层连出相关代码（P2_1_ReturnToFirstFloor)

```
# -*- coding: utf-8 -*-

from gurobipy import *
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from mpl_toolkits.mplot3d import axes3d

#绘图颜色
def randomcolor(path):
    colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
    color = ""
    random.seed(path*5)
    for i in range(6):
        color += colorArr[random.randint(0,14)]
    return "#"+color


info = pd.read_excel("测例 2.xlsx", "Sheet1",header = None,skiprows=None)
info = info.values
length = info[1][2]
width = info[0][2]
paths_num = info[2][2]


starts = []
ends = []
starts_path = []
ends_path = []
for path in range(paths_num):    #该 width 和 0 的位置
    start = (info[3][path+2]-1, 0)
```

```
        end = (info[4][path+2]-1, width-1)
        start_path = (path,info[3][path+2]-1, 0)
        end_path = (path,info[4][path+2]-1, width-1)

        starts.append(start)
        ends.append(end)
        starts_path.append(start_path)
        ends_path.append(end_path)


x_index = {}
y_index = {}
for path in range(paths_num):
        for i in range(length):
                for j in range(width):
                        for k in range(3):
                                if i-1>=0:      #针对每一个点有出去的
                                        x_index[path,i,j,k,i-1,j,k] = 0
                                if i+1<= length-1:
                                        x_index[path,i,j,k,i+1,j,k] = 0
                                if j-1>=0:
                                        x_index[path,i,j,k,i,j-1,k] = 0
                                if j+1<= width-1:
                                        x_index[path,i,j,k,i,j+1,k] = 0
                                if k-1>=0:
                                        x_index[path,i,j,k,i,j,k-1] = 0
                                if k+1<=2:
                                        x_index[path,i,j,k,i,j,k+1] = 0

try:
        mo = Model("MIP")

        #x = mo.addVars(point1_index.keys(),obj=1, vtype=GRB.BINARY, name='x')
        x = mo.addVars(x_index.keys(), vtype=GRB.BINARY, name='x')
        #y = mo.addVars(y_index.keys(), lb=0, name='x')
        #z = mo.addVars(y_index.keys(), lb=0, name='x')


        obj = 0
```

```
    for path in range(paths_num):
        for k in range(3):
            for i in range(length):
                for j in range(width):
                    obj = obj + x.sum(path,i,j,k,'*','*',k)

    for path in range(paths_num):
        for i in range(length):
            for j in range(width):
                for k in range(3):
                    obj = obj + 5* x.sum(path,i,j,k,i,j,'*')


    mo.setObjective(obj, GRB.MINIMIZE)
    ####### 约束条件：终点只有进无出，起点只出不进
    for i in range(length):     #对 path
        for j in range(width):
            if (i,j) in starts:
                for path in range(paths_num):
                    ##起点其他金属层
                    mo.addConstr((x.sum(path,i,j,1,'*','*','*') - x.sum(path,'*','*','*',i,j,1)
== 0),"5") #起点所有 path 1\2 层进出守恒
                    mo.addConstr((x.sum(path,i,j,2,'*','*','*') - x.sum(path,'*','*','*',i,j,2)
== 0),"5")

                    if (path,i,j) in starts_path: #(i,j)对应 path
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 1),"1")   #起点 0
层出去为 1，进来为 0
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")
                    else:   #(i,j)不对应的 path
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"1")
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")



            elif (i,j) in ends:
                for path in range(paths_num):
                    ##终点其他金属层
```

```
                        mo.addConstr((x.sum(path,i,j,1,'*','*','*')  -  x.sum(path,'*','*','*',i,j,1)
== 0),"5")
                        mo.addConstr((x.sum(path,i,j,2,'*','*','*')  -  x.sum(path,'*','*','*',i,j,2)
== 0),"5")
                        if (path,i,j) in ends_path: #终点所有层对每条 path
                            mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"3")
                            mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 1),"4")
                        else:
                            mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"1")
                            mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")

            else:
                for k in range(3):
                    for path in range(paths_num):
                        mo.addConstr((x.sum(path,i,j,k,'*','*','*')                    -
x.sum(path,'*','*','*',i,j,k) == 0),"5")
                    mo.addConstr((x.sum('*',i,j,k,'*','*','*') <= 1),"6")

            for k in range(3): #对终点 ij 所有 path 每个 k 进和出小于等于 1
                mo.addConstr((x.sum('*',i,j,k,'*','*','*') <= 1),"6")
                mo.addConstr((x.sum('*','*','*','*',i,j,k) <= 1),"6")


    #mo.setParam("MIPGap",0.3)
    mo.optimize()
    bestX = {}
    bestX_1 = []
    for i in x_index.keys():
        bestX[i] = x[i].x
        if bestX[i] ==1:
            print(i)
            bestX_1.append(i)
    #print(bestX)


    #图 1
    fig = plt.figure(figsize = (8,4))
    ax = fig.gca(projection='3d')
    # set figure information
```

```
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_1_test3_RETURN——1', dpi=300, bbox_inches = 'tight')
plt.show()


#图 2
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=-90)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
```

```
            if arc[0] == path:
                color = randomcolor(path)
                figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_1_test3_RETURN——2', dpi=300, bbox_inches = 'tight')
plt.show()

#图 3
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=90)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
```

```
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_1_test3_RETURN——3', dpi=300, bbox_inches = 'tight')
plt.show()


#图 4
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=90,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_1_test3_RETURN——4', dpi=300, bbox_inches = 'tight')
plt.show()


#图 5
fig = plt.figure(figsize = (8,4))
```

```
    ax = fig.gca(projection='3d')
    # set figure information
    #ax.set_title("3D")

    ax.set_xlabel("Length",size=12)
    ax.set_ylabel("Width",size=12)
    ax.set_zlabel("Metal layers",size=12)
    #ax.view_init(elev=90,azim=0)
    # draw the figure, the color is r = red
    for arc in bestX_1:
        for path in range(paths_num):
            if arc[0] == path:
                color = randomcolor(path)
                figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
    #figure = ax.plot(X, Y, Z, c='r')
    ax.axis([0,length-1,0,width-1])
    my_x_ticks = np.arange(0,length,1)
    my_y_ticks = np.arange(0,width, 1)
    my_z_ticks = np.arange(0,3,1)
    ax.set_xticks(my_x_ticks)
    ax.set_yticks(my_y_ticks)
    ax.set_zticks(my_z_ticks)
    plt.tick_params(labelsize=6)
    #ax.set_title("Test3_Wiring")
    plt.savefig('P2_1_test3_RETURN——5', dpi=300, bbox_inches = 'tight')
    plt.show()
except GurobiError as exception:
    print('Error code ' + str(exception.errno) + ": " + str(exception))
```

（2）假设下引脚可从各层连出的相关代码（P2_2_notReturnToFirstFloor)

```
# -*- coding: utf-8 -*-

from gurobipy import *
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from mpl_toolkits.mplot3d import axes3d
```

```
def randomcolor(path):
    colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
    color = ""
    random.seed(path*5)
    for i in range(6):
        color += colorArr[random.randint(0,14)]
    return "#"+color

info = pd.read_excel("测例 2.xlsx", "Sheet1",header = None,skiprows=None)
info = info.values

length = info[1][2]
width = info[0][2]
paths_num = info[2][2]


starts = []
ends = []
starts_path = []
ends_path = []
for path in range(paths_num):   #该 width 和 0 的位置
    start = (info[3][path+2]-1, 0)
    end = (info[4][path+2]-1, width-1)
    start_path = (path,info[3][path+2]-1, 0)
    end_path = (path,info[4][path+2]-1, width-1)

    starts.append(start)
    ends.append(end)
    starts_path.append(start_path)
    ends_path.append(end_path)


x_index = {}
y_index = {}
for path in range(paths_num):   ###改一维的？？？？？
    for i in range(length):
        for j in range(width):
            for k in range(3):
                if i-1>=0:      #针对每一个点有出去的，添加的删除
```

```
                        x_index[path,i,j,k,i-1,j,k] = 0
                        #x_index[path,i-1,j,k,i,j,k] = 0
                    if i+1<= length-1:
                        x_index[path,i,j,k,i+1,j,k] = 0
                        #x_index[path,i+1,j,k,i,j,k] = 0
                    if j-1>=0:
                        x_index[path,i,j,k,i,j-1,k] = 0
                        #x_index[path,i,j-1,k,i,j,k] = 0
                    if j+1<= width-1:
                        x_index[path,i,j,k,i,j+1,k] = 0
                        #x_index[path,i,j+1,k,i,j,k] = 0
                    if k-1>=0:
                        x_index[path,i,j,k,i,j,k-1] = 0
                        #x_index[path,i,j,k-1,i,j,k] = 0
                    if k+1<=2:
                        x_index[path,i,j,k,i,j,k+1] = 0
                        #x_index[path,i,j,k+1,i,j,k] = 0

try:
    mo = Model("MIP")

    #x = mo.addVars(point1_index.keys(),obj=1, vtype=GRB.BINARY, name='x')
    x = mo.addVars(x_index.keys(), vtype=GRB.BINARY, name='x')
    #y = mo.addVars(y_index.keys(), lb=0, name='x')
    #z = mo.addVars(y_index.keys(), lb=0, name='x')


    obj = 0
    for path in range(paths_num):
        for k in range(3):
            for i in range(length):
                for j in range(width):
                    obj = obj + x.sum(path,i,j,k,'*','*',k)

    for path in range(paths_num):
        for i in range(length):
            for j in range(width):
                for k in range(3):
                    obj = obj + 5* x.sum(path,i,j,k,i,j,'*')
```

```
    mo.setObjective(obj, GRB.MINIMIZE)
    ####### 约束条件：终点只有进无出，起点只出不进
    for i in range(length):     #对 path
        for j in range(width):
            if (i,j) in starts:
                for path in range(paths_num):
                    mo.addConstr((x.sum(path,i,j,1,'*','*','*') - x.sum(path,'*','*','*',i,j,1)
== 0),"5") #起点所有 path 1\2 层进出守恒
                    mo.addConstr((x.sum(path,i,j,2,'*','*','*') - x.sum(path,'*','*','*',i,j,2)
== 0),"5")

                    if (path,i,j) in starts_path: #起点 0\1\2 层
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 1),"1")   #起点 0
层出去为 1，进来为 0
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")
                    else:
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"1")
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")



            elif (i,j) in ends:
                for path in range(paths_num):
                    if (path,i,j) in ends_path: #终点所有层对每条 path
                        mo.addConstr((x.sum(path,i,j,'*','*','*','*') == 0),"3")
                        mo.addConstr((x.sum(path,'*','*','*',i,j,'*') == 1),"4")
                    else:
                        for k in range(3):
                            mo.addConstr((x.sum(path,i,j,k,'*','*"*',)                    -
x.sum(path,'*','*','*',i,j,k) == 0),"5")

            else:
                for k in range(3):
                    for path in range(paths_num):
                        mo.addConstr((x.sum(path,i,j,k,'*','*','*',)                    -
x.sum(path,'*','*','*',i,j,k) == 0),"5")
                    mo.addConstr((x.sum('*',i,j,k,'*','*','*',) <= 1),"6")
```

```
            for k in range(3): #对终点 ij 所有 path 每个 k 进和出小于等于 1
                mo.addConstr((x.sum('*',i,j,k,'*','*','*',) <= 1),"6")
                mo.addConstr((x.sum('*','*','*','*',i,j,k) <= 1),"6")




#mo.setParam("MIPGap",0.3)
mo.optimize()
bestX = {}
bestX_1 = []
for i in x_index.keys():
    bestX[i] = x[i].x
    if bestX[i] ==1:
        print(i)
        bestX_1.append(i)
#print(bestX)


#图 1
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
```

```
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_2_test3_RETURN——1', dpi=300, bbox_inches = 'tight')
plt.show()


#图 2
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=-90)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_2_test3_RETURN——2', dpi=300, bbox_inches = 'tight')
plt.show()
```

```
#图 3
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=90)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_2_test3_RETURN——3', dpi=300, bbox_inches = 'tight')
plt.show()


#图 4
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
```

```
ax.view_init(elev=90,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P2_2_test3_RETURN——4', dpi=300, bbox_inches = 'tight')
plt.show()


#图 5
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
#ax.view_init(elev=90,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
```

```
    my_x_ticks = np.arange(0,length,1)
    my_y_ticks = np.arange(0,width, 1)
    my_z_ticks = np.arange(0,3,1)
    ax.set_xticks(my_x_ticks)
    ax.set_yticks(my_y_ticks)
    ax.set_zticks(my_z_ticks)
    plt.tick_params(labelsize=6)
    #ax.set_title("Test3_Wiring")
    plt.savefig('P2_2_test3_RETURN——5', dpi=300, bbox_inches = 'tight')
    plt.show()
except GurobiError as exception:
    print('Error code ' + str(exception.errno) + ": " + str(exception))
```

3.3 基于改进遗传算法的单层布线算法

```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
from numpy import linalg as la, inexact
import time
import pandas as pd
import copy
from mpl_toolkits.mplot3d import axes3d


# position of the best path search
class Position:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def getX(self):

        return self.x

    def getY(self):
        return self.y
```

```
    def getZ(self):
        return self.z

class PathData:
    def __init__(self):
        # list of paths that can be continued, containing: a path and Fscore
        self.openList = []
        # List of coordinates already visited
        self.closedList = []

    # function to put a path as openPath in the open list
    def putInOpenList(self, path, F):
        openPath = [path, F]
        self.openList.append(openPath)

    # function to put an element in the closed list
    def putInClosedList(self, pos):
        self.closedList.append(pos)

    # function to check if a position is in the closed list
    def inClosedList(self, position):
        for element in self.closedList:
            if element.getX() == position.getX() and element.getY() == position.getY() and
element.getZ() == position.getZ():
                return True
        return False

    # function to delete element from the open list
    def deleteFromOpenList(self, path):
        counter = 0
        for element in self.openList:
            if path is element[0]:
                del self.openList[counter]
            counter += 1

    def getLowestFScore(self, endPos):
        lowestFscore = 10000
        H = 10000
        bestPath = self.openList[0]
        for possiblePath in self.openList:
```

```
                if possiblePath[1] < lowestFscore:
                    lowestFscore = possiblePath[1]
                    H = calcDistance(possiblePath[0][-1], endPos)
                    bestPath = possiblePath[0]
                elif possiblePath[1] == lowestFscore and calcDistance(possiblePath[0][-1],
endPos) < H:
                    lowestFscore = possiblePath[1]
                    H = len(possiblePath[0])
                    bestPath = possiblePath[0]
        return bestPath
    def getRandomFScore(self, endPos):
        H = 10000
        bestPath = self.openList[random.randint(0,len(self.openList))]
        H = calcDistance(bestPath[-1], endPos)
        return bestPath

class Grid:
    # dus dit is voor het maken van het grid van 17 bij 12
    def __init__(self, width, height, length):
        self.grid = [[[0 for x in range(length)] for x in range(height)] for x in range(width)]
        self.width = width
        self.height = height
        self.length = length

    def setPointToValue(self, position, value):
        self.grid[position.getX()][position.getY()][position.getZ()] = value

    def printGrid(self):
        for row in self.grid:
            print (row)
        return self.grid

    def setStartEnd(self, startPos, endPos):
        self.start = startPos
        self.end = endPos
        self.grid[int(startPos.getX())][int(startPos.getY())][int(startPos.getZ())] = 1
        self.grid[int(endPos.getX())][int(endPos.getY())][int(endPos.getZ())] = 2

    def getStart(self):
        return self.start
```

```
    def getEnd(self):
        return self.end

    # find all possible next paths in the grid
    def getPossibleNextPaths(self, currentPath, pathData):
        nextPaths = []
        x = int(currentPath[-1].getX())
        y = int(currentPath[-1].getY())
        z = int(currentPath[-1].getZ())

        possiblePath = copy.copy(currentPath)
        a=0
        rightx = x + 1
        if rightx <= self.width-1 and (self.grid[rightx][y][z] == 0 or self.grid[rightx][y][z] ==
2) and not pathData.inClosedList(Position(rightx, y, z)):
            possiblePath.append(Position(rightx, y, z))
            nextPaths.append(possiblePath)
            a=a+1

        possiblePath = copy.copy(currentPath)
        leftx = x - 1
        if leftx >= 0 and (self.grid[leftx][y][z] == 0 or self.grid[leftx][y][z] == 2) and not
pathData.inClosedList(Position(leftx, y, z)):
            possiblePath.append(Position(leftx, y, z))
            nextPaths.append(possiblePath)
            a=a+1
        possiblePath = copy.copy(currentPath)
        upy = y + 1
        if upy <= self.height-1 and (self.grid[x][upy][z] == 0 or self.grid[x][upy][z] == 2) and
not pathData.inClosedList(Position(x, upy, z)):
            possiblePath.append(Position(x, upy, z))
            nextPaths.append(possiblePath)
            a=a+1
        possiblePath = copy.copy(currentPath)
        downy = y - 1
        if downy >= 0 and (self.grid[x][downy][z] == 0 or self.grid[x][downy][z] == 2) and
not pathData.inClosedList(Position(x, downy, z)):
            possiblePath.append(Position(x, downy, z))
            nextPaths.append(possiblePath)
```

```
                a=a+1
            possiblePath = copy.copy(currentPath)
            shangz = z + 1
            if shangz <= self.length-1 and (self.grid[x][y][shangz] == 0 or self.grid[x][y][shangz]
== 2) and not pathData.inClosedList(Position(x, y, shangz)):
                possiblePath.append(Position(x, y, shangz))
                nextPaths.append(possiblePath)
                a=a+1
            possiblePath = copy.copy(currentPath)
            xiaz = z - 1
            if xiaz >= 0 and (self.grid[x][y][xiaz] == 0 or self.grid[x][y][xiaz] == 2) and not
pathData.inClosedList(Position(x, y, xiaz)):
                possiblePath.append(Position(x, y, xiaz))
                nextPaths.append(possiblePath)
                a=a+1
            if a==0:
                return []
            else:
                return nextPaths

        def drawPath(self, path):
            X =[]
            Y= []
            Z= []
            for i in range(self.width):
                for j in range(self.height):
                    for k in range(self.length):
                        if self.grid[i][j][k] == 1:
                            self.grid[i][j][k] = 0
            for step in path:
                self.grid[step.getX()][step.getY()][step.getZ()] = 1
                print(step.getX(),step.getY(),step.getZ())
                X.append(step.getX())
                Y.append(step.getY())
                Z.append(step.getZ())

            #figure = ax.plot(X, Y, Z, c='r')

def findPath(grid):
```

```
    pathData = PathData()

    currentPath = []
    currentPath.append(grid.getStart())
    endPos = grid.getEnd()

    while not(currentPath[-1].getX() == endPos.getX() and currentPath[-1].getY() ==
endPos.getY() and currentPath[-1].getZ() == endPos.getZ()):

        pathData.deleteFromOpenList(currentPath)

        # add currentPos to closedList
        pathData.putInClosedList(currentPath[-1])
        # expand the openList
        continuationList = grid.getPossibleNextPaths(currentPath, pathData)
        if len(continuationList)==0:
            print("无解")
            F=1000
            break

        for possiblePath in continuationList:
            # calculate cost and heuristic
            G = len(possiblePath) - 1
            H = calcDistance(possiblePath[-1], endPos)
            F = G+H
            # put path,cost and heuristic in the openList like [[path], F-score]
            pathData.putInOpenList(possiblePath, F)

        # set currentPath to the lowest F-score path in the openList
        currentPath = pathData.getLowestFScore(endPos)
        grid.setPointToValue(currentPath[-1], 1)
        # time.sleep(0.3)
        # print ""
        # grid.printGrid()
        # print pathData.closedList
    #print ("visited:")
    #grid.printGrid()
    print ("found path:")
    # new a figure and set it into 3d
```

```
    '''
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    # draw the figure, the color is r = red
    # set figure information
    ax.set_title("3D")
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    grid.drawPath(currentPath)
    '''
    G = len(currentPath) - 1
    H = calcDistance(currentPath[-1], endPos)
    if H>0:
        F=10
    else:
        F = G+H
    grid=grid.printGrid()
    return F,grid

# function to calculate manhattan distance
def calcDistance(pos1, pos2):
    distance = abs(pos1.getX() - pos2.getX()) + abs(pos1.getY() - pos2.getY()) + abs(pos1.getZ() - pos2.getZ())
    return distance

def get_total_distance(FF):
    Fsum=FF.sum()
    return Fsum

def improve(paths_order,paths_num,length,width,height,starts,ends):
    newGrid = Grid(length,width,height)#长，宽=width, height
    FF=[]
    for i in paths_order:
        start = Position(starts[i][0],starts[i][1],starts[i][2]) #x-1,y
        end = Position(ends[i][0],ends[i][1],ends[i][2])
        newGrid.setStartEnd(start, end)
        F,grid=findPath(newGrid)
        FF.append(F)
```

```
            x=grid
            for i in range(length):
                for j in range(width):
                    for k in range(height):
                        if grid[i][j][k] == 1:
                            grid[i][j][k] = 3
        for i in range(length):
            for j in range(width):
                for k in range(height):
                    if grid[i][j][k] == 3:
                        grid[i][j][k] = 1
        x=grid
        return sum(FF),x

#适应度
def selection(population,grade):
            graded=grade
            graded=[x[1] for x in sorted(graded)]
            retain_length=int(len(graded)*retain_rate)
            parents=graded[:retain_length]
            for chromosome in graded[retain_length:]:
                    if random.random()<random_rate:
                            parents.append(chromosome)
            return parents

#交叉繁殖
def crossover(parents):
            #还需要的子代个数，保持总数 count 稳定
            target_count=count-len(parents)
            #子代列表
            children=[]
            while len(children)<target_count:
                    #父母代索引
                    male_index=random.randint(0,len(parents)-1)
                    female_index=random.randint(0,len(parents)-1)
                    if male_index!=female_index:
                            #父母代个体
                            male=parents[male_index]
                            female=parents[female_index]
```

```
                #交叉规则
                left=random.randint(0,len(male)-3)
                right=random.randint(left+1,len(male))
                gene1=male[left:right]
                gene2=female.copy()
                for j in gene1:
                        gene2.remove(j)
                child=gene1+gene2
                # gene2=female[left:right]
                #
                # child1_c=male[right:]+male[:right]
                # child2_c = female[right:] +female[:right]
                # child1=child1_c.copy()
                # child2 = child2_c.copy()
                # for i in gene2:
                #         child1_c.remove(i)
                # for i in gene1:
                #         child2_c.remove(i)
                # child1[left:right]=gene2
                # child2[left:right] = gene1
                # child1[right:]=child1_c[0:len(child1)-right]
                # child1[:left]=child1_c[len(child1)-right:]
                #
                # child2[right:] = child2_c[0:len(child1) - right]
                # child2[:left] = child2_c[len(child1) - right:]
                # children.append(child1)
                # children.append(child2)
                children.append(child)
        # if target_count>0:
        #         children=children[: target_count + 1]
        return children

#变异
def mutation(children):
        for i in range(len(children)):
                if random.random()<mutation_rate:
                        child=children[i]
                        u=random.randint(1,len(child)-4)
                        v=random.randint(u+1,len(child)-3)
```

```
                    w = random.randint(v+1, len(child) - 2)
                    child=child[0:u]+child[v:w]+child[u:v]+child[w:]
                    children[i]=child
            return children

#获得最好结果
def get_result(population,grade):
        graded= grade
        graded=sorted(graded)
        return graded[0][0],graded[0][1]
def pailie(data):
    output=[]
    # 当只有一个元素的时候排列只有一个，直接返回
    if len(data)==1:
        return [data]
    # 当有 n 个元素的时候：假设除了第一个元素外，后边的 n-1 个元素已经拍好了，
于是：
    for i in data:
        data_c=data.copy()  #这句不是递归的步骤；这一句是复制数组的副本 ，找到后
n-1 个元素
        data_c.remove(i)
        # 这个就是递推公式了
        output+=[[i]+j for j in pailie(data_c)]
    return output




if __name__ == "__main__":
    time1 = time.time()

    #强者存活率
    retain_rate=0.3
    #弱者存活概率
    random_rate=0.5
    #变异率
```

```
mutation_rate=0.3
#迭代次数
item_time=2

info = pd.read_excel("测例 1.xlsx", "Sheet1",header = None,skiprows=None)
info = info.values
length = info[1][2]
width = info[0][2]
height=3
paths_num = info[2][2]
starts = []
ends = []

for path in range(paths_num):   #该 width 和 0 的位置
    start = (info[3][path+2]-1, 0,0)
    end = (info[4][path+2]-1, width-1,0)
    starts.append(start)
    ends.append(end)
order=math.factorial(paths_num)
data=[i for i in range (paths_num)]
if order>50:
    #种群数
    count=50
    order=random.sample(pailie(data),50)
else:
    count=order
    order=pailie(data)
population=[]
grade=[]
for i in range(count):
    paths_order=order[i]
    #population.append(index)
    imp1,imp2=improve(paths_order,paths_num,length,width,height,starts,ends)
    population.append(imp2)
    grade.append([imp1,imp2])
i=0
distance_list=[]
result_path_list=[]
while i<item_time:
```

```
        parents= selection(population,grade)
        if len(parents)<count:
            children=crossover(parents)
            children=mutation(children)
            population=parents+children
        distance,result_path=get_result(population,grade)
        distance_list.append(distance)
        result_path_list.append(result_path)
        i=i+1



    plt.show()
```

4. 问题三算法程序

4.1 考虑通孔约束的多层三维布线模型

1）未考虑共 *z* 轴通孔约束的模型求解代码（为本小节3）中代码的一部分）与结果

2）模型

$$\min \sum_{k\in L}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(l,m,k)\in N_{ijk}} x_{ijk,lmk}^{r's'} + 5^{*}\sum_{(i,j)\in N}\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'}\sum_{(i,j,n)\in N_{ijk}} x_{ijk,ijn}^{r's'}$$

s.t.

$$\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = 1 \quad (i,j,k) = r', \forall\, r's' \in R'S' \quad (1)$$

$$\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 1 \quad (i,j,k) = s', \forall\, r's' \in R'S' \quad (2)$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} = \begin{cases} 1 & (i,j,k)\in R' \quad (3) \\ 0 & (i,j,k)\in S' \quad (4) \end{cases}$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = \begin{cases} 0 & (i,j,k)\in R' \quad (5) \\ 1 & (i,j,k)\in S' \quad (6) \end{cases}$$

$$\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} - \sum_{(i,j,k)\in N'} x_{lmn,ijk}^{r's'} = 0 \quad (i,j,k) \neq r'、s', \forall\, r's' \in R'S' \quad (7)$$

$$\sum_{r's'\in R'S'}\sum_{(i,j,k)\in N'} x_{ijk,lmn}^{r's'} \le 1 \quad \forall (i,j,k)\in N' \quad (8)$$

$$z_{ij} * \sum_{i'j'\in NE_{ij}} z_{i'j'} = z_{ij} * \sum_{i'j'\in NE_{ij}}\sum_{r's'\in R'S'}\sum_{(l,m,n)\in N_{i'j'1}} x_{i'j'1,lmn}^{r's'} = 0 \quad \forall (i,j)\in N$$

$$zup\_out_{ij} = \sum_{r's'\in RS}\sum_{(l,m,n)\in N_{ij1}} x_{ij1,ij2}^{r's'} \quad \forall (i,j)\in N$$

$$z\text{down\_out}_{ij} = \sum_{r's'\in R'S'}\sum_{(l,m,n)\in N_{ij1}} x_{ij1,ij0}^{r's'} \quad \forall (i,j)\in N$$

$$zup\_\text{in}_{ij} = \sum_{r's'\in R'S'}\sum_{(l,m,n)\in N_{ij1}} x_{ij2,ij1}^{r's'} \quad \forall (i,j)\in N$$

$$zdown\_\text{in}_{ij} = \sum_{r's'\in R'S'}\sum_{(l,m,n)\in N_{ij1}} x_{ij0,ij1}^{r's'} \quad \forall (i,j)\in N$$

$$zup\_out_{ij} + zdown\_\text{in}_{ij} + z\text{down\_out}_{ij} + zup\_\text{in}_{ij} \le 1 \quad \forall (i,j)\in N$$

$$x_{ijk,lmn}^{r's'} \in \{0,1\} \quad (9)$$

3）考虑共 *z* 轴通孔约束的模型求解代码 (P3)

```
# -*- coding: utf-8 -*-

from gurobipy import *

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import random

from mpl_toolkits.mplot3d import axes3d
```

```
def randomcolor(path):
    colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
    color = ""
    random.seed(path*5)
    for i in range(6):
        color += colorArr[random.randint(0,14)]
    return "#"+color


info = pd.read_excel("测例2.xlsx", "Sheet1",header = None,skiprows=None)
info = info.values


length = info[1][2]
width = info[0][2]
paths_num = info[2][2]



starts = []
ends = []
starts_path = []
ends_path = []
for path in range(paths_num):    #该width和0的位置
    start = (info[3][path+2]-1, 0)
    end = (info[4][path+2]-1, width-1)
    start_path = (path,info[3][path+2]-1, 0)
    end_path = (path,info[4][path+2]-1, width-1)


    starts.append(start)
```

```
        ends.append(end)
        starts_path.append(start_path)
        ends_path.append(end_path)



x_index = {}
y_index = {}
for path in range(paths_num):   ###改一维的？？？？？
    for i in range(length):
        for j in range(width):
            for k in range(3):
                if i-1>=0:      #针对每一个点有出去的，添加的删除
                    x_index[path,i,j,k,i-1,j,k] = 0
                    #x_index[path,i-1,j,k,i,j,k] = 0
                if i+1<= length-1:
                    x_index[path,i,j,k,i+1,j,k] = 0
                    #x_index[path,i+1,j,k,i,j,k] = 0
                if j-1>=0:
                    x_index[path,i,j,k,i,j-1,k] = 0
                    #x_index[path,i,j-1,k,i,j,k] = 0
                if j+1<= width-1:
                    x_index[path,i,j,k,i,j+1,k] = 0
                    #x_index[path,i,j+1,k,i,j,k] = 0
                if k-1>=0:
                    x_index[path,i,j,k,i,j,k-1] = 0
                    #x_index[path,i,j,k-1,i,j,k] = 0
                if k+1<=2:
```

```
                        x_index[path,i,j,k,i,j,k+1] = 0
                        #x_index[path,i,j,k+1,i,j,k] = 0


try:
    mo = Model("MIP")

    #x = mo.addVars(point1_index.keys(),obj=1, vtype=GRB.BINARY, name='x')
    x = mo.addVars(x_index.keys(), vtype=GRB.BINARY, name='x')
    z = mo.addVars(range(length),range(width),name = 'z')
    #y = mo.addVars(y_index.keys(), lb=0, name='x')
    #z = mo.addVars(y_index.keys(), lb=0, name='x')
    # zup_out = mo.addVars(range(length),range(width),name = 'zup_out')
    # zdown_out = mo.addVars(range(length),range(width),name = 'zdown_out')
    # zup_in = mo.addVars(range(length),range(width),name = 'zup_in')
    # zdown_in = mo.addVars(range(length),range(width),name = 'zdown_in')



    obj = 0
    for path in range(paths_num):
        for k in range(3):
            for i in range(length):
                for j in range(width):
                    obj = obj + x.sum(path,i,j,k,'*','*',k)


    for path in range(paths_num):
        for i in range(length):
            for j in range(width):
```

```
                for k in range(3):
                    obj = obj + 5* x.sum(path,i,j,k,i,j,'*')



    mo.setObjective(obj, GRB.MINIMIZE)
    ####### 约束条件：终点只有进无出，起点只出不进
    for i in range(length):     #对path
        for j in range(width):
            if (i,j) in starts:
                for path in range(paths_num):
                    mo.addConstr((x.sum(path,i,j,1,'*','*','*') - x.sum(path,'*','*','*',i,j,1)
== 0),"5") #起点所有path 1\2层进出守恒
                    mo.addConstr((x.sum(path,i,j,2,'*','*','*') - x.sum(path,'*','*','*',i,j,2)
== 0),"5")

                    if (path,i,j) in starts_path: #起点0\1\2层
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 1),"1")    #起点0
层出去为1，进来为0
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")
                    else:
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"1")
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")



            elif (i,j) in ends:
                for path in range(paths_num):
                    mo.addConstr((x.sum(path,i,j,1,'*','*','*') - x.sum(path,'*','*','*',i,j,1)
```

```
== 0),"5")
                    mo.addConstr((x.sum(path,i,j,2,'*','*','*') - x.sum(path,'*','*','*',i,j,2)
== 0),"5")
                    if (path,i,j) in ends_path: #终点所有层对每条path
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"3")
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 1),"4")
                    else:
                        mo.addConstr((x.sum(path,i,j,0,'*','*','*') == 0),"1")
                        mo.addConstr((x.sum(path,'*','*','*',i,j,0) == 0),"2")

            else:
                for k in range(3):
                    for path in range(paths_num):
                        mo.addConstr((x.sum(path,i,j,k,'*','*','*') -
x.sum(path,'*','*','*',i,j,k) == 0),"5")
                    mo.addConstr((x.sum('*',i,j,k,'*','*','*') <= 1),"6")

            for k in range(3): #对终点ij所有path每个k进和出小于等于1
                mo.addConstr((x.sum('*',i,j,k,'*','*','*') <= 1),"6")
                mo.addConstr((x.sum('*','*','*','*',i,j,k) <= 1),"6")



            mo.addConstr((z[i,j] == x.sum('*',i,j,1,'*','*','*')),'z')
            neighb_hole_values = 0
            if i-1 >= 0:
                neighb_hole_values += z[i-1,j]
            if i+1 <= length-1:
                neighb_hole_values += z[i+1,j]
```

```
            if j-1 >= 0:
                neighb_hole_values += z[i,j-1]
            if j+1 <= width-1:
                neighb_hole_values += z[i,j+1]
            if i-1 >= 0:
                if j-1 >=0:
                    neighb_hole_values += z[i-1,j-1]
            if i-1 >= 0:
                if j+1 <= width-1:
                    neighb_hole_values += z[i-1,j+1]
            if i+1 <= length-1:
                if j-1 >=0:
                    neighb_hole_values += z[i+1,j-1]
            if i+1 <= length-1:
                if j+1 <= width-1:
                    neighb_hole_values += z[i+1,j+1]
            mo.addConstr(neighb_hole_values*z[i,j] == 0)



#mo.params.NonConvex = 2
#mo.setParam("MIPGap",0.3)
mo.optimize()
bestX = {}
bestX_1 = []
for i in x_index.keys():
```

```
        bestX[i] = x[i].x
        if bestX[i] ==1:
            print(i)
            bestX_1.append(i)
#print(bestX)


#图1
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
```

```
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P3_test2——1', dpi=300, bbox_inches = 'tight')
plt.show()



#图2
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=-90)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
```

```
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P3_test2——2', dpi=300, bbox_inches = 'tight')
plt.show()

#图3
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=0,azim=90)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
        if arc[0] == path:
            color = randomcolor(path)
            figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
```

```
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P3_test2——3', dpi=300, bbox_inches = 'tight')
plt.show()



#图4
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")


ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
ax.view_init(elev=90,azim=0)
# draw the figure, the color is r = red
for arc in bestX_1:
    for path in range(paths_num):
```

```
            if arc[0] == path:
                color = randomcolor(path)
                figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
#figure = ax.plot(X, Y, Z, c='r')
ax.axis([0,length-1,0,width-1])
my_x_ticks = np.arange(0,length,1)
my_y_ticks = np.arange(0,width, 1)
my_z_ticks = np.arange(0,3,1)
ax.set_xticks(my_x_ticks)
ax.set_yticks(my_y_ticks)
ax.set_zticks(my_z_ticks)
plt.tick_params(labelsize=6)
#ax.set_title("Test3_Wiring")
plt.savefig('P3_test2——4', dpi=300, bbox_inches = 'tight')
plt.show()



#图5
fig = plt.figure(figsize = (8,4))
ax = fig.gca(projection='3d')
# set figure information
#ax.set_title("3D")

ax.set_xlabel("Length",size=12)
ax.set_ylabel("Width",size=12)
ax.set_zlabel("Metal layers",size=12)
#ax.view_init(elev=90,azim=0)
```

```
        # draw the figure, the color is r = red
        for arc in bestX_1:
            for path in range(paths_num):
                if arc[0] == path:
                    color = randomcolor(path)
                    figure = ax.plot([arc[1],arc[4]],[arc[2],arc[5]],[arc[3],arc[6]],c=color)
        #figure = ax.plot(X, Y, Z, c='r')
        ax.axis([0,length-1,0,width-1])
        my_x_ticks = np.arange(0,length,1)
        my_y_ticks = np.arange(0,width, 1)
        my_z_ticks = np.arange(0,3,1)
        ax.set_xticks(my_x_ticks)
        ax.set_yticks(my_y_ticks)
        ax.set_zticks(my_z_ticks)
        plt.tick_params(labelsize=6)
        #ax.set_title("Test3_Wiring")
        plt.savefig('P3_test2——5', dpi=300, bbox_inches = 'tight')
        plt.show()



except GurobiError as exception:
        print('Error code ' + str(exception.errno) + ": " + str(exception))
```