**引言**

本数据分析报告旨在探索testdata.txt数据集内部的亲属关系结构，特别是检测其中潜在的数据异常，使用Python、Neo4j及Graphviz等工具对数据进行预处理、结构化分析，并展示数据图谱。以揭示图结构中的关键节点、最长路径和最大家族群体。

所有相关的代码、数据文件及生成的图形均已上传至 GitHub 仓库，地址为 <https://github.com/lizhihang1017/GraphDataSourceCode>。

**1. 数据概述**

本数据集包含约七万条记录，每条记录均以空格分隔，如：40609 40611 1 l m。其中，40609：第一个节点的ID，表示某个个体的唯一标识符。40611：第二个节点的ID，表示另一个个体的唯一标识符。m：关系类型的简写，表示母亲关系f和s分别表示父亲和配偶关系。

**2. 数据处理**

经过使用Pandas进行检测，数据集中未发现重复值和空值。然而，数据中存在一些不合理的关系，尤其在配偶关系上。例如，某节点的配偶竟然是其孙女，这显然不符合常理。此外，有些节点同时扮演了多个角色，例如一个节点既是另一节点的父亲，又是另一节点的母亲。数据中还存在多个父亲或多个母亲的情况，这可以理解为包括养父母在内的特殊家庭结构。

**2.1 配偶关系异常处理**

如下图2-1所示，存在配偶关系异常情况，节点5393与自己的孙子5439结成了配偶关系。

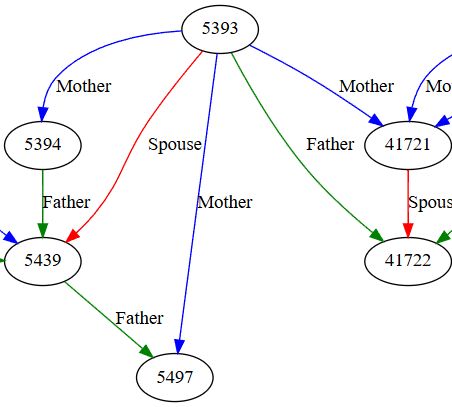


图2-1关系异常

处理方法，遍历每一条数据记录，构建家庭树，每个节点都有自己的子孙集合，以及配偶。根据家庭树，识别配偶关系异常情况，例如一个节点的配偶竟然是其子女或孙辈。对于检测到的无效配偶关系，将其从数据集中删除，以确保家庭结构的合理性，最终共清理掉3609条异常配偶关系数据。

主要代码：

if spouse is not None:

for child in children:

if child == spouse:

invalid\_entries.append((person, child, '配偶是子女'))

if spouse in children:

invalid\_entries.append((person, spouse, '子女是配偶'))

for child in children:

if child in family\_tree:

grandchildren = family\_tree[child]['children']

if spouse in grandchildren:

invalid\_entries.append((person, spouse, '配偶是孙子孙女'))

**2.2 多角色节点处理**

如上图2-1所示，节点5393扮演了多个角色，节点5393是节点5394的母亲的同时又是节点41722的父亲。

对于一个节点可能同时作为父亲和母亲的情况，采取“少数服从多数”的处理办法。如果某节点被标记为父亲的记录多于被标记为母亲的记录，则将母亲标记修改为父亲。反之亦然。如果父亲和母亲标记的数量相同，则按照首次出现的角色标记为准——即若父亲角色先出现，则将所有母亲标记修改为父亲；若母亲角色先出现，则修改为母亲。最终共修改数据3796条。

主要代码：

if father\_count > mother\_count and mother\_count != 0:

for mother in roles['mother\_nodes']:

modified\_relationships.append((person, mother, 'f'))

elif mother\_count > father\_count and father\_count != 0:

for father in roles['father\_nodes']:

modified\_relationships.append((person, father, 'm'))

elif father\_count == mother\_count and father\_count > 0:

first\_father = roles['father\_nodes'][0] if roles['father\_nodes'] else None

first\_mother = roles['mother\_nodes'][0] if roles['mother\_nodes'] else None

if first\_father:

for mother in roles['mother\_nodes']:

modified\_relationships.append((person, mother, 'f'))

elif first\_mother:

for father in roles['father\_nodes']:

modified\_relationships.append((person, father, 'm'))

**3. 数据分析**

为了更有效地分析和可视化数据，利用python脚本将所有记录已导入到Neo4j图数据库中。数据结构如下：

节点类型：每个个体被表示为Person节点，节点的ID作为属性存储。

关系类型：根据关系的类型（m、f、s），建立以下关系：MOTHER\_OF、FATHER\_OF 和SPOUSE\_OF。

**3.1度数最大节点分析**

在Neo4j中，通过使用Cypher语句可以轻松查询到度最大节点的ID为“49244”，为方便起见，将其称为MAX\_NODE\_ID。在Neo4j中展示 MAX\_NODE\_ID的关系图，如图 3-1 所示。根据图示可知，MAX\_NODE\_ID节点为女性，且具有配偶，并且共有七个孩子。然而，令人困惑的是，MAX\_NODE\_ID被标记为拥有七个母亲，这显然不符合常理。

为进一步探究，查询MAX\_NODE\_ID的父亲节点，但未能找到相关信息。同时，在查询MAX\_NODE\_ID七个母亲的配偶时，也没有获得任何结果。这种情况可能是由于数据的不完整性所导致。

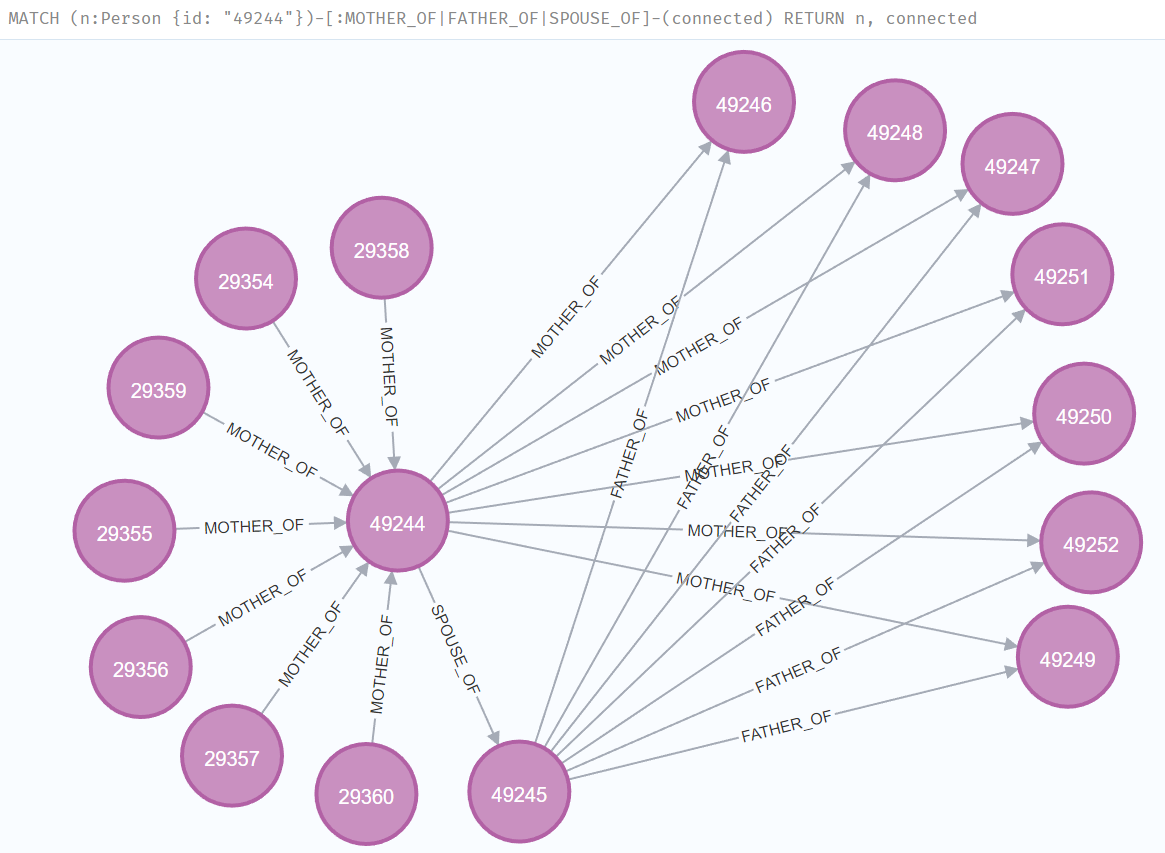


图3-1 MAX\_NODE\_ID节点关系图

**3.2 最长路径分析**

在研究图的最长路径时，可以采用两种方法：

方法一： 在Neo4j中使用Cypher语句，直接获取最长路径及所经过的节点。

方法二：使用Python编程语言和NetworkX库，通过动态规划结合拓扑排序计算有向图中的最长路径。首先对图进行拓扑排序，以确保节点按依赖关系顺序处理。算法将每个节点的最长路径长度初始化为0，并记录其前驱节点。随后，依次遍历排序后的节点，逐步更新每个后继节点的最长路径长度，确保始终保持路径最大。最后，通过选择路径长度最大的终点，回溯前驱节点还原出最长路径，并记录其长度。

如下图3-2所示，这条路径从节点9687到节点23555，路径长度为14，反映了图中距离最远的两个节点之间的关系链。

若将数据图构建成无向图去查找最长路径，则需要巨大的算力，初步估算以我的电脑算力，大约需要300天可以最终求得最长路径。

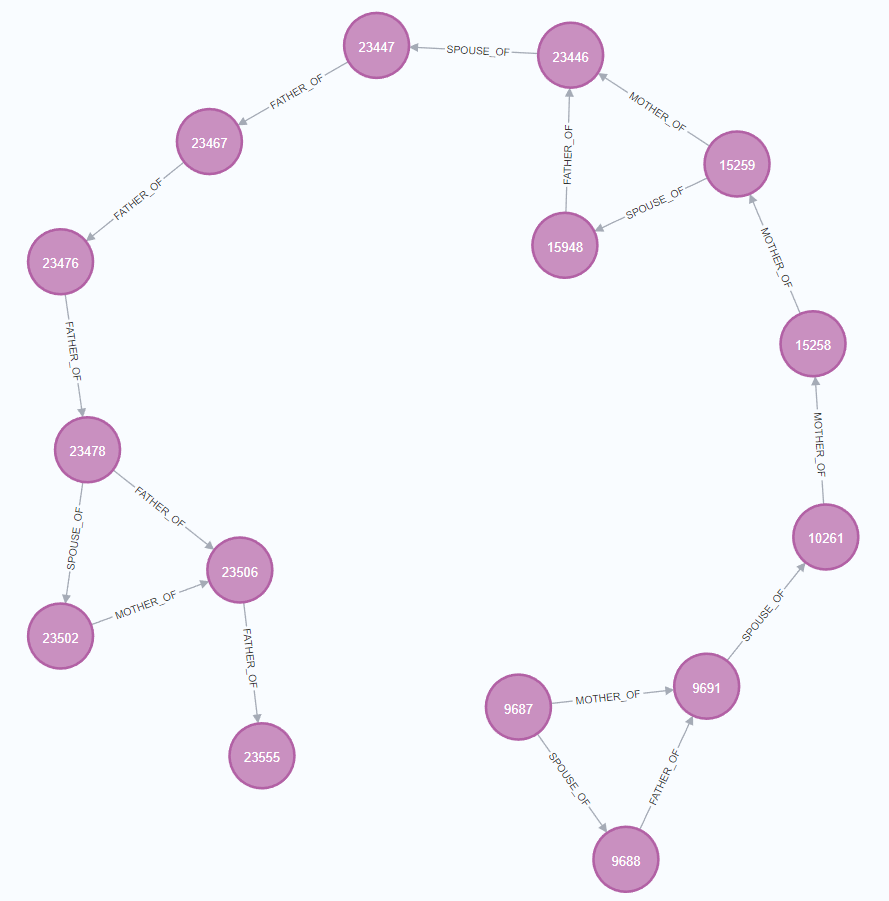


图3-2最长路径关系图

**3.3 孤立节点分析**

使用Neo4j和NetworkX对数据集进行孤立节点检测。在Neo4j中，通过执行MATCH (n) WHERE NOT (n)-[]-() RETURN n的Cypher查询，验证数据集中是否存在无关联的独立节点。结果表明，数据集中不存在任何孤立节点。为进一步验证结果的可靠性，在NetworkX中构建了同样的网络图，并使用nx.isolates(G) 方法检测孤立节点。分析结果再次证实，数据集中的每一个节点均具备至少一条与其他节点的连接关系。

**3.4 家族族谱分析**

使用社区检测算法来识别数据集中的不同家庭关系群体，从而找到规模最大的家族族谱。通过该算法，根据家庭成员间的亲属关系，将数据划分为多个子图社区，从中识别出最大社区，即包含最多成员且关系最紧密的家族结构。

为了对这一规模最大的家族族谱进行可视化展示，尝试了多种方法：首先通过Python导入Neo4j进行可视化，但效果不甚理想。随后使用GraphXR连接 Neo4j数据库，选择树形关系的3D可视化展示，效果最佳，但由于受到企业版限制，无法导出图片。因此，最终选Graphviz生成并导出 family\_ancestry\_horizontal.svg 文件，实现了该家族关系的可视化。

主要代码：

dot = Digraph(comment='Family Ancestry', format='svg')

dot.attr(rankdir='LR')

for \_, row in family\_edges.iterrows():

person1\_id = str(row['person1\_id'])

person2\_id = str(row['person2\_id'])

relation = row['relation\_type']

dot.node(person1\_id, person1\_id)

dot.node(person2\_id, person2\_id)

if relation == 'm':

dot.edge(person1\_id, person2\_id, label='mother', color='blue')

elif relation == 'f':

dot.edge(person1\_id, person2\_id, label='father', color='green')

elif relation == 's':

dot.edge(person1\_id, person2\_id, label='Spouse', color='red')