EE447 Mobile Internet Project Report

Xie Zhihui
517030910356

Xu Zihan
517030910385

Xu Shangning
517030910384

Cao Jianzhen
517030910368

ACM Reference Format:
Xie Zhihui, Xu Zihan, Xu Shangning, and Cao Jianzhen. 2020. EE447 Mobile Internet Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 OVERVIEW

In this project report, we present our efforts to collect collaboration data on US and Chinese grants, connecting data to existing Acemap data, analysis and insights into collaboration data. The code is open source, accesible at https://github.com/madcpt/VisNSF. The dataset we've collected is available at https://jbox.sjtu.edu.cn/l/Vooi2L.

For collaboration between Chinese researchers on National Natural and Science Foundation (NNSF) grants, we utilize collaboration data provided by the course (the `cn_co` table).

For collaboration between US researchers on NSF grants, we utilize data on NSF grants provided on the NSF websites.

For collaboration between Chinese and US researchers, we try to map all researchers into Acemap, or more precisely, into the `am_author` table, so that each each researcher has an ID in the `am_author` table. After mapping, their collaboration is extracted by their co-authorships on papers, reflected in the table `am_paper`.

After data collection, collaboration graphs are stored in the graph database for fast retrieval. We apply complex network analysis to the collaboration graphs to gain insights into the dynamics of collaboration.

Next to section headings are italicized member names, which indicate that he or she is credited for that section's contribution *and* is the author of that section in the report..

2 DATA COLLECTION

By Cao Jianzhen

In this section, we present our efforts on mapping authors from Chinese NNSF grant collaboration data (the `cn_co` table) into `am_author`, so that each researcher has an author ID from the `am_author` table.

2.1 Crawler

Since there is no practical way to match researchers from the NSFC grants directly into Acemap, we need to utilize other information to match. Fortunately we find a website, ***http://output.nsfc.gov.cn***, which contains papers sponsored by each grant. We think if we can get these paper information, and combine with the table am_paper, then authors',i.e. the principals' and participants' IDs can be matched. So we design a crawler to get the paper data under grants we want to match.

First of all, we find that the table cn_co's content is in unicode, so we decode and extract useful information from it, i.e. ratify number of each grant and name of each participant. With these ratify numbers we design a crawler to website ***http://output.nsfc.gov.cn***. Our crawler realizes

(1) According to a certain ratify number, get each paper's title and achievementid which is an identical number of a paper in this website, of a grant.
(2) According to a certain achievementid, get journal name of a certain paper.

The clue is clear but there is still some hardness, that is, the anti-crawler mechanism of NSFC website. Through experiments we find NSFC has a serious but not very strict anti-crawler mechanism. The 'strict' means that if one crawler visits it too frequently, relevant ip will be forbidden for a long time(about 2 days). However, 'not very strict' means that the limit seems relatively high(we let crawler stop 0.01s per visit).
The large amount of data(300,000 grants) is difficult to deal as well. We attempt to use proxy technique including tunnel proxy and proxy pool, but find the proxies are in poor quality though we buy some proxy service. After a series of attempts we decide to cut part of grants(about 150,000) into 4 segments for crawling together at last.

2.2 ID Matching

Getting the data from Internet, we commence matching process. The ultimate goal is getting principals' and participants' IDs in table cn_co.
We have known that principals and participants of grants are also some authors, whose names and papers can be found in NSFC website. We explain how to run the matching process, and for simplicity we concentrate on principal matching.
Now we begin to card the thread. We have ratify numbers of grants and want to get relevant principal id but there is not direct way. However, with paper titles crawled, we can search relevant paper_id in database am_paper, table am_paper. Then we get paper_ids under a grant ratify number. Similarly with a paper_id searched, we can further get a paper's author_id from database am_paper, table am_paper_author.
After that we get author_ids under a grant ratify number, the next is to find which one is the principal's id. The answer is in database NSF_CN, table nsfc_conclusion, where all grants' principals' names exist.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Note that there are only names in database NSF_CN, table nsfc_conclusion (no id), so we need to search names in database am_paper, table am_author with author_id, and use the searched names to search in database NSF_CN, table nsfc_conclusion. If a name exists in the table, then relevant author_id is the principal_id.
We normalize the thread for clarity.

(1) Search paper_id with title crawled from Internet in database am_paper, table am_paper.
(2) Search author_id with paper_id in database am_paper, table am_paper_author.
(3) Search name with author_id in database am_paper, table am_author.
(4) Search(Check) name in database NSF_CN, table nsfc_conclusion. If the name exists, execute 5; otherwise 1.
(5) Insert grant ratify number and relevant author_id into table cn_nnsf_grants.

It seems that matching has been accomplished but not yet. A difficult problem exists in step 1, that is, title is a string and the table am_paper is very large, thus searching becomes very slow.
To handle it we need to utilize journal name. With the information we can search journal_id(integer) and narrow the scope(condition query). We now normalize the new thread again.

(1) If a paper has a journal name, search journal_id with journal name in database am_paper, table am_journal. Otherwise the omit and go to the next.
(2) Search paper_id belonging to a certain journal_id with title crawled from Internet in database am_paper, table am_paper.
(3) Search author_id with paper_id in database am_paper, table am_paper_author.
(4) Search name with author_id in database am_paper, table am_author.
(5) Search(Check) name in database NSF_CN, table nsfc_conclusion. If the name exists, execute 5; otherwise 0.
(6) Insert grant ratify number and relevant author_id into table cn_nnsf_grants.

Actually there exist some papers which we cannot crawl their journals, so a trade-off between speed and quantity is inevitable. We think it is worthy because speed accelerates a lot and ultimately we match over 190,000 grants, which is not bad as well.

3 AUTHOR MAPPING

By Xu Zihan

In this section, we present our solution to the problem of mapping US researchers into the Acemap am_author table. This problem is different from Chinese collaboration because unlike Chinese NNSF grants, papers associated with each grant are not available. Therefore, we must find a researcher's author ID in am_author solely based on their name.

3.1 Problem Description

Author Mapping is used for present an author-ID matching for authors. Given an author full name in a source database, we would need to map from the name to Acemap-ID. For example, author with name 'Maria M. Almanzar' should be mapped to AuthorID 1000000415'. Author disambiguation has been a major issue in academic database management and research performance evaluation. Several unsupervised methods have been designed for working within a single database. However, most of these existing methods rely on features that are specific to bibliography data, such as citation relationships and coauthor patterns, which are not available for other types of scholarly contributions. As a manual name disambiguation approach, the Open Researcher and Contributor ID initiative (www.orcid.org) attempts to assign global identifiers to researchers, hence linking authors across several publishers. Although this system might be able to request each author to link their new scholarly contribution to their ORCID with the cooperation of several existing databases, a huge number of records accumulated over time still remain without author identification. Our research on Author Mapping can contribute to this type of database management.

3.2 Difficulties

Due to many reasons, there are lots of difficulties in putting Author Mapping in practice.

(1) Large volume of data: 91,458,238 authors in total. Even with Acemap server, it took almost 2 minutes to traverse, and it caused over 8 GB dump;
(2) Confusion between duplicate names: more than 10 authors with name 'A. A. A. Mohamed';
(3) Ambiguity rising from degeneracy: 'San Zhang' or 'San ZHANG' or 'Zhang San';
(4) Ambiguity rising from multilinguality: non-English names could result in ambiguity;
(5) Multiple Institutions: for instance, Geoffrey Hinton worked for Google and the University of Toronto at the same time;
(6) Typo: 'Keith Ross' (NYU Professor) and 'Keith Rose' (surgeon) and 'Keith Ros' (nobody);

We designed several strategies for this task. We also constructed a real-world database based on AcemapKG. Experiments with different settings were conducted, with analysis and comments made.

3.3 Strategy One: Exact String Matching

The most straight-forward way of mapping an author to his/her ID is to run string matching. If we can find a name in database that matches exactly with this name, then we will consider it a successful match. Otherwise, if we cannot find such a match even after traversing the whole database, we will use a special notation 'NULL' for the author's ID, indicating that we failed to find a exact string match.

The workflow of Strategy One can be formulated as:

(1) Normalize author name;
(2) Traverse the database, retrieve every author name;
(3) Find out if exact string match could be found.

The advantages of Strategy One are obvious. Strategy One is low in system resource consumption, all the system do is to traverse the database and do exact string matching. As a result, query system based on Strategy One is ultra fast.

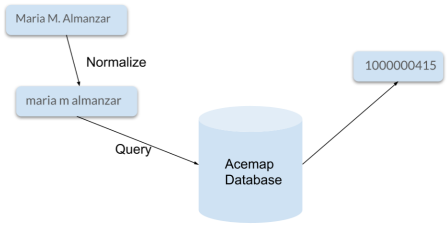

Figure 1: An Overview of Strategy One

There are also many disadvantages of Strategy One. First is it cannot deal with even character-level typos. Moreover, it still suffers from name ambiguity.

3.4 Strategy Two: Exact String Matching + Affiliation Matching

Another easy way of author mapping is to run string matching and affiliation matching. If we can find a name in database that matches exactly with this name and also matches at the affiliation, then we will consider it a successful match. If there is no exact match of both conditions, we will run Strategy One instead and return the result. If we still cannot find such a match even after all two steps, we will use a special notation 'NULL' for the author's ID, indicating that we failed to find a exact string match.

The workflow of Strategy Two can be formulated as:

(1) Normalize author name and corresponding affiliation;
(2) Traverse the database, retrieve every author's name and affiliation;
(3) Find out if exact string match and affiliation match could be satisfied;
(4) If not, run Strategy One and return results;

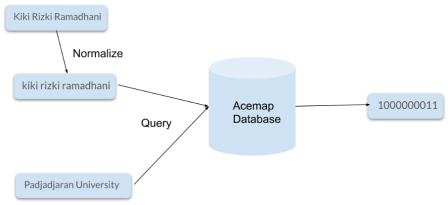

Figure 2: An Overview of Strategy Two

The advantages of Strategy Two are similar with Strategy One. First, it is low in system resource consumption since the only extra operation is running affiliation matching. As a result, query system based on Strategy Two is ultra fast (while still slower than Strategy One, which is reasonable).

While Strategy Two is able to at some extent ease the name ambiguity problem, it still lacks the ability of correcting typos. Also, when an author has no affiliation, this strategy may fail to work.

3.5 Strategy Three: Similarity Ranking of Name + Affiliation

In order to ease the problem caused by typo errors, we decide to introduce a new way of evaluating string matching. Here comes Levenshtein distance.

The Levenshtein distance between two string is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. In approximate string matching, the objective is to find matches for short strings in many longer texts, in situations where a small number of differences is to be expected. The short strings could come from a dictionary, for instance. Here, one of the strings is typically short, while the other is arbitrarily long. This has a wide range of applications, for instance, spell checkers, correction systems for optical character recognition, and software to assist natural language translation based on translation memory. The Levenshtein distance can also be computed between two longer strings, but the cost to compute it, which is roughly proportional to the product of the two string lengths, makes this impractical. Thus, when used to aid in fuzzy string searching in applications such as record linkage, the compared strings are usually short to help improve speed of comparisons. In linguistics, the Levenshtein distance is used as a metric to quantify the linguistic distance, or how different two languages are from one another. It is related to mutual intelligibility, the higher the linguistic distance, the lower the mutual intelligibility, and the lower the linguistic distance, the higher the mutual intelligibility.

The Levenshtein distance between two strings a,b can be calculated as:

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) \text{ if } \min(i,j) = 0 \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

Before running the string match, we will concatenate the author name and affiliation. For example, 'Paul' in 'Shanghai Jiao Tong University' will be presented as 'Paul Shanghai Jiao Tong University'. Then we will use Levenshtein distance as the scoring metric, and compute a similarity score for each name-affiliation pair in the database. Finally, pick the ID with max similarity score.

The workflow of Strategy Three can be formulated as:

(1) Normalize author name and corresponding affiliation, and concatenate them for scoring;
(2) Traverse the database, retrieve every author's name and affiliation;
(3) Run distance algorithm on each author to give a similatity score;
(4) Pick the top-ranking author.

Many advantages of Strategy Three are observed. First, the ambiguity problem can be well-addressed due to the use of affiliation information. Moreover, typos can be tolerated by the algorithm due to the introduction of Levenshtein distance. As can be also imagined, with Strategy Three, you can always find a top matching, which is due to the benefits of soft-matching.

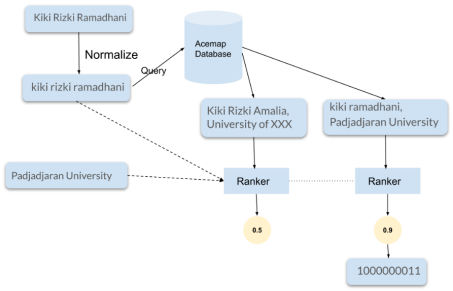

Figure 3: An Overview of Strategy Three

However, Levenshtein algorithm also introduces much higher system overhead. Although it can stilled be executed in large scale, we cannot guarantee the speed.

3.6 Strategy Four: similarity ranking of Name + Collaborators (based on Graph Neural Network)

All the above strategies focus on the author him/herself, while the collaborators can also be helpful. For an author, we can extract all the collaborators and use the collaboration information to assist Author Mapping.

We use Graph Neural Network (GNN) to do the job. Graph Neural Network(GNN) recently has received a lot of attention due to its ability to analyze graph structural data. This article gives a gentle introduction to Graph Neural Network. It covers some graph theories for the ease to understand graphs and the problems in analyzing graphs. It then introduces Graph Neural Network in different forms and their principles. It also covers what GNN can do and some applications of GNN. A graph is a data structure consisting of two components: vertices, and edges. It is used as a mathematical structure to analyze the pair-wise relationship between objects and entities. Typically, a graph is defined as G=(V, E), where V is a set of nodes and E is the edges between them. A graph is often represented by an Adjacency matrix, A. If a graph has N nodes, then A has a dimension of (NxN). People sometimes provide another feature matrix to describe the nodes in the graph. If each node has F numbers of features, then the feature matrix X has a dimension of (NxF). As we know, a graph does not exist in a Euclidean space, which means it cannot be represented by any coordinate systems that we are familiar with. This makes the interpretation of graph data much harder as compared to other types of data such as waves, images, or time-series signals("text" can also be treated as time-series), which can be easily mapped to a 2-D or 3-D Euclidean space.

We first use Char-level embedding and a GRU to generate the representation for names, then we use GNN to encode the collaboration network and get the corresponding representation of the author node. Finally we use cosine similarity as a score metric for ranking.

The workflow of Strategy Four can be formulated as:

(1) Normalize author name, corresponding affiliation, and all collaborators;
(2) Traverse the database, retrieve every author's information;
(3) Run Char-level GRU to generate name representations;
(4) Run GNN encoder on each author to derive corresponding representations;
(5) Use cosine similarity function to give a similatity score;
(6) Back propagate when training, or pick the top-ranking author when testing.

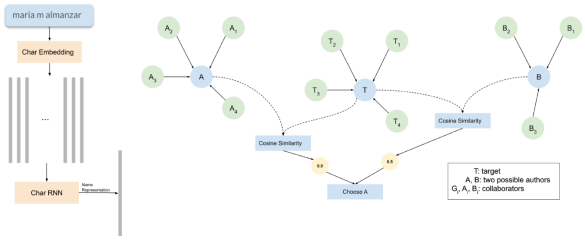

Figure 4: An Overview of Strategy Four

The advantages of Strategy Four are obvious. First, the ambiguity problem can be well-addressed due to the use of collaboration information. Second, typos can be tolerated by the algorithm due to the use of Char-level GRU. Moreover, the model can learn the ability of deduction.

However, this approach is of course too slow and resource-consuming to be put in practice.

3.7 Dataset

To construct a real-world dataset, we use authors from NSF_US and randomly picked 100,000 authors as dataset, randomly extract 10% of it as test set. We manually create typos at rate α in test set. The type of typo includes insertions, deletions or substitutions, which are reasonable. We also set a drop rate β of affiliation and collaborators, since these information may not always be present in real-world practice.

As is shown below, we conducted experiments using different parameters, and further analysis was made regarding the difference.

3.8 Experiments

We conducted several experiments using different parameters and different mapping models. We also measured the training time and the inference time of each model. All the experimental results are shown in Table 1.

As can be seen from the results, there are two settings: ideal scenario ($\alpha = \beta = 0$) and real-world scenario ($\alpha = 0.2, \beta = 0.8$). In the ideal scenario, all methods worked out pretty well. In the term of inference time, Strategy One and Two have great advantage over the other two, with the Strategy Four requiring extra training time. In the real-world scenario, Strategy Four achieved highest accuracy, and Strategy Three was not so bad. However, Strategy One and Two are both sensitive to the two parameters, which means that they do not have very good robustness.

	Training Time*	Inference Time*	Acc*	Acc**
Strategy One (Name)	-	0.01 s/item	0.93	0.42
Strategy Two (Name + Affiliation)	-	0.02 s/item	0.98	0.53
Strategy Three (Levenshtein)	-	0.42 s/item	0.99	0.89
Strategy Four (GNN)	2 hours with 1080Ti	1.98 s/item	0.97	0.95

Table 1: Results of Author Mapping Experiments. (* : with α = 0 and β = 0; **: with α = 0.2 and β = 0.8)

3.9 Collaboration Mining

Now that we have a list of US grant participants and a list of CN grant participants, we need to find an efficient way to extract US-CN collaboration Graph. Since the database is too big to fit into memory, we designed an efficient way of extracting collaboration graph.

The first step is to extract all papers that US or CN scholars participated. We use batched query to accelerate IO operation, which proves to be very useful. With single-record query, it would take over a whole day to complete; while with batched query (batch size was set to 100), it only took 20 minutes to finish. In total, we got 57,371,915 papers that were related to US scholars and 13,600,629 papers that were related to CN scholars.

The second step is to traverse both list to find all US-CN scholar pairs where the US scholar co-authored with the CN scholar. This step is terrible in resource consumption. It took over 3 hours to find 2,017,246 matches.

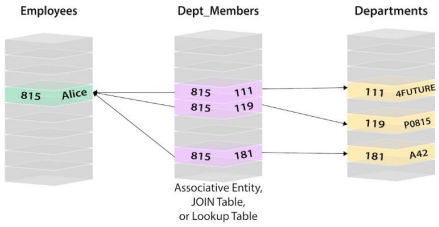

Figure 5: Relational Model

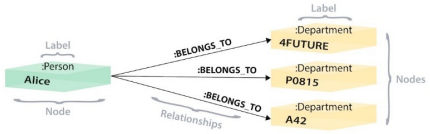

Figure 6: Graph Model

4 GRAPH DATABASE

By Xie Zhihui

4.1 Introduction

4.1.1 Motivation. As the scale of data grows exponentially, a significant issue lies on how to store and query them in an efficient way. For storage, We not only need to consider how these data are placed in the database at the very beginning, but also reduce the overhead of reconstructing when new data arrive. For query, which is usually more important, all we care about is speed.

We may rely solely on relational databases, which sure can provide us with the ability to maintain and access data for a thousand items of authors. But how does it scale? We may do some simple analysis here using an example of employee management[1].

Suppose you are a HR who forgets which department Alice belongs to. If the technicians of your company are SQLers, they may provide you with such a solution: First, check our employeeName-to-employeeID table to obtain the ID of Alice. Next, you can use the employeeID-to-departmentID table to get the ID of department. And don't forget the last step: you have to finally use the departmentID-to-departmentName table to get the right name of department! You might think it quite a complicated solution. Any better idea?

A natural idea is to avoid these intermediate processes and go straight for the answer. Suppose we consider employees and departments as nodes and there are edges connecting them, which represent some kind of affiliation. The whole company can then be organized as a large graph, with simple node and edge elements. In this manner, finding which department Alice belongs to is equivalent to finding the outgoing edge with the label of 'belongs to'.

Therefore, to handle complex networks, we have to take the property of complex networks (or more generally, graphs) into account, which naturally leads us to the topic of graph databases.

4.1.2 Graph Database. A graph database is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data. Unlike traditional relational databases which link data implicitly, graph databases can directly define the relationship between nodes. The relationships allow data in the store to be linked together directly and, in many cases, retrieved with one operation. Graph databases hold the relationships between data as a priority. Querying relationships is fast because they are perpetually stored in the database. Relationships can be intuitively visualized using graph databases, making them useful for heavily inter-connected data.

Another advantage of graph database is about expansibility. For relational databases, minor adjustments may lead to the reconstuction of a whole table. Graph databases, on the other hand, have no such an issue since relationship information are stored directly into pieces of data, which makes it much easier to maintain in the long run.

But graph databases are not just the opposite of relational databases. Rather, they stand for some abstraction beyond the physical devices where the data is actually stored. We can still depend on a relational engine and store the graph data in tables. However, a more natual way is to use a NoSQL[7] database for storage (e.g., HBase[4]), making them inherently NoSQL structures.

Retrieving data from a graph database requires a query language other than SQL, which was designed for the manipulation of data in

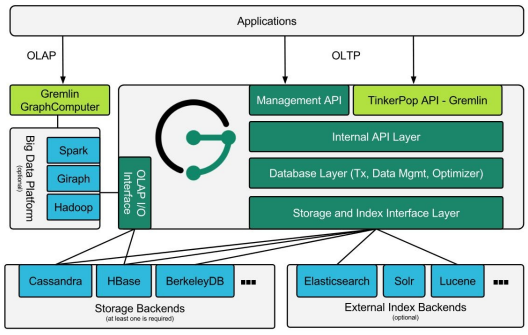

Figure 7: Basic Framework

a relational system and therefore cannot 'elegantly' handle traversing a graph. In this project, we use Gremlin[3] to query, which a graph traversal language and virtual machine developed by Apache TinkerPop of the Apache Software Foundation.

4.2 Implementation

To implement the graph database, we choose the open source, distributed graph database JanusGraph[5]. It is optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster. The framework is similar to the following:

4.2.1 Use HBase as Storage Backend. We deploy a standalone HBase database on the same local host as JanusGraph. JanusGraph and HBase communicate with one another via a 'localhost' socket. To manage collarboration data of three types—US-to-US, CN-to-CN, and US-to-CN—we construct three different tables 'janus_us', 'janus_cn', and 'janus_uscn' in HBase.

The process of importing data into HBase is done with Janusgraph-utils[6], a code pattern for how to use OLTP APIs to define schema, ingest data, and query graph. It utilizes graph schema definitions and data mappers and write to JanusGraph. For more information, please refer to our 'README'.

4.2.2 Use Solr as External Index Backend. While JanusGraph's composite graph indexes are natively supported through the primary storage backend, mixed graph indexes require that an indexing backend is configured. Mixed indexes provide support for geo, numeric range, and full-text search.

The choice of index backend determines which search features are supported, as well as the performance and scalability of the index. JanusGraph currently supports three index backends: Elasticsearch, Apache Solr and Apache Lucene.

In this project, we choose Solr[8] as our external index backend, which only requires easy configuration with JanusGraph.

4.2.3 Use Gremlin for Traversing Graphs. Once the backend is set up, we can use Gremlin to traverse these three collaboration networks.

Every Gremlin traversal is composed of a sequence of (potentially nested) steps. A step performs an atomic operation on the data stream. Every step is either a map-step (transforming the objects in the stream), a filter-step (removing objects from the stream), or a sideEffect-step (computing statistics about the stream). The Gremlin step library extends on these 3-fundamental operations to provide users a rich collection of steps that we can compose in order to ask any conceivable question.

4.2.4 Connect to Gephi for Easy Visualization. Finally, JanusGraph supports various kinds of visualization tools with easy access. We can directly connect our graph database with Gephi using Graph Streaming plugin[2].

The purpose of the Graph Streaming API is to build a unified framework for streaming graph objects. Gephi's data structure and visualization engine has been built with the idea that a graph is not static and might change continuously. By connecting Gephi with external data-sources, we leverage its power to visualize and monitor complex systems or enterprise data in real-time.

4.3 Results

In actual deployment, we only reserve the basic structure of graphs to save time and space. Each author is represented by a unique ID with labels 'country' and corresponding 'author_id' in Acemap database. Collaboration relationships are represented as undirected edges.

We did some simple tests for our graph database system. For example, it only takes 5.75s to query the US author who has the most number of collaborations with others.

As for visualization via Graph-Streaming, we are faced with the problem of efficiency. It seems that the plugin has very poor performance even comparing to directly exporting the graph to graphml file and then importing to Gephi. Therefore, we in the end abandon this scenario. Some other methods are discussed in the next section.

5 ANALYSIS

By Xu Shangning

After collaboration mining, we construct three collaboration graphs from our data, for US grants, Chinese grants and collaboration between US and China. The collaboration data are constructed from the `am_paper` table of Acemap, where collaboration is represented by co-authoring a paper. This approach is chosen because the alternative, namely collaboration represented by Chinese NNSF grants of type "global collaboration", gives few data and thus is not representative.

We apply techniques from complex network analysis to gain insights into scholars' collaboration on grants and paper. The following section presents methods we use.

5.1 Methods

There are several well known metrics which are widely utilized in complex network analysis. In this section, we briefly provide an overview of the metrics that we use in our analysis.

Size is one of the most basic properties of a network, and is quantified by the number of nodes $|V|$ and the number of edges $|E|$.

The basic characteristic to infer a network's connectivity is average node degree $k = 2n/e$. The degree k of a node is the number of edges that are adjacent to the node. A node with degree k is called as k-degree node, and $n(k)$ is the set of all k-degree nodes in a network. The average node degree can also be calculated by taking the

Framework	Functionality	Speed	Extensibility	Compatibility	Maintenance
graph-tool	Good	Fast	Good, Python or C++	Good	Good
igraph	Good	Good	Good	Good	Good
lightgraph	Good	Fast	Poor, Julia	Poor	De factor standard in Julia
Networkkit	Good	Good	Good	Good	Good
Snap	Poor	Medium	Good	Good	Good
Networkx	Complete	Poor, pure Python	Good, Python	Good	Excellent
Gephi	Poor	Good	Poor, Java	Poor	Poor

Table 2: Cross comparison of frameworks for complex network analysis

mean of the degree of all nodes in the network. Weighted Degree of a node is the sum of the weights of all of the edges that this node has. Node degree distribution is the probability distribution of the node degrees where the probability of having a k-degree node in the network is expressed as $P(k) = n(k)/n$.

Distance is the shortest path length between a pair of nodes in the network. Average Path Length stands for the average distance between all pairs of nodes in the network. Diameter is the maximal shortest distance between all pairs of nodes in the graph, and gives an idea of how far apart are the two most distant nodes.

Assortativity, or assortative mixing is a preference for a network's nodes to attach to others that are similar in some way. The assortativity coefficient is the Pearson correlation coefficient of degree between pairs of linked nodes. Positive values of r indicate a correlation between nodes of similar degree, while negative values indicate relationships between nodes of different degree. In general, r lies between -1 and 1. When $r = 1$, the network is said to have perfect assortative mixing patterns, when $r = 0$ the network is non-assortative, while at $r = -1$ the network is completely disassortative.

The assortativity coefficient is given by

$$r = \frac{\sum_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2}.$$

The term q_k is the distribution of the remaining degree. This captures the number of edges leaving the node, other than the one that connects the pair. The distribution of this term is derived from the degree distribution p_k as

$$q_k = \frac{(k+1)p_{k+1}}{\sum_{j\geq 1} jp_j}.$$

Finally, e_{jk} refers to the joint probability distribution of the remaining degrees of the two vertices. This quantity is symmetric on an undirected graph, and follows the sum rules $\sum_{jk} e_{jk} = 1$ and $\sum_j e_{jk} = q_k$.

Rich club coefficient measure the extent to which well-connected nodes also connect to each other. Two forms of the rich club coefficient have been proposed. The non-normalized form is given by

$$\phi(k) = \frac{2E_{\geq k}}{N_{\geq k}(N_{\geq k} - 1)},$$

where $E_{\geq k}$ is the number of edges between the nodes of degree greater than or equal to k and $N_{\geq k}$ is the number of nodes with degree not less than k. The non-normalized form of the rich club coefficient can be understood as the number of edges between nodes with degree no less than k, divided by the number of edges between the same set of nodes if they are in a complete graph.

A criticism of the above metric is that it does not necessarily imply the existence of the rich-club effect, as it is monotonically increasing even for random networks. In certain degree distributions, it is not possible to avoid connecting high degree hubs. To account for this, it is necessary to compare the above metric to the same metric on a degree distribution preserving randomized version of the network. This updated metric is defined as:

$$\rho_{\text{rand}}(k) = \frac{\phi(k)}{\phi_{\text{rand}}(k)},$$

where $\phi_{\text{rand}}(k)$ is the rich-club metric on a maximally randomized network with the same degree distribution $P(k)$ of the network under study. This new ratio discounts unavoidable structural correlations that are a result of the degree distribution, giving a better indicator of the significance of the rich-club effect. For this metric, if for certain values of k we have $\rho_{\text{rand}}(k) > 1$, this denotes the presence of the rich-club effect.

Clustering coefficient is the measure of how well the adjacency (i.e., neighbors) of a node are connected. The neighbor set *ns* of a node *a* is the set of nodes that are connected to *a*. If every node in the *ns* is connected to each other, then the *ns* of *a* is complete and will have a clustering coefficient of 1. If no nodes in the *ns* of *a* are connected, then the clustering coefficient of a will be 0. High clustering coefficient is the indicator of small-world effect along with small average shortest path.

5.2 Experiment

We analyze three collaboration graphs by running and trying to interpret the metrics. Recognizing the scale of our graphs, we present a comparison between popular frameworks for complex network analysis in the following section.

5.2.1 Framework Comparison. We perform a cross comparison between 7 frameworks to find the best compromise between our criteria, namely functionality, speed, extensibility, compatibility with our skill set, maintenance. The frameworks selected for comparison are graph-tool, igraph, lightgraph, networkkit, snap, networkx and Gephi. The comparison is necessary not only because of the diversity of the frameworks, but also because of the scale of our dataset. In this comparison, we draw benchmark results from [9] and rate each frameworks on different levels for each criteria. Table 2 shows the result our cross comparison.

Dataset	# nodes	# edges	Avg degree	Pseudo diameter	Assortativity	Avg clustering coefficient	Rich club coefficient
US	104524	235320	4.5027	20.0	0.0718	0.4555	< 1
CN	497866	4823099	19.3751	15	-0.0016	0.6430	< 1
CN+US	2723671	1723671	1.2657	2.0	-0.7237	0.8324	< 1

Table 3: Network metrics for our datasets

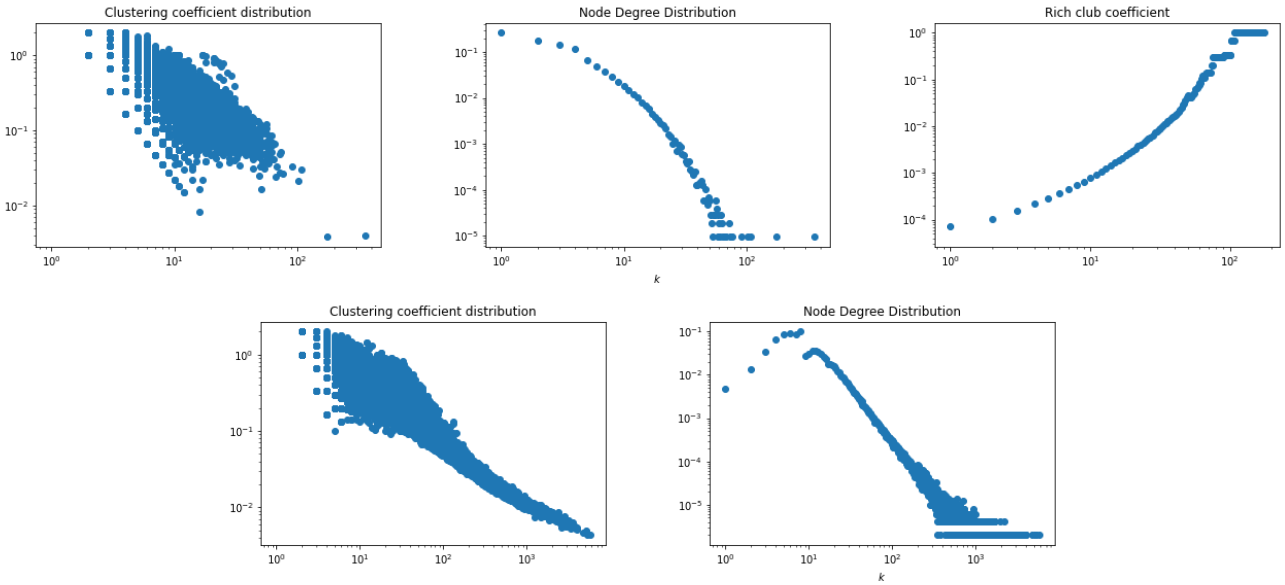

Figure 8: Metrics for collaboration graphs. The first row shows the metric for the US graph, the second row for China. The rich club coefficient is omitted for China due to limited computing resources.

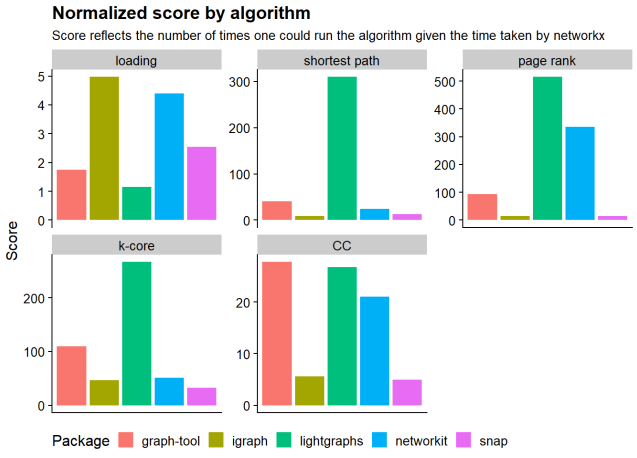

Figure 9: Benchmark results of complex network analysis frameworks, courtesy of [9]

Some frameworks are noteworthy in this comparison. NetworkX is the *de facto* standard for network analysis in Python due to its documentation, maintenance and feature completeness. However, NetworkX is extremely slow. Figure 9 shows that NetworkX is 10 times slower than the second slowest framework, making it undesirable for anything beyond toy datasets. On the other hand, lightgraph is extremely performant and feature rich, with good maintenance due to its status as the *de facto* standard in Julia for network analysis, but is not compatible with our skill set and hard to extend. Gephi was the standard for visualization and network analysis, but recently it is comparatively unmaintained and poor in features, making it the least attractive in our comparison. graph-tool is mostly written in C++ but its primary interface is Python (like Numpy), combining the speed of C++ and the ease of use of Python. Its primary drawback is that it doesn't cover all our metrics, especially the rich club coefficient.

Based on the comparison, we choose to go forward with graph-tool. We implement the missing rich-club coefficient to use in our analysis.

5.2.2 Implementation. We implement the algorithm for computing the rich club coefficient. We use `graph_tool.GraphView` to extract the subgraph for nodes with degree no less than k, for each k. Then wen compute the unnormalized rich-club coefficient for each k on its associated subgraph. To compute the normalized rich-club coefficient, a random graph must be constructed with the same node degree distribution as the input graph. This is accomplished

by the use of `graph_tool.random_rewire`, which swap the edges for two pairs of nodes with the same node degree combination, effectively generating a random graph from the input graph. The unnormalized rich-club coefficient is normalized by the rich-club coefficient of this new random graph.

5.3 Results

Table 3 shows the metrics for each collaboration graph. Figure 8 shows other metrics we explored. The collaboration graphs have millions of edges each. Note that CN+US is the only collaboration graph which have *less* edges than nodes, which is also reflected on its average node degree. This is an indication of the sparseness of the CN+US collaboration graph.

Note that collaboration graphs here ranges from minimally assortataive (US) to highly disassortative (CN+US), rare especially compared to academic collaboration networks studied in [10], which are scale-free network where the degree distribution follows a power law with an exponential cutoff—most authors are sparsely connected while a few authors are intensively connected. Moreover, common scientific collaboration network has an *assortative* nature—hubs tend to link to other hubs and low-degree nodes tend to link to low-degree nodes. Our hypothesis for this phenomenon is the exclusiveness of grants. That is, while well-connected scientists collaborate extensively with their well-connected peers, they *exclude* other well-connected peers from their own grants. More precisely, participants in a grant are always classified into a hierarchical structure involving principal investigators (PI), co-PIs and other contributors. In this way, a well-connected scientist, despite having participated in many grants, mostly have collaborators from "lower" levels in academic. Based on our hypothesis, we can also predict that rich clubs don't exist in grant collaboration graphs, which is confirmed by the rich club coefficient.

It is also surprising that the CN+US colloboration graph only have a diameter of 2, implying the poor connectedness of Sino-US collaboration. In other words, Sino-US collaboration can be compared to relationships in marriage, with an emphasis on "exclusiveness": most researchers have fixed oversea "partners" with which they collaborate.

As a final note, every graph has a high clustering coefficient, which is understandable since all participants in a grant form a complete graph on corresponding collaboration graph.

6 CONCLUSION

In this project report, we present our efforts to collect collaboration data on US and Chinese grants, connecting data to existing Acemap data, analysis and insights into collaboration data.

REFERENCES

[1] Concepts: Relational to graph. URL https://neo4j.com/developer/graph-db-vs-rdbms/.
[2] Graph streaming plugin for gephi. URL https://gephi.org/plugins/#/plugin/graph-streaming.
[3] Gremlin. URL https://tinkerpop.apache.org/gremlin.html.
[4] Apache hbase. URL https://hbase.apache.org/.
[5] Janusgraph, . URL https://janusgraph.org/.
[6] Janusgraph utils, . URL https://github.com/IBM/janusgraph-utils.
[7] Nosql. URL https://en.wikipedia.org/wiki/NoSQL.
[8] Solr. URL https://lucene.apache.org/solr/.
[9] Timothy Lin. Benchmark of popular graph/network packages v2, 2020. URL https://www.timlrx.com/2020/05/10/benchmark-of-popular-graph-network-packages-v2/.
[10] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001. ISSN 0027-8424. doi: 10.1073/pnas.98.2.404. URL https://www.pnas.org/content/98/2/404.