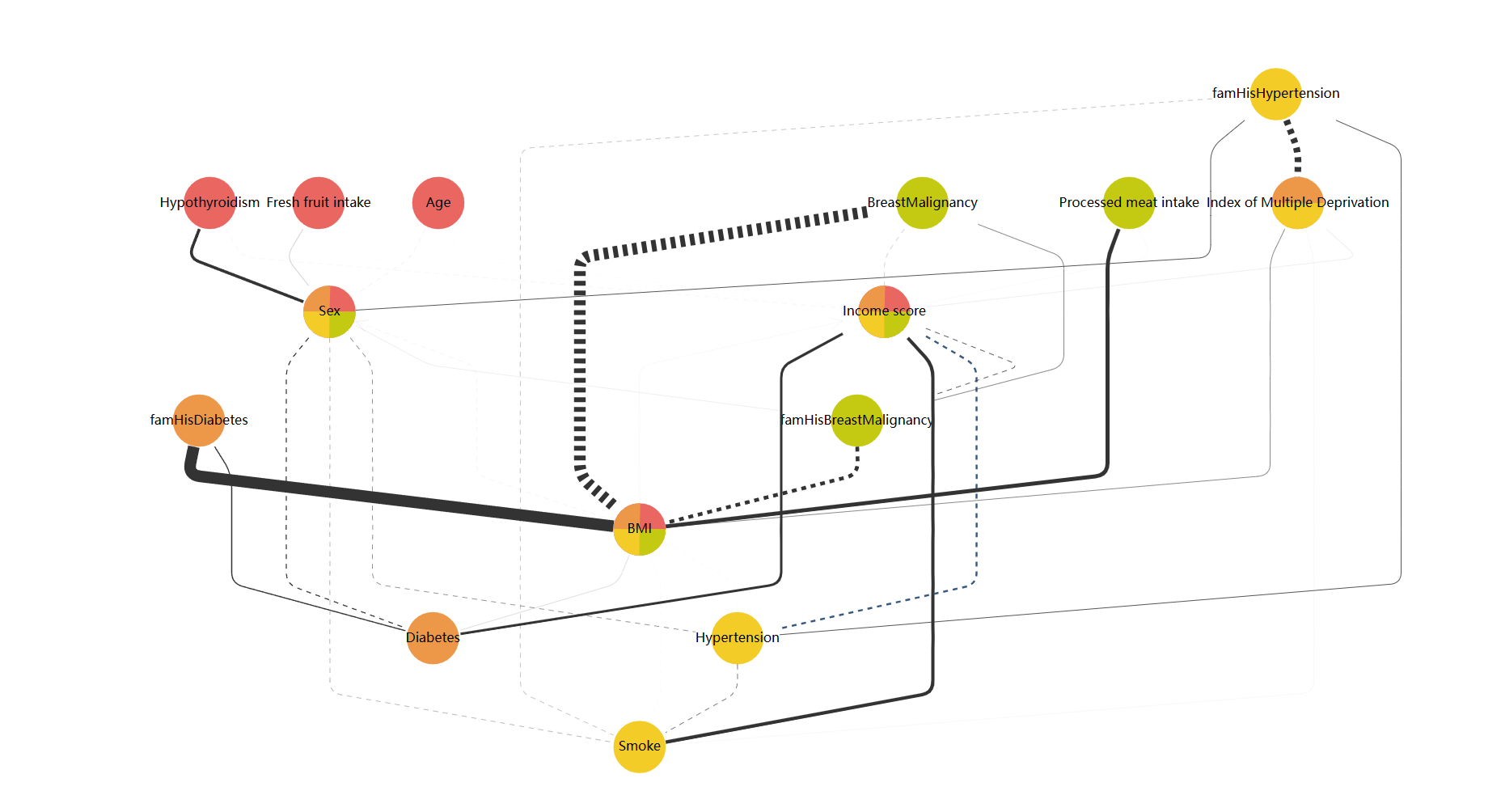
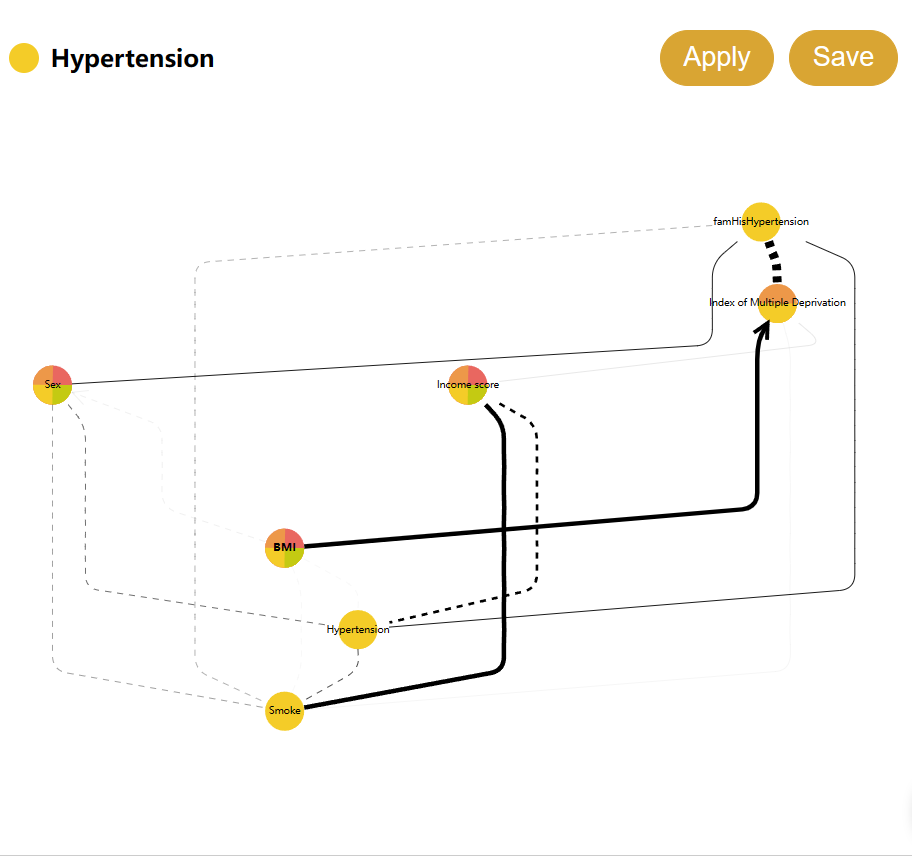
使用Dagre-d3工具（紧致树算法）获取Super Graph（上下方向），获得Super Graph中所有坐标（y坐标和x坐标可分别用来计算每个节点的rank属性和在该层内的节点排序），然后使用Jointjs重新绘图



Extracted Graph：

获取点的**绝对坐标**，绘制每个子图；每个子图根据绘图区域大小进行了合适的整体缩放。



Optimal Graph：

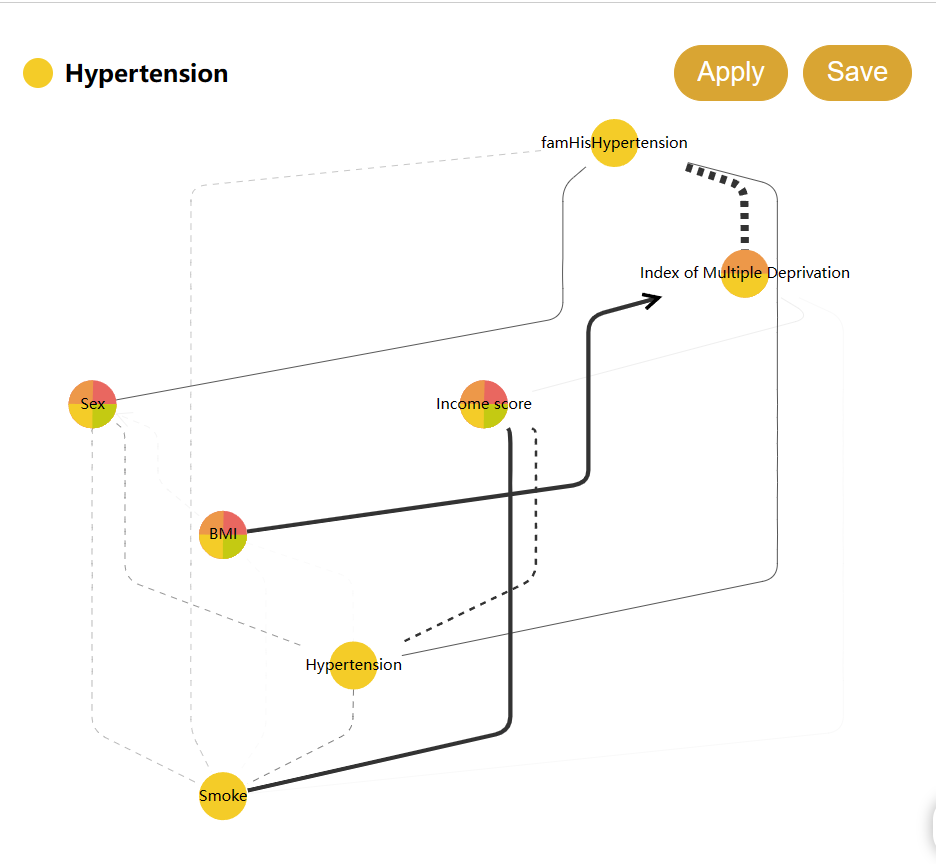
获取上述超图中得到的所有点的绝对坐标，并计算其原始layer和rank。

纵轴压缩：去掉子图中所有没有点的layer；【基于超图得到的绝对进而相对layer和rank坐标，对于单个子图来说不是最优，因为会受其他子图的影响（超图的绘制是综合考虑的所有的节点，故而对于单个子图来说，其布局不是最佳）】

横轴压缩: 重新获得每一层的点排序，但不改变原始的垂直对齐（比如下图中的Hypertension仍然在BMI和income score之间），去除不必要的rank【同上】

根据绘图区域大小计算x-gap和y-gap，计算出每个点的坐标【每个子图均不太一样，根据此步算出来的点的坐标充分利用的绘图区域，会导致每个子图的纵横比（此处应主要指的是x-gap和y-gap，甚至节点大小）不太一样】

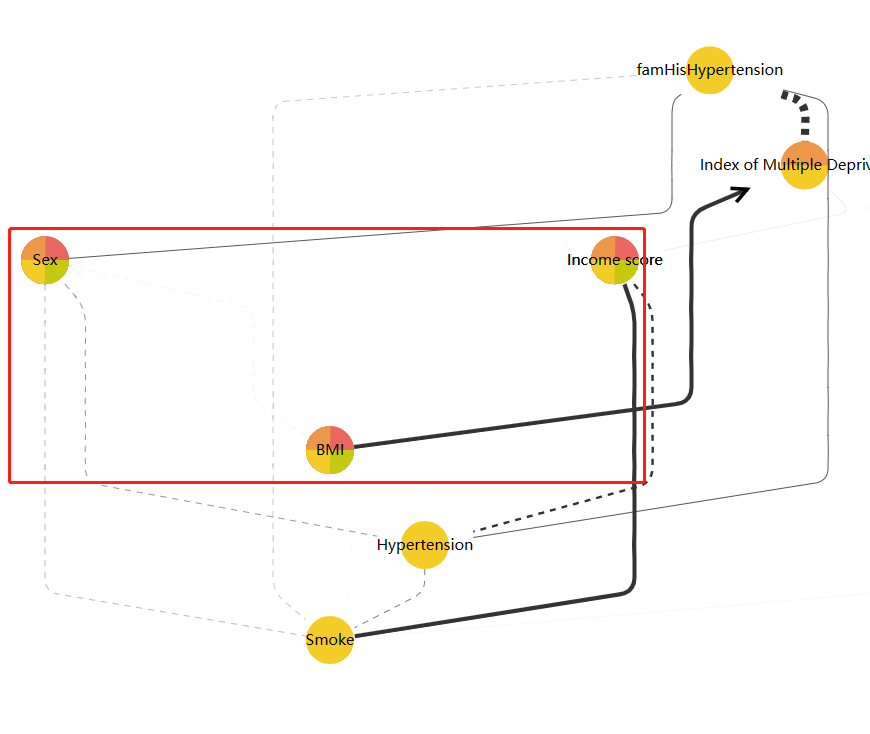
线的拐点按照在与最近的layer和rank的相对距离进行坐标的计算【进一步修改了assignNodeIntersects 函数内部的intersectRect 函数为 intersectCircle，以适配圆形节点】



Tightened Graph：

基本处理方案与Optimal类似，区别在于，不对共用点的layer和rank进行处理【是我们想要做的共用点（需注意：共用点指的是被**两个及以上**子图共用的点）锚定】。根据最初获得的layer和rank锚定共用点，共用点划定区域[Min\_X~Max\_X]（[Min\_Y~Max\_Y]），其余点x坐标在此范围内则不改变，y坐标在此范围内则不改变，在此范围之外的点同样去除不必要的rank（layer），即完成共用点之外的点的坐标压缩。【同上：基于超图得到的绝对进而相对坐标，对于单个子图来说不是最优，因为会受其他子图的影响（超图的绘制是综合考虑的所有的节点，故而对于单个子图来说，其布局不是最佳），**所以即便是在上述所谓划定的区域之内的点，也是需要紧致的**】

（下图中彩色方框为后来添加，不代表绘图效果）

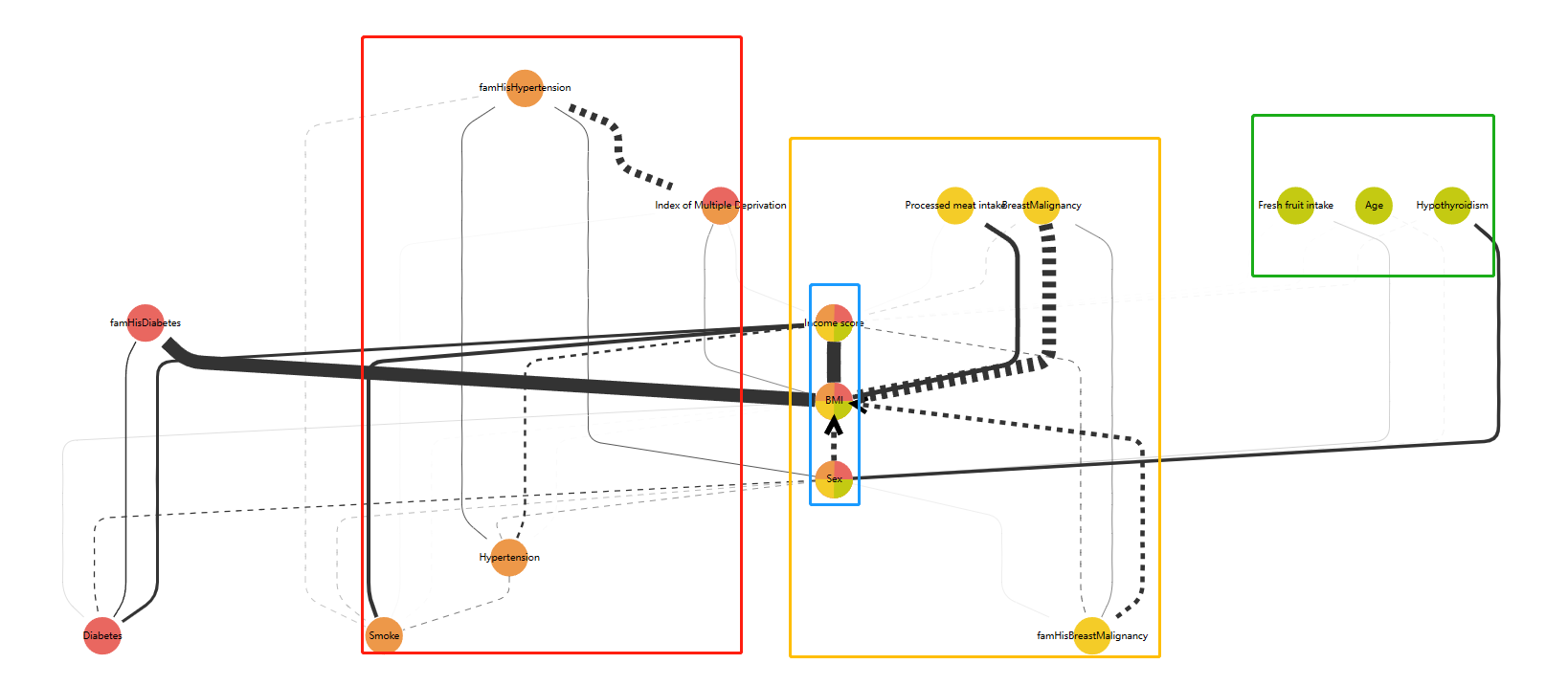


Original Graph：

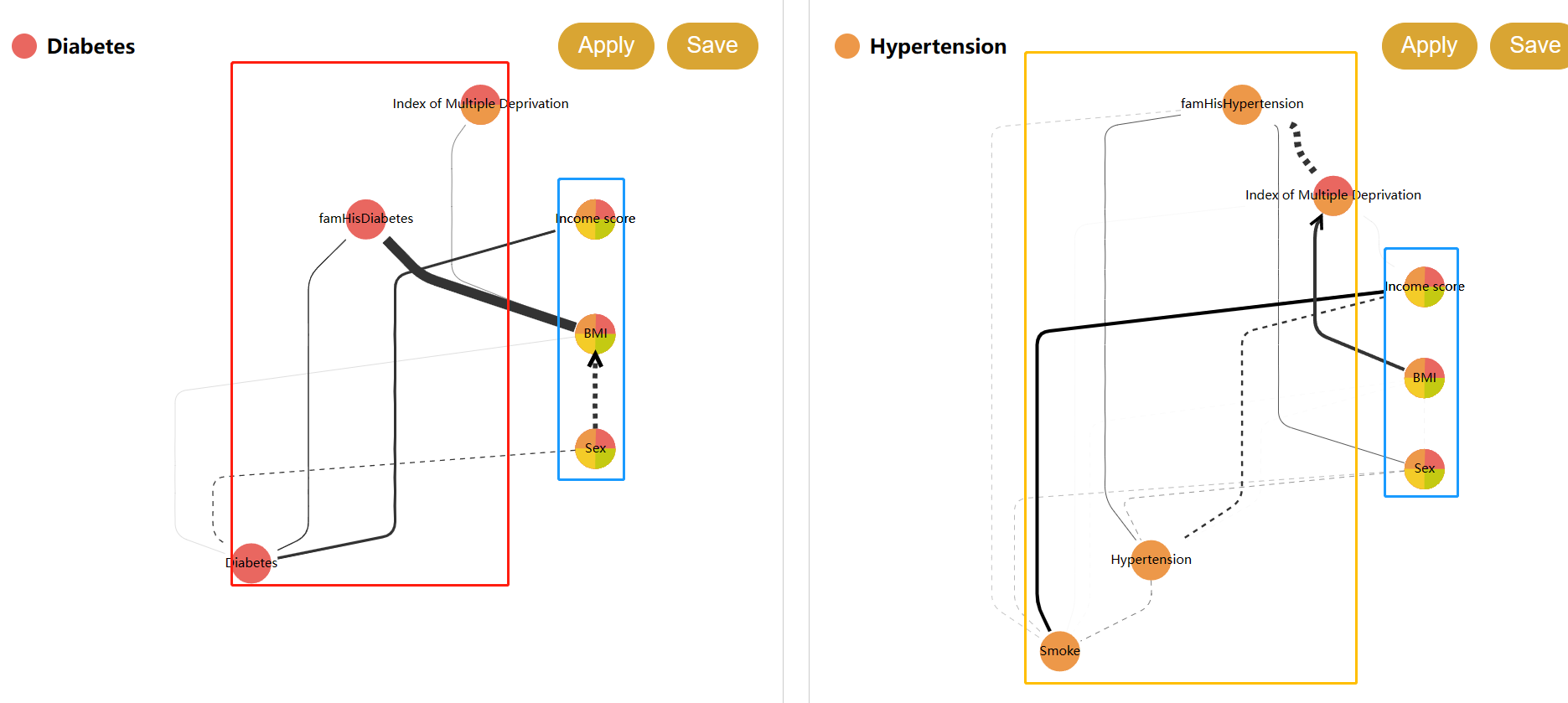
子图单独使用dagre计算并绘制

**新方案：**

使用Dagre Cluster绘制分层复合超图，即对点进行分组后再使用dagre绘制，以使每个子图包含的点位置紧密，同时共用点的位置紧密（图中彩色方框为后来添加，不代表绘图效果）



使用此方案，Tightened Graph能够有比较明显的效果（图中彩色方框为后来添加，不代表绘图效果）



TODO：不管是基于前述使用Dagre-d3工具（紧致树算法）获取的超图，还是新方案的Dagre Cluster绘制得到的分层复合超图，在比较多个子图时，均需要在锚定共用节点（只考虑共用节点，获取每个共用节点的layer和rank）的情况下，在每个子图内部继续使用算法以紧致其他非共用节点（包括最小化边交叉等），进而获取每个子图内所有节点的坐标（共用节点在考虑锚定时，横纵坐标均已限制了为整数（layer和rank），而非共用节点的横纵坐标根据算法，最终可能为小数），然后在视图中绘制（为直观显示锚点，需使用坐标系）。

注：dagre\_fixed.js已对会影响节点or边控制点坐标的函数做了修改

需**注意**，前述获取的边的控制点的坐标应该有问题（原代码的节点是用的矩形）