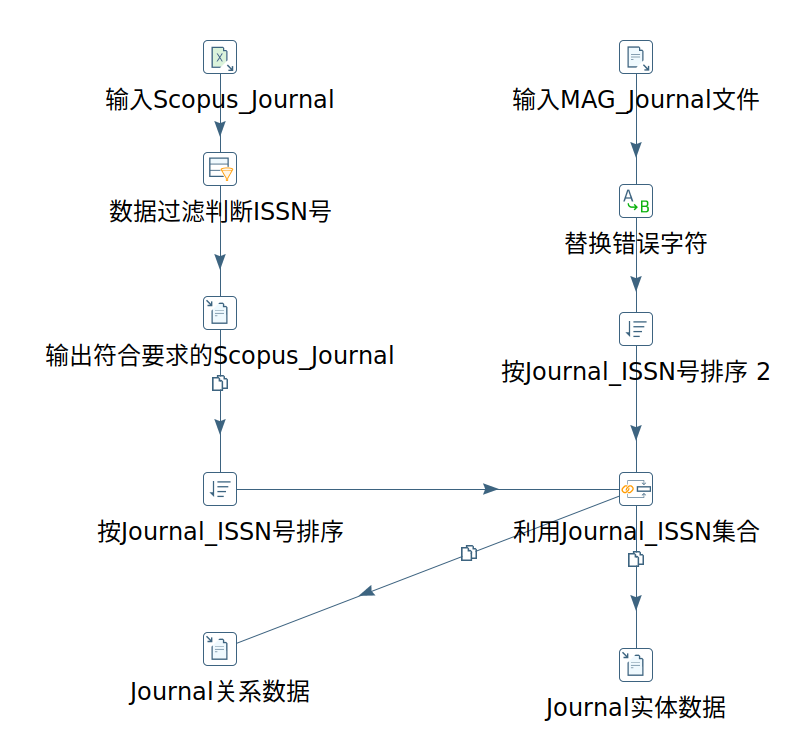


**1\_1kettle**

（1）期刊数据

期刊数据获取主要有三个主要步骤，第一是需要进行scopus期刊数据的清洗，得到可用来和MAG期刊数据融合的数据（即期刊信息完全的数据，特别是期刊ISSN号和期刊的学科分类情况）。第二是需要通过期刊ISSN号链接两个数据库数据，第三是数据得输出其中要注意的是图数据库的存储是需要将实体数据和关系数据分开存储的。该步骤见代码**1Journal.ktr**具体流程图如下所示。



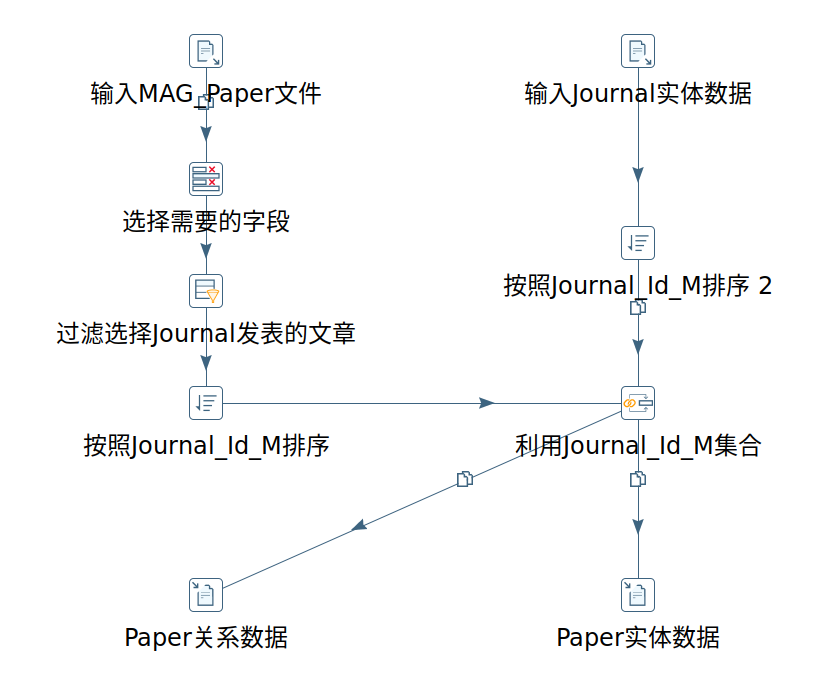
①scopus数据清洗。scopus数据为excel数据，其中有些期刊数据缺失项，需要先从进行处理再进行数据集成。输入数据为scopus的excel数据，然后进行条件判断，具体的条件为期刊的ISSN号、期刊所研究的子领域学科、期刊研究的学科不为空，且期刊的ISSN号不允许存在重复。经过条件判断之后的数据，就是可供使用的Scopus数据，最后将其输出保存。

②MAG期刊数据和Scopus期刊数据合并。输入为上一步中得到的Scopus数据和MAG的期刊数据即文件MAG\_Journal.txt文件，然后通过期刊唯一的标识ISSN属性内连接两数据，使用kettle合并数据时有个必须要做的事情就是按照关键字排序，最终会得到筛选出MAG和Scopus数据库中所共同拥有的期刊数据。

③数据分类输出。由于Scopus数据期刊的所有信息都存储在同一个文件中，即包含期刊的实体信息又包含期刊的关系信息，但在做图数据存储时，需要将实体数据和关系数据分开，故讲上一步所得到的期刊数据按照关系和实体分别输出，得到的文件为期刊的实体信息(Journal)、期刊研究的学科子领域(Journal→sub-Discipline)、期刊主研究学科(Journal→sub-Discipline)

（2）文献数据

数据获取主要有三个主要步骤，第一是需要进行MAG文献数据的清洗，得到可用来和期刊数据融合的数据（即文献信息为发表在期刊上的文献）。第二是需要通过期刊在MAG中的编号链接两个数据，筛选出满足要求可供使用的文献数据。该步骤见代码**2Paper.ktr**具体流程图如下所示。



①MAG文献数据清洗。首先进行错误筛选，避免数据量较大存在格式错误，筛选出Paper数据中没有按照MAG指定格式输出的数据，输出结果为空，即所有结果都满足指定格式。接着进行数据冗余处理，选择需要使用到的属性信息。数据干扰项排除，整理发现有些属性单个数据中包含逗号会影响csv的读取，所以将包含该符号的属性值相关数据进行替换，避免输出数据被读取错误。数据过滤，Paper包含3种发布类型（journal、patent、conference），该步骤为筛选出在Journal发布的Paper相关信息，即获取用于本课题研究的文献数据

②文献数据与期刊数据合并。数据集合，数据输入是上一步整理的文献数据及确定存入数据库中的期刊数据，使用两个文件中的共同属性Journal\_MAG\_Id，作为链接主键，得到最终的可使用的paper\_Journal关系，且将其中的连接属性更换为Journal\_ISSN。由于文献数据中包含文献实体数据（Paper）和文献和期刊间的发布关系数据（Paper→Journal），故也需分开输出。

**1\_2JAVA**

（1）去除数据冗余**1分割.java**

（2）去重计数由大到小排序**2去重计数.java**

（3）得到关系数据首先存入数据库neo4j **3\_1存入数据库.java**然后再获取关系数据**3\_2获取关系.java**

**2\_1TigerGraph**

TigerGraph除了可用GSQL进行图谱创建以外还可以使用图形化界面创建数据，主页面如图下所示，具体的功能为①Design Schema，用于查看和建立数据模式 ②Map Data to Graph，用于映射数据源，注意该步骤只连接数据源 ③Load Data，用于上一步中数据源的加载④Explore Graph，查看数据信息⑤Write Querise，使用GSQL编译需要完成的查询和计算任务。TigerGraph中模式层的建立由Design Schema步骤完成，数据存储由Map Data to Graph和Load Data共同完成。

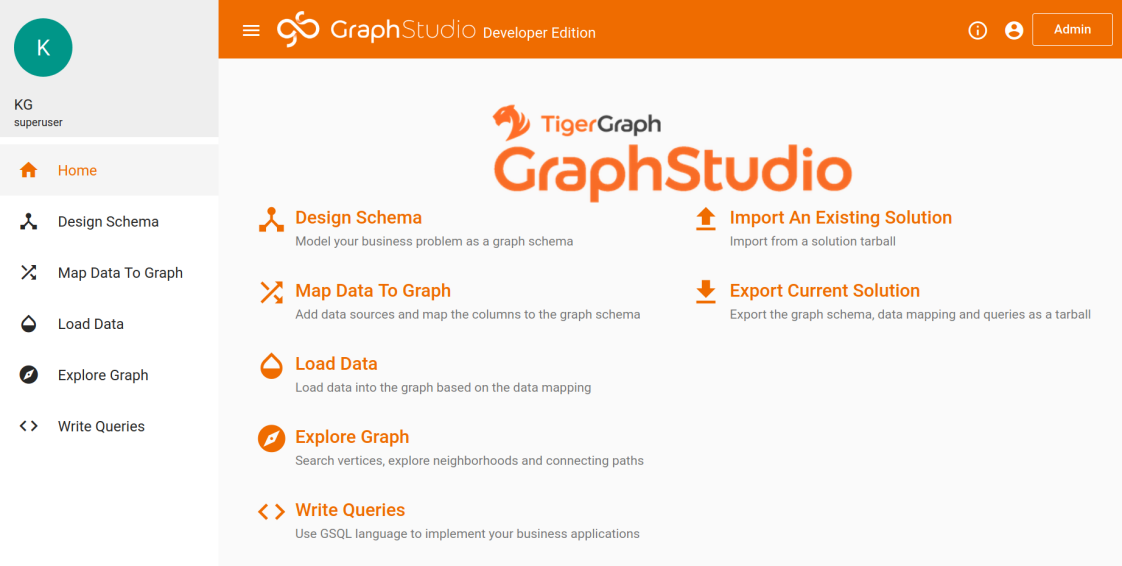


图4.13 TigerGraph可视化主界面图

（1）建立图模型

TigerGraph中，图模型由顶点集合和边集合定义，本文研究数据集是基于图的数据结构，学术实体和关系分别存储于顶点集合和边集合，每个学术实体或关系对应顶点或边集合中的文档，边集合文档中包含两个必需属性"From"和"To"，属性值分别表示关系的头部和尾部学术实体。

先建立图模式，在TigerGraph中模式层的建立方法有两种，一种是使用GSQL语言通过建立节点（代表实体）和边（代表关系），、建立模式层，以创建Paper节点和引文关系的GSQL语句为例，具体见**1KG.gsql**：

#使用CREATE VERTEX命令定义顶点类型

CREATE VERTEX Paper(

PRIMARY\_ID Paper\_Id INT,

Paper\_Title STRING, Paper\_doi STRING,

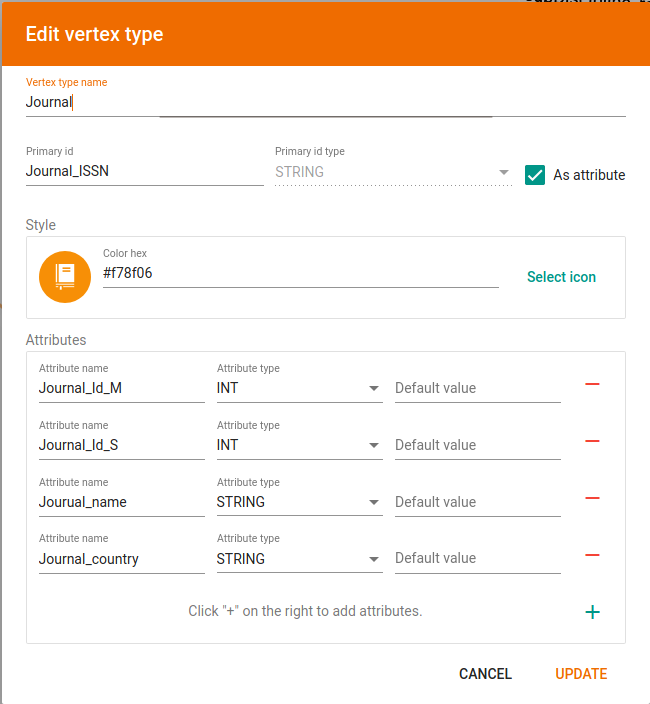
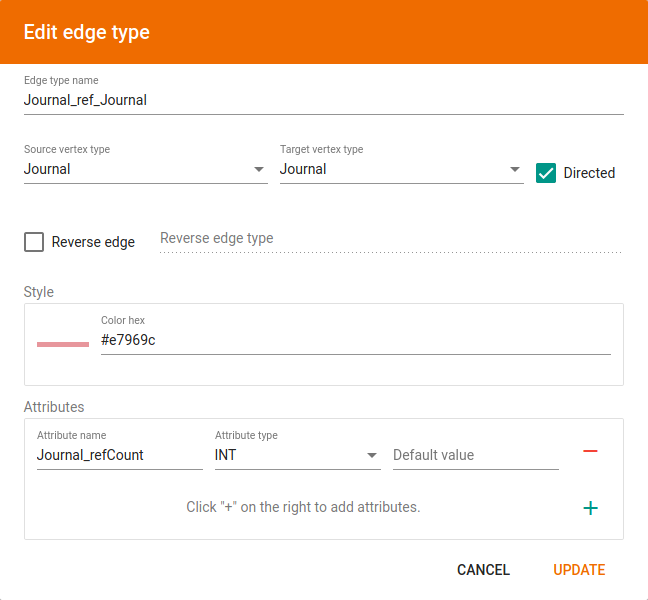
Paper\_rank INT,Paper\_year INT

) WITH STATS="OUTDEGREE\_BY\_EDGETYPE"

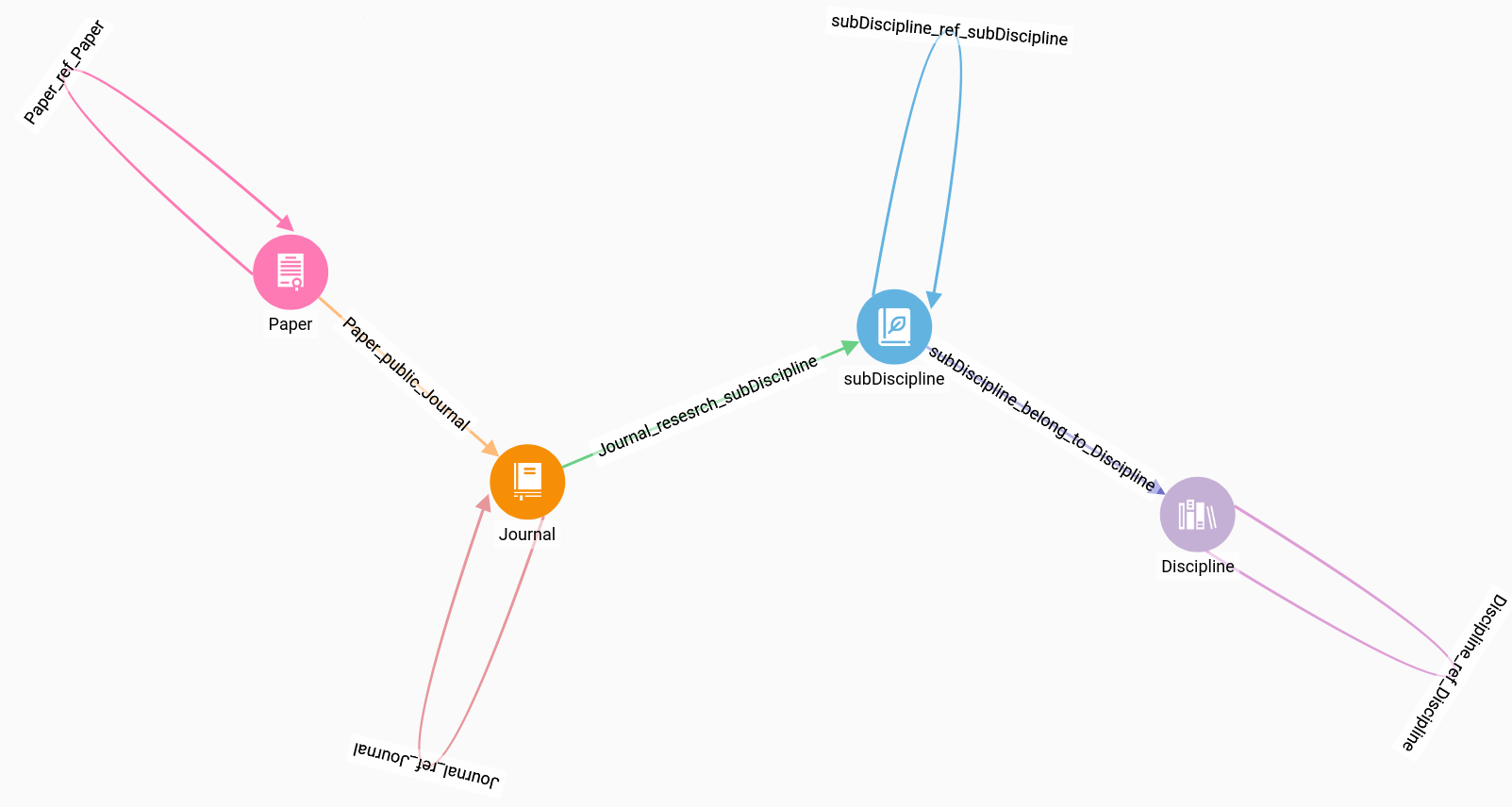
#使用CREATE DIRECTED/UNDIRECTED EDGE命令,创建边

CREATE DIRECTED EDGE Paper\_ref\_Paper(FROM Paper, TO Paper)

当然也可使用图形化界面创建顶点和边，具体如图下所示，其中左图为创建节点的示例，右图为创建边的示例。

无论是利用GSQL创建节点和边还是利用图形化界面，最终得到的模式层数据如下所示。



（2）加载数据

TigerGraph在图模型建立时已经完成了数据层的设计，而具体的数据映射和上传通过GSQL可直接上传，且上传速度极快，具体GSQL语句以Paper实体和Paper\_ref\_Paper关系为例如下所示，具体见**2load\_data.gsql**：

同样也可以使用图形化界面完成此任务，如图4.16所示，其中csv文件是在Map Data to Graph功能中上传的文件，并进行数据列与数据模式属性映射工作；Load Data功能是将映射好的数据上传至数据库中，图示FINSHED表示上传成功。

CREATE LOADING JOB load\_KG FOR GRAPH KG {

#首先加载本地数据

DEFINE FILENAME file\_Paper="./Paper.txt";

DEFINE FILENAME file\_Paper2Paper="./Paper2Paper.txt";

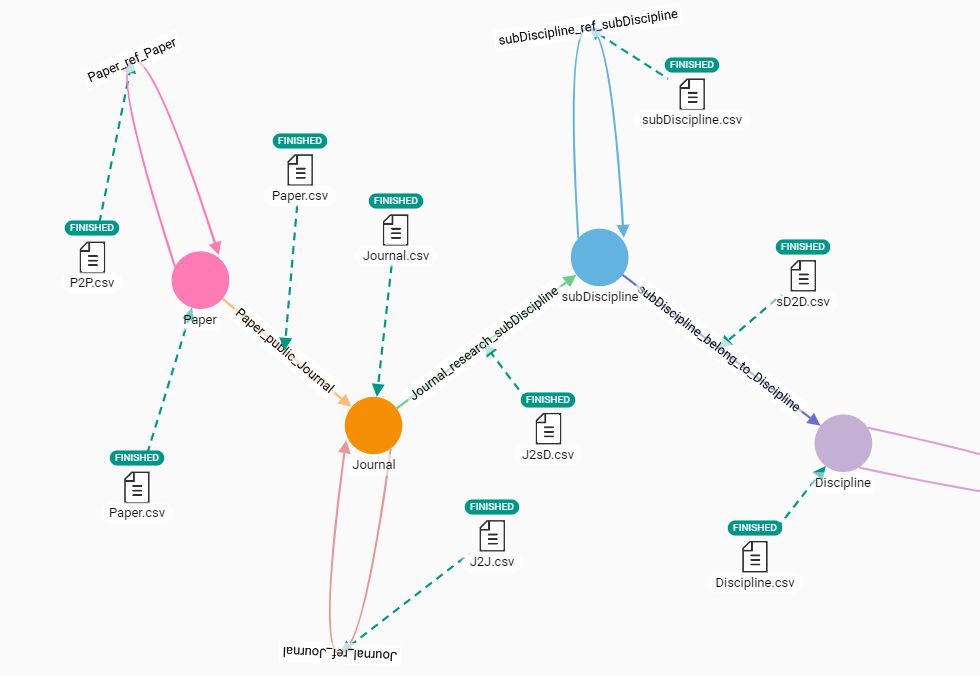
#然后映射数据,注意区分节点和关系

#header表示是否包含表头数据,separator为文件分隔符

LOAD file\_Paper TO VERTEX Paper VALUES ($0, $1, $2,$3, $4) USING header="true", separator=",";

LOAD file\_Paper2Paper TO EDGE Paper\_ref\_Paper VALUES ($0, $1) USING header="true", separator=",";

}



**2\_2Neo4j**

（1）存实体数据见**1存关系.java**

（2）存关系数据见**2存实体.java**

**3\_1社区划分**

第一个阶段：主要是不断地遍历网络中的节点，将每个节点划分到与其邻接的节点所在的社区中，尝试将单个节点加入能够使模块度提升至最大的社区中，直到所有节点都不再变化。首先将每个节点指定到唯一的一个社区，即假设每个节点为1个社区，N个节点，初始化N个社区，然后按顺序将节点在这些社区间进行移动。假设有一节点i，它有三个邻居节点j1、j2、j3，分别尝试将节点i移动到j1、j2、j3所在的社区，并计算相应的modularity变化值ΔQ，哪个变化值最大就将节点i移动到相应的社区中去（这里要求最大的modularity变化值要为正，如果变化值均为负，则节点i保持不动）。按照这个方法反复迭代，直到网络中任何节点的移动都不能再改善总的modularity值为止。

移动到一个社区C中所获得的模块化Q增益，其中Σtot是关联到C中节点链路权重的总和，ki是关联到节点i的链路权重总和，ki，in是从节点i连接到C中节点的链路总和，m是网络中所有链路权重的总和

第二个阶段：主要是将第一步划分出来的社区聚合成为一个点，即根据上一步生成的社区结构重新构造网络。重复以上的过程，直到网络结构不再改变为止。将第一个阶段得到的社区视为新的“节点”（一个社区对应一个），重新构造子图，两个新“节点”之间边的权值为相应两个社区之间各边权值的总和。如图5.3所示，如果第一个阶段得到的社区有以下四个，那么第二个阶段的任务就是将这四个社区分别看一个新的节点，然后将任意两个节点间所有连线的权重相加得到的和，作为这两个节点间连线的权重。

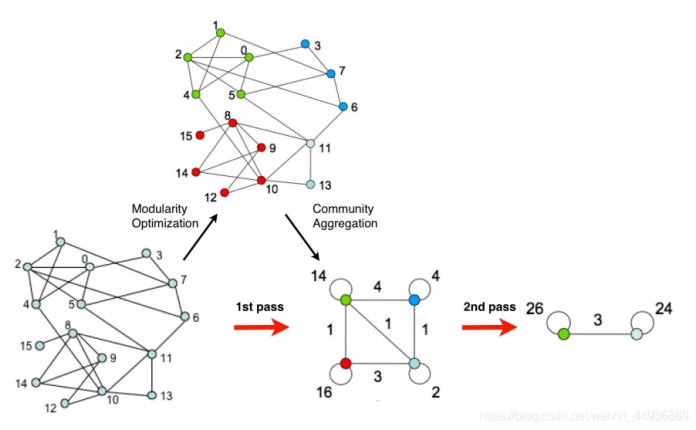


图5.3 社区聚合示例图

将上述两个阶段合起来称为一个pass，显然，这个pass可以继续下去。具体见**FU.py**

**3\_2聚类系数**

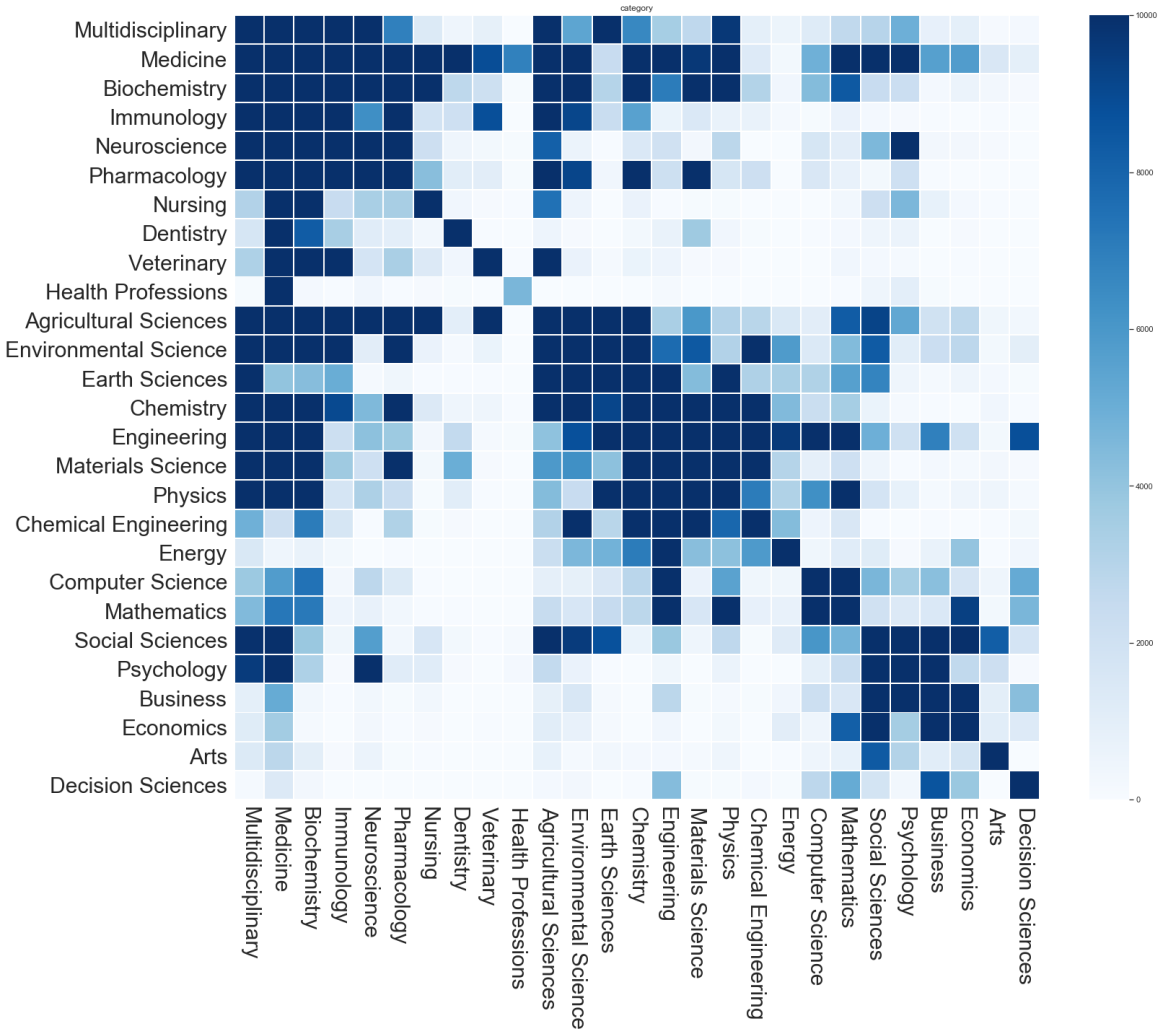
聚类系数也称为聚集系数，描述了网络中个体的邻居节点也互为邻居的可能性，是图中的点倾向于集聚程度的一种度量。聚类系数的目标是比较群组的聚合紧密程度与其能够达到的聚合紧密程度，三角形计数是聚类系数计算的基础。聚类系数分为全局聚类系数与局部聚类系数两种。全局聚类系数可以给出整个网络聚集程度的评估，其计算结果为网络中所有三元组（有两条（开三元组）或三条（闭三元组）无向边连接的三个结点）中封闭三元组的比例数。

局部聚类系数则可以测量图中每一个结点附近的紧密程度，旨在度量单个结点的嵌入性，。聚类系数实际上也是一个相对的评估指标，分母的意思是衡量某个节点和其邻节点如果构成完全图（即图上节点的两两都存在相互连接）的三角形数量，分子部分就是改节点实际所在的三角形的计数，二者的比值就是节点的聚类系数。在由向图中，图中节点i的局部聚类系数为：，其中{eij}为节点i相连边集合，ki为节点i相邻结点的数量。

聚类系数的大小影响了网络上的传播动力学，当其他参数不变时，节点聚类，数越大，则说明其越可能形成一个小世界，只与网络中的局部节点紧密联系，全局网路的传播越慢。具体见**聚类系数.py**

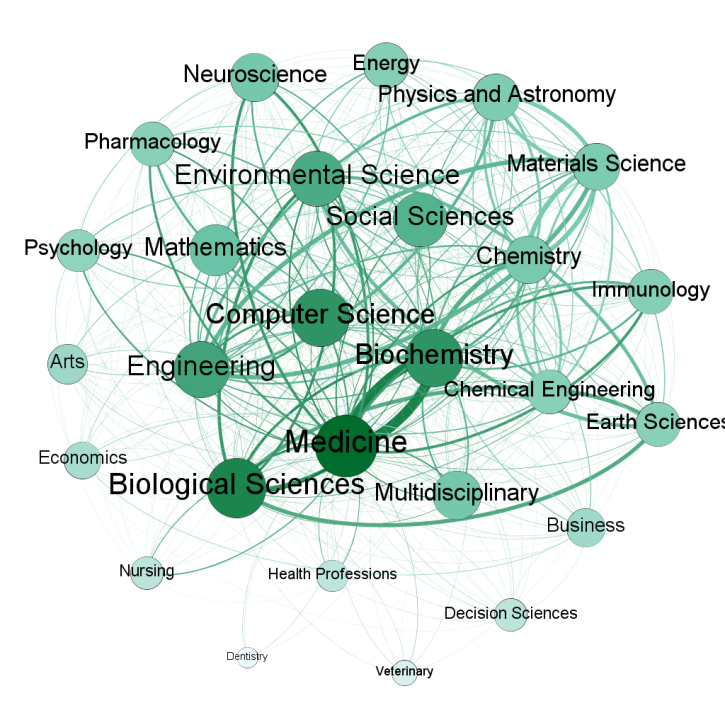
**4\_1学科引用矩阵**

效果图为



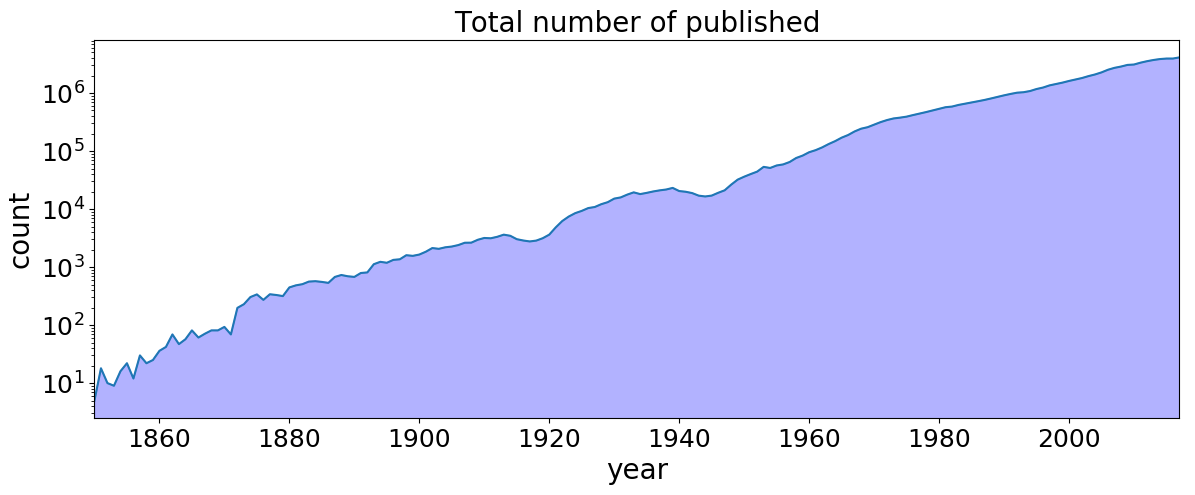
**4\_2学科引文网络**

效果图为



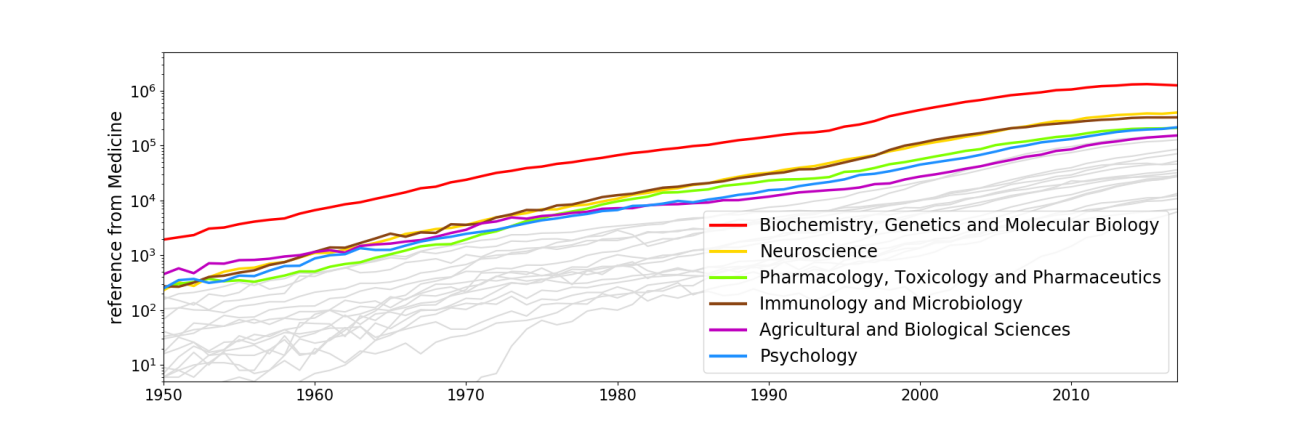
**4\_3统计图**

效果图为



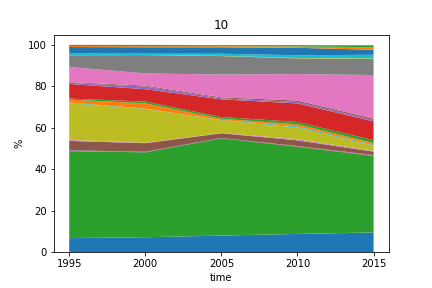
**4\_4折线图**

效果图为



**4\_4堆叠图**

效果图为

****

**4\_5可视化**

效果图为

****