**开源软件开发与社区治理报告**

组员：张雯怡（51265903051） 胡思劼（51265903082）

任务类型：**OpenPerf**

任务名称：[Open source repository collaboration network and npm artifact library dependency network mapping dataset](https://github.com/X-lab2017/open-perf/issues/62)

**一、任务描述与目标**

本任务将构建一个数据集，其旨在表达npm包的依赖网络与开源仓库协作网络之间的映射关系。这一数据集的构建目的是：由于个人贡献和仓库名称更改导致npm注册表中元数据不完整或过时，该数据集能够帮助开发者重新从一个仓库出发，映射到npm依赖网络的相应的npm包所对应的节点，从而找回属于该仓库所需的npm依赖包。

我们需要构建开源仓库协作网络和npm工件库依赖网络，其信息如下：

* **开源仓库协作网络：**

**节点：**代表一个仓库（repo）。

**属性：**包括贡献数量、贡献的性质（代码、文档等）以及协作持续时间等指标。

**边：**代表仓库间的协作关系（同一开发者在两个repo均有贡献）

* **npm工件库依赖网络：**

**节点：**代表单独的npm包。

**属性：**包括版本号、更新频率和受欢迎程度指标（下载量、描述）。

**边：**代表依赖链接，即一个包依赖于另一个包。

由于网络结构的复杂性，两个网络无法完全映射，但两个网络的子集可以具有相应的关系，可以根据npm包信息中的repo\_url字段进行映射。

npm包以及npm依赖数据我们是直接从X-lab开源社区所提供的clickhouse数据库中直接查询提取的，数据总量在4亿左右，我们应当做适当的样本筛选，减少数据量；协作网络的数据则是利用python的requests模块，通过公共API从GitHub上收集。

构建完网络后，我们可以定量分析网络的指标，如度数、聚类系数、平均路径长度、直径、中心性、密度、模块性、连通分量等；也可以定性分析网络结构，比如可视化分析。

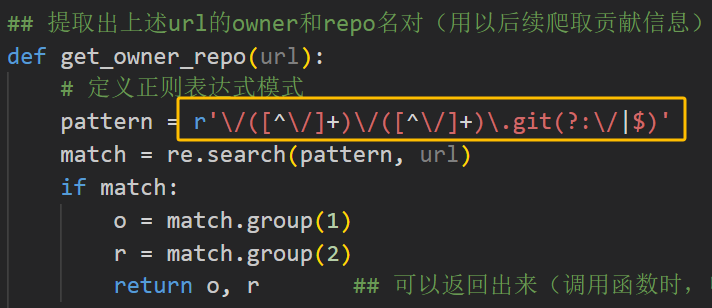
此外，两个网络映射的分析是本任务很重要的部分，也可以做可视化分析，通过检查依赖链及其对软件可靠性的影响来研究软件生态系统的弹性与趋势。

**二、开源协作数据的构建及其网络构建**

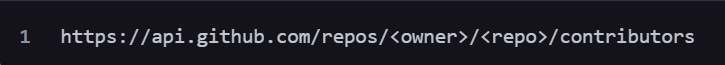
首先，我们要从npm包的repo字段中获得到所要处理的这些开源仓库的url（爬取对象），而在利用API爬取数据时，还需要给出仓库的owner和repo名，这两者蕴含在了url中，如下图所示。



owner和repo名的提取需要用正则表达式作用于url进行提取，我们所想到的正则表达式的规则是：在url中找到第一个“xxx/yyy.git”的形式，然后提取出xxx和yyy，他们分别是owner和repo名。故正则表达式应为：



随后，便可以使用GitHub的API来构造请求url，其格式如下：



将<owner>替换为我们要处理的仓库所有者，将<repo>替换为仓库名称。通过调用python的requests库的函数来接收并解析响应，GitHub的服务器会响应你的请求，并返回一个JSON格式的数据包，其中包含了仓库的贡献者信息；调用requests的接口来解析JSON数据包以获取贡献者的名字，而后写入pandas的DataFrame（使用pandas是为了方便），最后持久化到csv文件，写入本地磁盘，保存好数据以便后续使用。

这一部分较为麻烦的是，爬取的请求频率是有限制的，为了进行更大量请求，我们在headers中写上了token进行访问认证；处理时仍然可能出现异常，我们的策略是“有限的重试”，重试到一定上限次数便停止，直接爬取下一个仓库的贡献者信息。

我们将npm包中提到的仓库全都保存至一个list中，然后对于list中的每个仓库，都做上述操作，最终形成的数据形式如下：

**repo\_name | contributors**

levelDB [‘Alice’,’Bob’,’Jack’] (后者是list类型)

RockDB [‘Alice’,’Mike’]

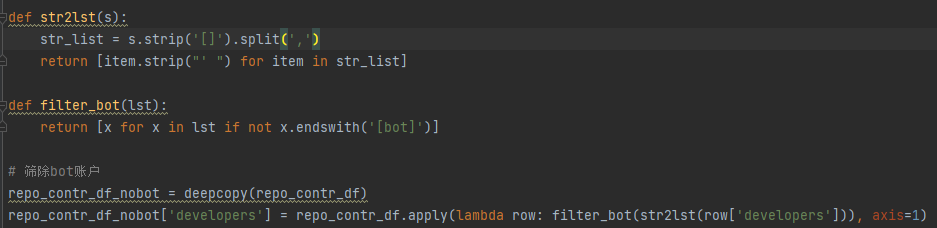
cnnGtb [‘Jack’, ‘Jerry’, ‘Eric’, ‘Jimm’, ‘Bob’]

完成数据收集后，即可使用网络分析工具（我们所使用的是networkx库）来构建和可视化开源协作网络。

网络上的各个节点代表各个仓库，那么该网络想展示的是各个**仓库之间的协作关系**，我们将这种协作定义为：若某个贡献者Alice在开源仓库repo\_1和repo\_2均有贡献，那么repo\_1和repo\_2这两个仓库具有协作关系。换句话说，如果repo\_1和repo\_2**具有至少一个相同的贡献者**，那么它们就具有协作关系，可以在两个仓库间建立一条无向边。

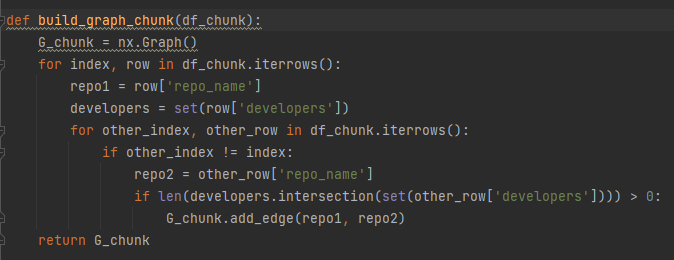
因此，上述代码逻辑可以写成：利用两个for循环两两比对repo的贡献者，如果两者的贡献者列表**取交集**以后所形成的集合的大小，那么就有共同的贡献者，可以连边。由于时间复杂度是，这个程序跑得很慢，所以考虑使用**并行化处理库**。

由于爬取来的数据均为string字符串的数据类型，所以我们要先对数据进行预处理，将贡献者列表从字符串格式转换为Python列表，并筛除了以[bot]结尾的bot账户，确保网络中仅包含人类贡献者。

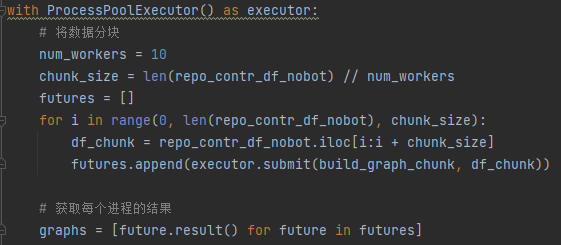


我们采用了**并行处理**的方法来加速网络构建过程，具体操作流程如下：

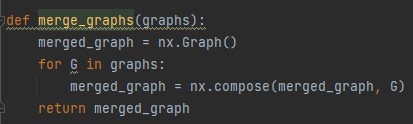
* 将数据集分割为多个子集，每个子集对应一个子图；
* 遍历数据子集构建单个子图，为具有共同贡献者的仓库对添加边；



* 利用concurrent.futures.ProcessPoolExecutor，我们实现了多进程并行处理，每个子集由一个独立的进程处理，以构建其对应的子图；

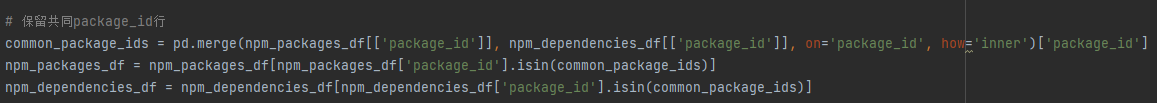


* 所有子图构建完成后，合并这些子图，形成一个完整的网络。



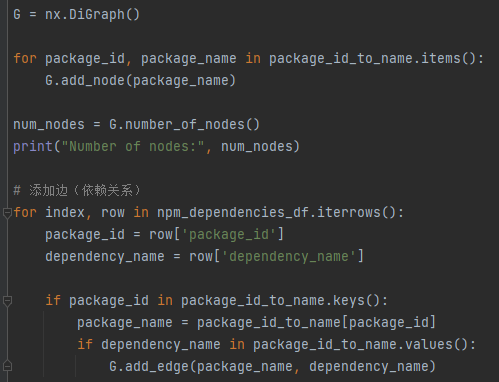
**三、npm依赖网络的构建**

这部分数据不需要爬取，来源于ClickHouse数据库，通过clickhouse\_driver库连接并查询数据，从而可以直接获取比较规整的数据。我们选择了两个主要表：npm\_records和npm\_dependencies，分别包含npm包的基本信息和依赖信息。使用pandas库读取查询结果，并将其转换为DataFrame对象。我们筛选了同时存在于npm\_records和npm\_dependencies表中的package\_id，确保了数据的一致性。



使用networkx库创建了一个有向图G，用于表示npm包之间的依赖关系，建图流程：

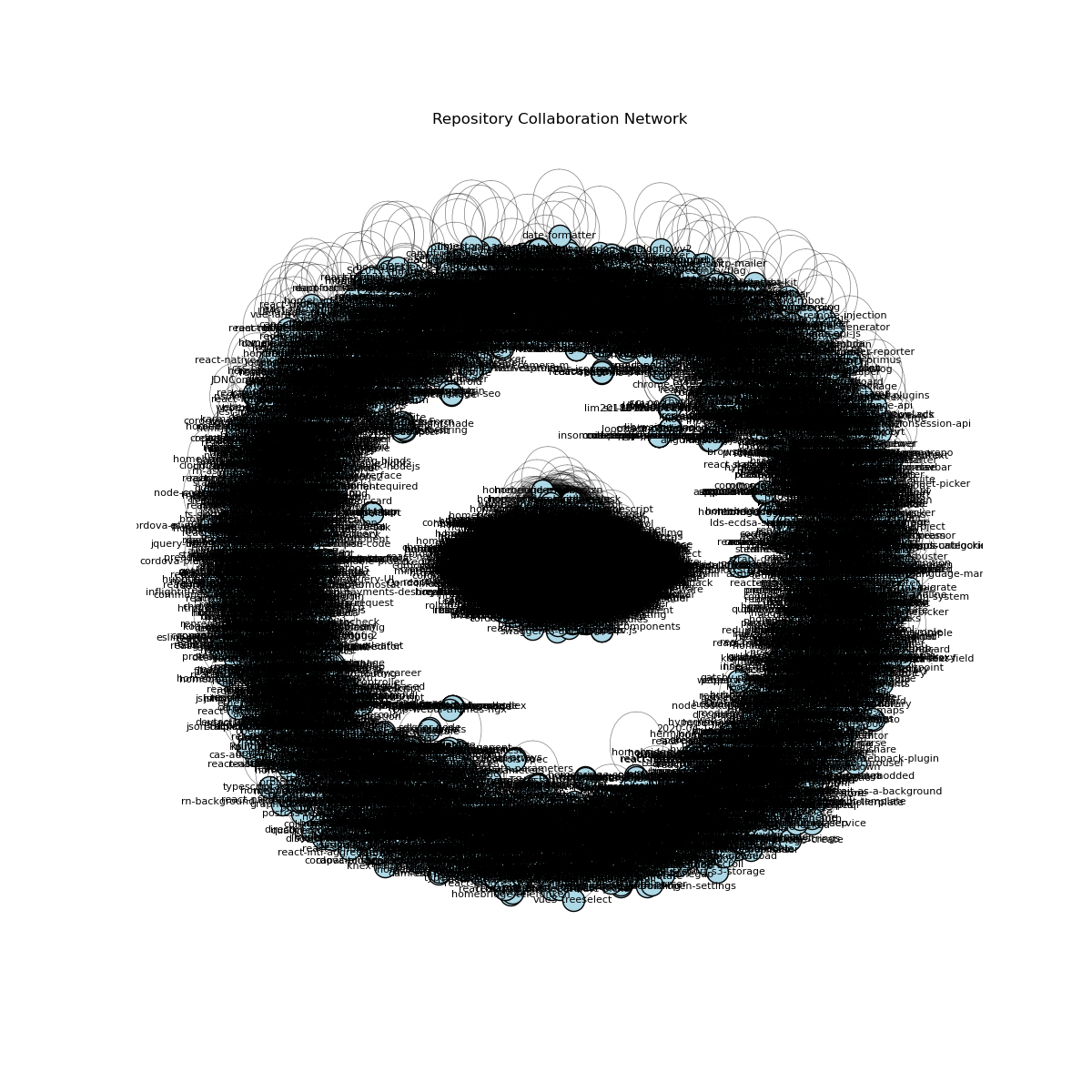
* 为每个npm包创建节点，并将包名映射到包ID；
* 遍历依赖信息DataFrame，为存在依赖关系的包对添加边；
* 移除了没有依赖关系（既不是依赖者也不是被依赖者）的孤立节点，以优化该网络的结构。



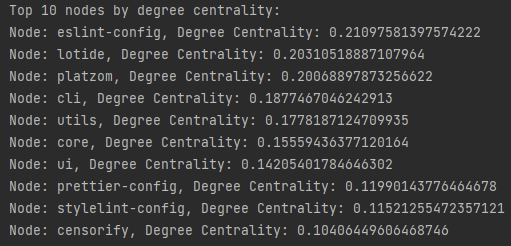
**四、网络特征的分析**

建立完上述G后，我们定量地对它们进行一些指标的度量，包括**节点数、边数、节点的度中心性、网络的聚类系数**（用以描述局部连接紧密度）、**平均路径长度和直径**（用以描述网络整体连接性），从而能够更好地理解图本身的基本特征。两个网络的分析如下图所示：

**开源协作网络：**

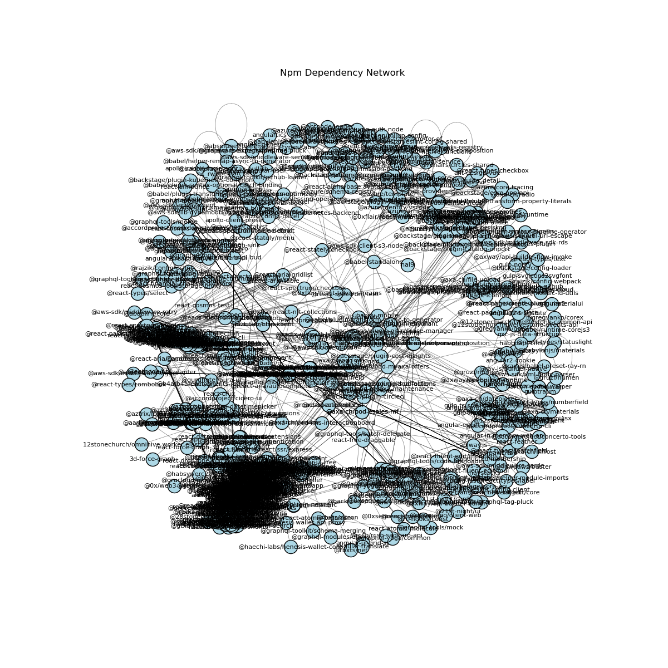


* 节点数：41802 边数：4079123
* 节点的度中心性top10：

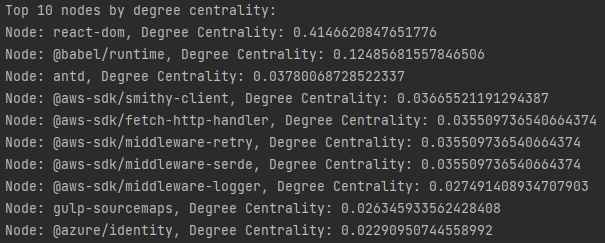


* 网络的平均聚类系数（局部连接紧密度）：0.5051
* 平均路径长度和直径（网络整体连接性）：非连通图，无法计算

**Npm依赖网络：**



* 节点数：874 边数：1351
* 节点的度中心性top10：

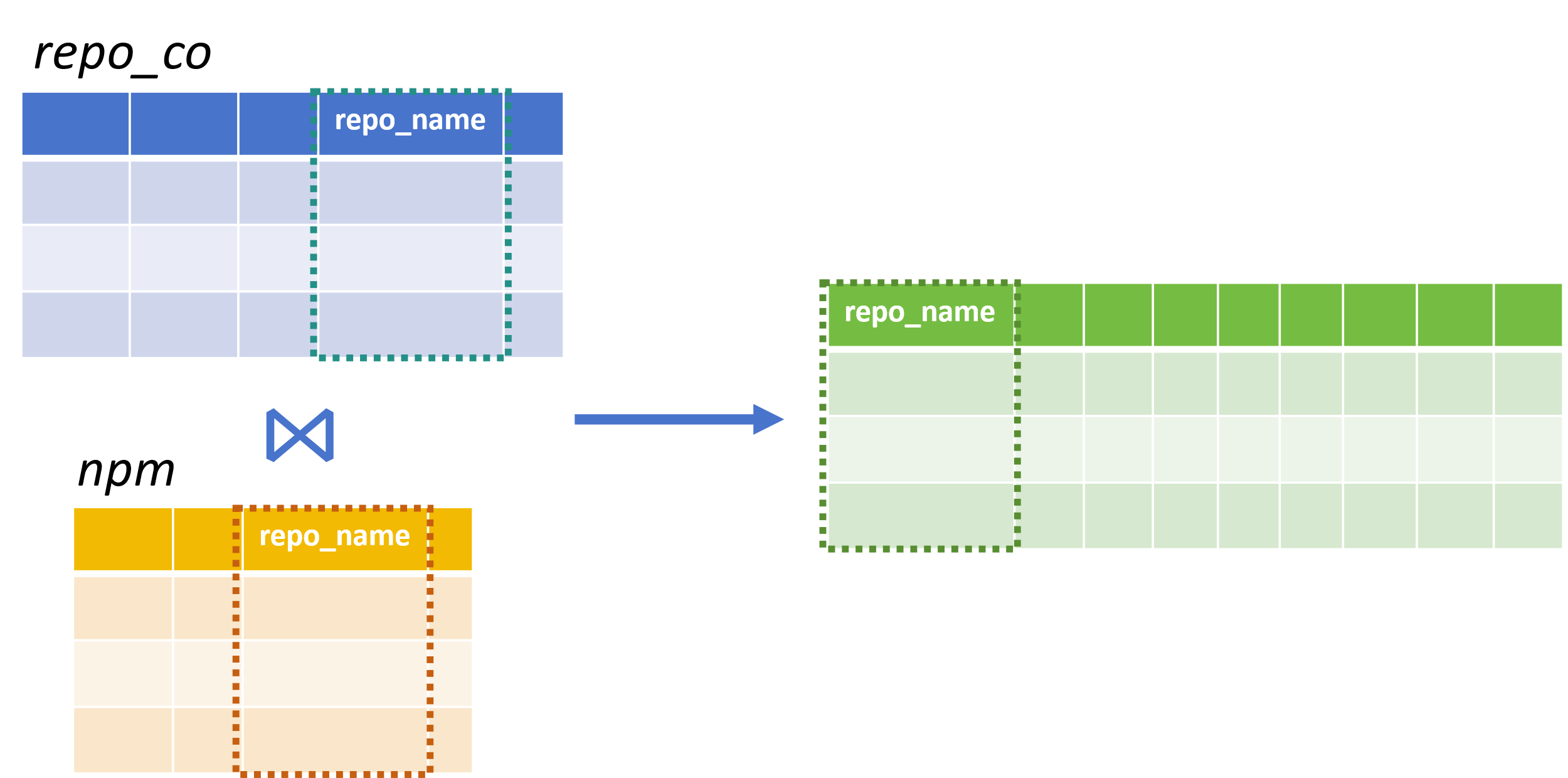


* 网络的平均聚类系数（局部连接紧密度）：0.1056
* 平均路径长度和直径（网络整体连接性）：非连通图，无法计算

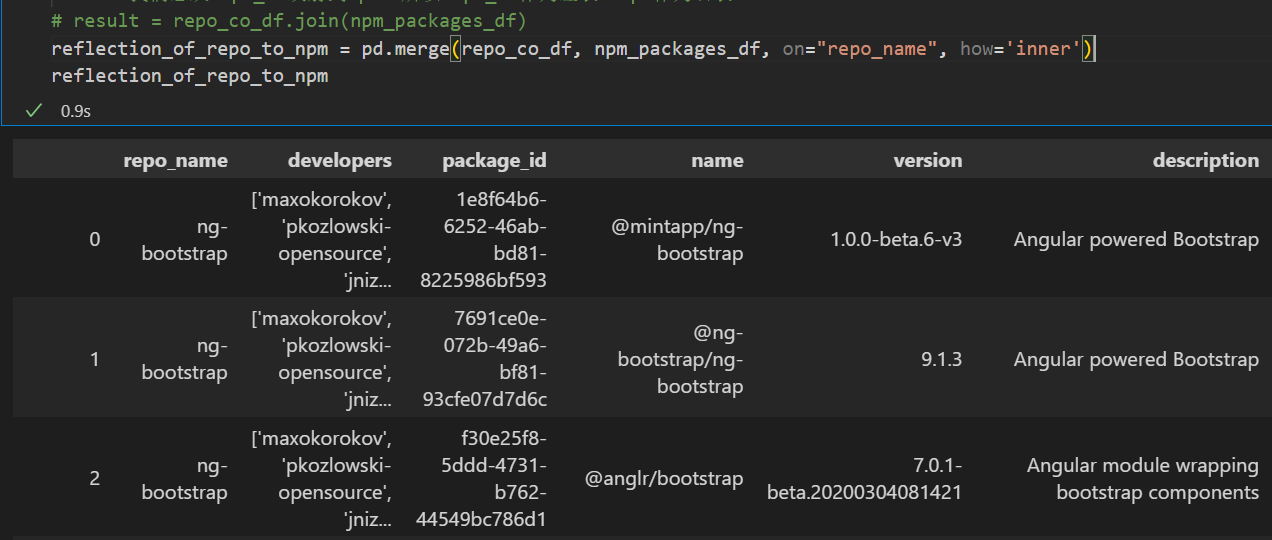
**五、开源协作网络与npm依赖网络的映射**

我们认为，将各个开源仓库到npm包的映射关系。现在有了包含npm包名字、repo\_url字段的csv数据以及包含repo\_url、贡献者字段的csv数据。我们发现，这两个数据可以通过共同的repo\_url进行自然连接，从而可以通过repo\_url映射到相应的npm包，给开发任务做出指导性意见。

为了使得这样的映射方式简单易用，我们便使用类似于关系型数据库的JOIN算子的方式，将repo\_co表和npm表进行连接操作，他们具有共同的字段repo\_name，所以可以使用自然连接的方式。



使用pandas的接口可以完成。由于我们想从repo\_co映射到npm，因此，应当将repo\_co表作为左表，而npm表作为右表。



我们生成的文件是csv格式的，便于在其它的开发任务中读取和使用，尤其是使用pandas框架来处理csv数据是极为方便的。在使用这个数据进行仓库开发中使用npm包的指导时，可以遍历数据框，找到开发者所关心的repo\_name的那些行，然后读取npm包，就可以知道，哪些包是和该repo相关联的了。

（注：由于生成的数据集太大，超过1.5GB，因此没有放在我们的github仓库里，我把这个文件单独发到助教老师您的邮箱里了，注意查收，麻烦老师了~）

**六、发现与结论**

在本次任务中，我们深入探索了**开源仓库协作网络与npm工件库依赖网络之间的复杂映射关系**，通过构建这两个网络并对其进行详细分析，并且做成了最后的大的数据集。

最初的任务是通过**GitHub API收集**了仓库的元数据，包括贡献数量、贡献性质及协作持续时间等关键属性。这些属性不仅揭示了仓库内部的活跃度与协作模式，还通过连边展示了开发者社区中跨项目的合作网络。我们发现，一些核心开发者在多个高影响力仓库中均有显著贡献，这些“桥梁”人物在促进知识流动与技术创新方面发挥了重要作用；而npm包的依赖链则揭示了包之间的复杂依赖网络，这一网络不仅是软件构建的基础，也是潜在风险与脆弱性的传播路径。

在将两个网络进行映射的过程中，我们利用**npm包信息中的repo\_url字段作为桥梁**，尽管这一映射并非完全一一对应，但确实为我们提供了有价值的视角来观察开源项目如何转化为npm包，并进而影响整个软件生态系统。通过映射分析，我们发现了一些开源仓库虽然活跃度很高，但其对应的npm包在生态系统中的影响力却相对有限，这提示我们可能需要更多的推广与集成工作来充分发挥这些仓库的潜力。

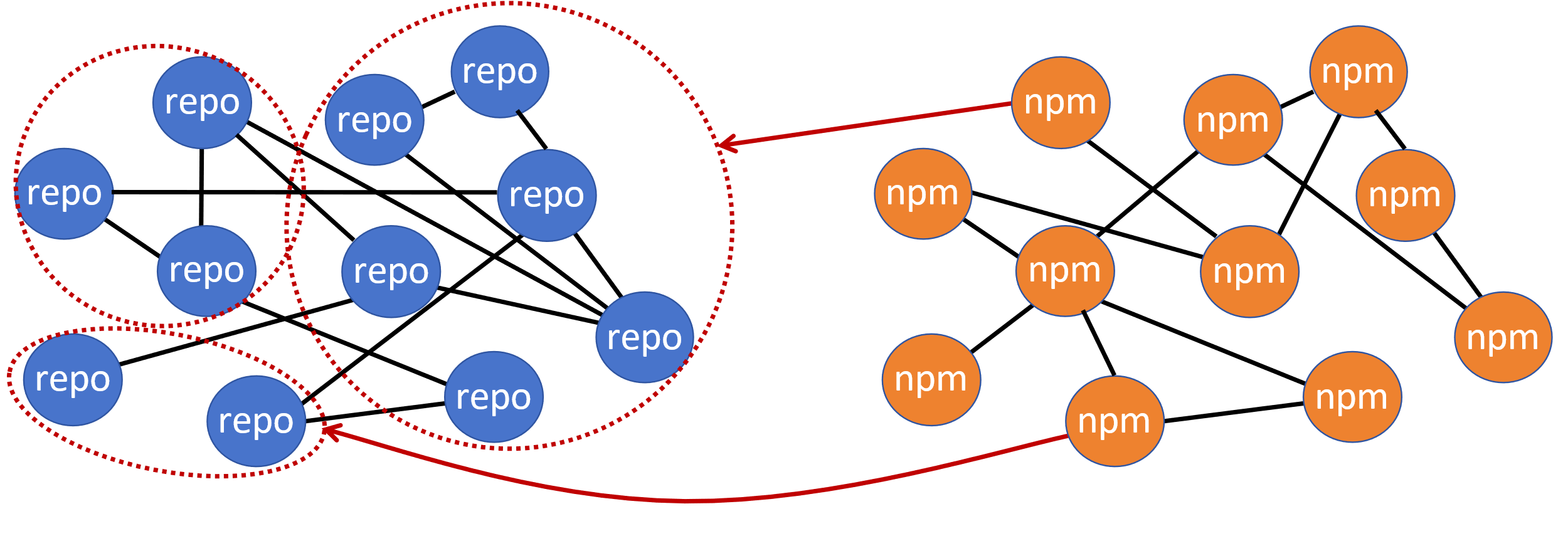
对于网络图本身，我们**定量地**计算了网络的度数、聚类系数**等指标**，提供了网络结构的量化描述，例如，高聚类系数可能意味着网络中存在紧密联系的社区结构，而低平均路径长度则表明信息在网络中传播的高效性；我们也**定性地**进行了网络**可视化分析**，直观地展示了网络的拓扑结构与关键节点，从而更好地理解了网络的结构特征，还为我们进一步研究软件生态系统的弹性、**评估软件开发实践**随时间的趋势提供了有力的工具。

上述对于开源仓库协作网络与npm工件库依赖网络的构建和分析能够揭示**软件生态系统**内部的复杂**关系**与动态**变化**。这些发现不仅加深了我们对开源软件与npm生态系统的理解，还为未来的研究与实践提供了宝贵的参考与启示。

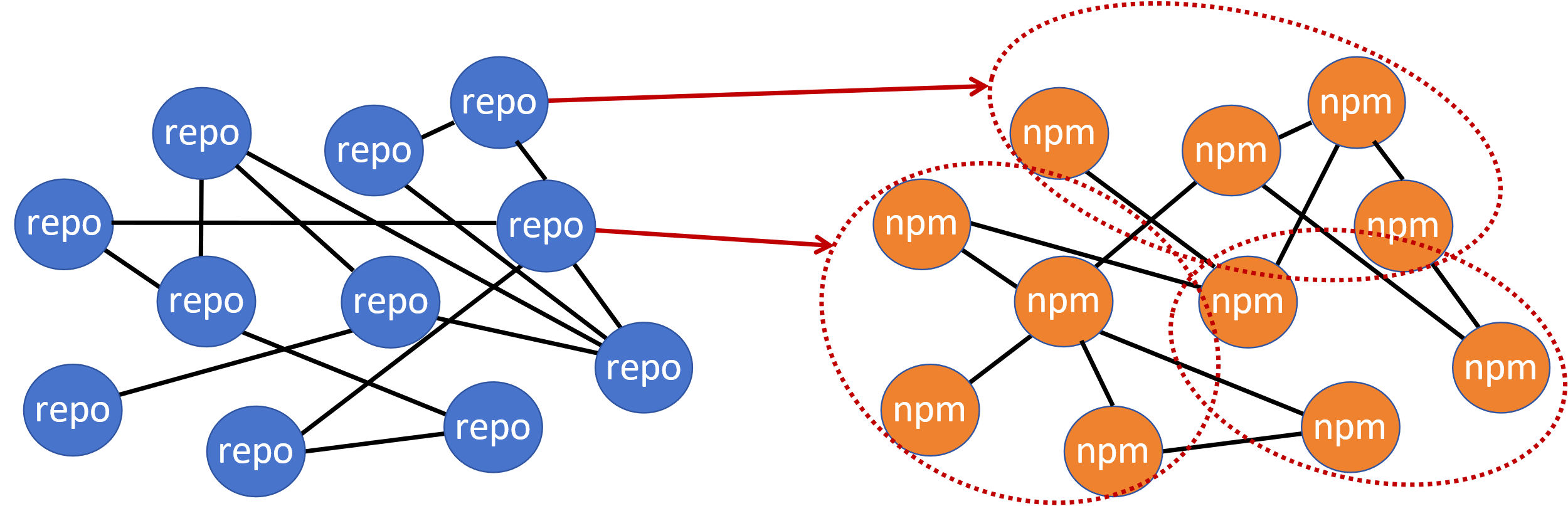
**七、未来工作展望**

针对上述任务的背景，我们认为未来可以尝试对网络做**社区发现**。社区发现能够将庞大的网络划分成多个社群，直观上来看，则是将细粒度的图转换成了粗粒度的图。通过社区发现算法，我们可以以“社群”为单位来进行映射，解释性或许会更强。

比如说，可以对开源仓库网络进行**社区发现**，建立起各个npm包到开源仓库的映射关系，这种关系可以指导各个npm包，让他们在提供相应的npm资源时，倾向于向映射的一大群开源仓库一同提供npm包。



也可以对npm依赖网络进行社区发现，建立起各个开源仓库到npm包的映射关系，这种关系可以指导各个仓库，让他们了解各自很可能使用到一大群npm包，从而可以帮助提高开发效率。



此外，也可以对两个网络都做社区发现，进行相互映射，使得上述映射的粒度变粗。

