# （1）基于2023.9.25的两种方法，主要是方法二

**方法一：**

1.共用点锚定（点的坐标为超图得到的rank和order），其他点往共用点所在矩形框靠近移动——当前 tightened视图 已实现；

2.todo：找到合适的连线方式（d3库里一些连线的方式可试试看）

**方法二（较方法一紧致一些，但仅保留了纵向在超图中的相对位置和横向在子图中的同层节点的相对位置不变）：**

**坐标：（new\_order\_relative\_centerx, rank）**

1、计算共用点的横坐标X0；

2、所有点的纵坐标保持不变（相对于超图的rank），横坐标均往X0靠近：

2的具体实现：

A. 若同层级内仅有一个点，则横坐标设为X0；

B. 若同层级内有多个点，则保持同层级的order顺序不变【计算的时候注意 深拷贝 的问题】，均匀在X0两侧分布；

**关于连线的控制点：**

方法一：取source和target的中点【不定，后续再看】；

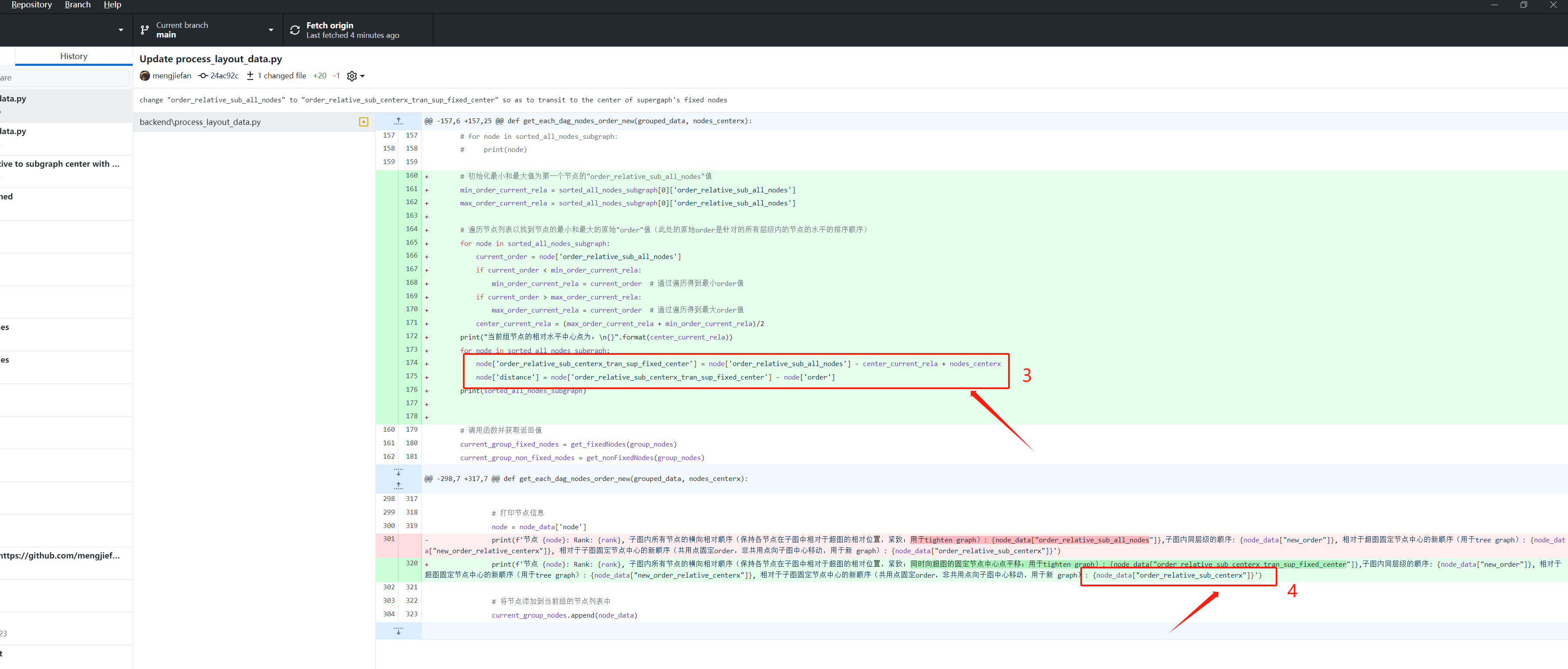
方法二：

1、如果source和target的横坐标不等，先计算两者的中点（(x1+x2)/2,(y1+y2)/2），判断两者中点的横坐标 (x1+x2)/2 是在center\_node的左侧还是右侧，如果是左侧，则控制点取【(x1+x2)/2 - 0.2\*|y2-y1| ,(y1+y2)/2）】，否则取【(x1+x2)/2 + 0.2\*|y2-y1| ,(y1+y2)/2）】，然后curveBasis;

2、如果source和target的横坐标相等（假设为 x），则控制点的纵坐标取两者纵坐标的中点（(y1+y2)/2），横坐标根据两者rank的差值，取 0.2\*rank\_diff来判断，即控制点的坐标为 【x + 0.2\*(y2-y1),(y1+y2)/2）】,其中【y2-y1 的正负正好也可以区分是否为反向的点】，然后curveBasis；

根据上述方法二得到的图为当前【Tree Subgraphs】

# （2）2023.10.9进一步改进上述9.25的方法二得到新的两种方法【均需加坐标，更方便比较】：



**坐标的映射：横向range为各子图的横向order【当前方法选择的特定order值】差值的最大值+2（2待定，主要是考虑两端的节点的移动向量横向所占空间）；纵向range为各子图的纵向rank差值的最大值；**

**以下两种方法均类似上述方法二直接拿坐标渲染即可；**

## 方法三-todo

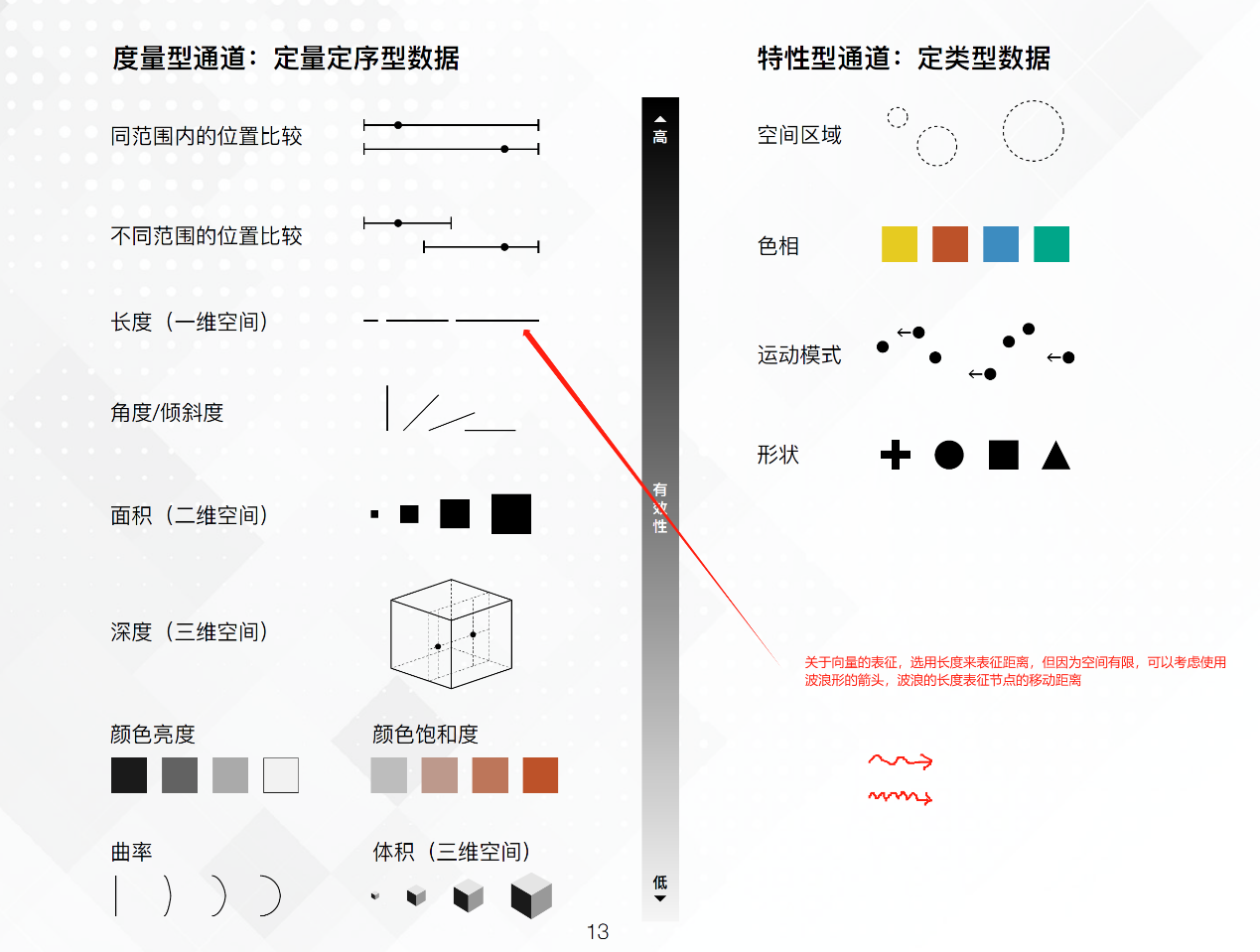
其实也算是优化上述方法一

1. **坐标：(order\_relative\_sub\_centerx\_tran\_sup\_fixed\_center,rank)**
2. 保持各节点的纵向位置不变，横向相对位置不变，紧致；并向超图的共用节点中心点平移；并据此得出各节点的移动距离distance；
3. 边的控制点同9.25；

## 方法四-todo

1. **坐标：(order\_relative\_sub\_centerx,rank)**
2. 保持超图的共用节点坐标不变(即为原order,rank)，子图中其他节点向**当前子图的共用节点的中心点**【与上述方法三中的sub\_centerx并不一致】移动靠近；
3. 边的控制点同9.25；

# （3）向量的表征-todo



[Mackinlay, Automating the Design of Graphical Presentations of Relational Information, 1986]引用文献

以上波浪箭头（波浪的个数表征距离远近d）也可以在箭头上加与其垂直的线，计算 d=(distance/0.5)，其中d为垂直线的个数，表征移动的距离远近

# （4）矩阵表征子图之间的相似性-todo

**内外两层：**

1. **内层**是节点相似性：（子图A的节点 ∩ 子图B的节点）/ （子图A的节点 ∪ 子图B的节点）
2. **外层**是子图整体的相似性（cal\_similarity.py）——计算欧氏距离：

邻接矩阵的计算：取所有的节点作为矩阵的大小，每个子图的矩阵的单元格取各自边的value值。

# （5）节点hover的时候显示 effect\_value – variants 条形图-todo

**【effect\_value取绝对值，按从大到小排序，并高亮当前子图中与当前节点有有向边相连的点】**

上述条形图的effect\_value的计算：dowhy中的source取各节点，target取当前hover的节点；

# （6）后端因果关系的进一步探索-todo