ENS 491/2 - Graduation Project
Progress Report I

Understanding the Academic World

Submitted by
Berkan Teber

Under the supervision of
Kamer Kaya
Hüsnü Yenigün

2017 - 2018

Sabancı University
Faculty of Engineering and Natural Sciences

1 Project Objectives

In this project, our goal is to have a better understanding of the academic world. To achieve this goal we will analyze papers, the building blocks of the academic world, and citations, the interaction between these papers. At the end of this project, we expect to have a method to evaluate authors with respect to different fields and the resulting evaluations. In addition, we expect to have a method to compare different fields and to find the most promising ones among them. To obtain the intended results and to achieve our goal, we have determined various objectives to fulfill throughout the project.

First objective is to preprocess the data we have. Since the data we have is huge, we need some preprocessing before we work on them. This preprocessing step includes extraction of the relevant information and construction of a more accessible data structure for our future work. Moreover, we need to perform some other operations such as Author Name Disambiguation to make the data more suitable to be worked on.

Second objective is to identify different fields. This identification can be performed by running a community detection algorithm such as the Louvain Method on our network and analyzing the comparison of the distribution of keywords within a community and the general distribution of keywords.

Third objective is to evaluate papers and authors with respect to the whole network, and then, with respect to different fields. While this evaluations can be performed in many ways, such as h-index which uses citation counts as the only criteria, we will use a Random Walk based model for our evaluations.

Fourth objective is to compare different fields with each other in various ways. As results of these comparisons, we expect to identify some attributes that a field behaves different than the other fielse. In addition, we expect to identify the promising fields among all fields looking at these attributes that makes a field distinguishable.

Last objective is to represent the results obtained in previous steps such as the paper and author evaluations or the attributes of the fields that distinguishes one from each other or the list of promising fields with various visualization techniques.

2 Performed Tasks

2.1 Constructing Graphs

Initially, we have had 2 datasets: Aminer and Microsoft Academic Graph. We have gathered the data from the Open Academic Society as JSON arrays. The structure of a JSON object in these files can be seen below:

Field Name	**Field Type**	**Description**
id	string	AMiner or MAG id
year	int	published year
authors	list of authors	list of authors
author.name	string	author name
author.org	string	author organization
references	list of strings	list of references
keywords	list of strings	list of keywords
fos	list of strings	list of fields of stufy
...	...	...

Table 1: JSON structure

In this task, I have extracted id, year, authors, references, keywords and fields of study of each paper from these JSON arrays and constructed the following text files:

1. dataset_files.txt
2. dataset_authors.txt
3. dataset_affils.txt
4. dataset_keywords.txt
5. dataset_fos.txt
6. dataset_papers.txt
7. dataset_structured_papers.txt

The layouts of the text files indicating indexes are given below:

dataset_files.txt	
Line No	**Line Content**
1	# of files (N)
2	(empty)
next N lines	file names

Table 2: files.txt layout

dataset_authors.txt	
Line No	**Line Content**
1	# of authors (N)
2	(empty)
next N lines	author names

Table 3: authors.txt layout

dataset_affils.txt	
Line No	**Line Content**
1	# of affiliations (N)
2	(empty)
next N lines	author affiliations

Table 4: affils.txt layout

dataset_keywords.txt	
Line No	**Line Content**
1	# of keywords (N)
2	(empty)
next N lines	keywords

Table 5: keywords.txt layout

dataset_fos.txt	
Line No	**Line Content**
1	# of fields of study (N)
2	(empty)
next N lines	fields of study

Table 6: fos.txt layout

dataset_papers.txt	
Line No	**Line Content**
1	# of papers (N)
2	(empty)
next N lines	paper id's

Table 7: papers.txt layout

The layout of the text file including the graph is given below:

dataset_structured_papers.txt	
Line No	**Line Content**
1	# of papers (N)
2	(empty)
next line	file id for paper #0
next line	year for paper #0
next line	# of authors for paper #0 (A0)
next 2 x A0 lines	author and affiliation id's
next line	# of references for paper #0 (R0)
next R0 lines	referenced paper id's
next line	# of keywords for paper #0 (K0)
next K0 lines	keyword id's
next line	# of fields of study for paper #0 (F0)
next F0 lines	fos id's
next line	(empty)
next line	file id for paper #1
next line	year for paper #1
next line	# of authors for paper #1 (A1)
next 2 x A1 lines	author and affiliation id's
next line	# of references for paper #1 (R1)
next R1 lines	referenced paper id's
next line	# of keywords for paper #1 (K1)
next K1 lines	keyword id's
next line	# of fields of study for paper #1 (F1)
next F1 lines	fos id's
next line	(empty)
...	...

Table 8: structured_papers.txt layout

2.2 Merging Graphs

In this task, I have merged 2 graphs I have constructed in the previous task. In the end, I have constructed the following text files:

1. merged_authors.txt
2. merged_affils.txt
3. merged_keywords.txt
4. merged_fos.txt
5. merged_papers.txt
6. merged_structured_papers_info_aminer.txt
7. merged_structured_papers_info_mag.txt
8. merged_structured_papers_refs.txt
9. merged_structured_papers_keys.txt

The layouts for merged_authors.txt, merged_affils.txt, merged_keywords.txt, merged_fos.txt are the same as the layouts for dataset_authors.txt, dataset_affils.txt, dataset_keywords.txt, dataset_fos.txt.

The layout for merged_papers.txt is as follows:

merged_papers.txt	
Line No	**Line Content**
1	# of papers (N)
2	(empty)
3	aminer id
4	mag id
5	(empty)
6	aminer id
7	mag id
8	(empty)
...	...

Table 9: merged_papers.txt layout

In merged_structured_papers_info_dataset.txt, I have used the year and the authors from the given dataset when the data is available in both datasets. Otherwise, I have used the dataset with the existing data.

The layout for merged_structured_papers_info_dataset.txt is as follows:

merged_structured_papers_info_dataset.txt	
Line No	**Line Content**
1	# of papers (N)
next line	(empty)
next line	aminer file id for paper #0
next line	mag file id for paper #0
next line	year for paper #0
next line	# of authors for paper #0 (A0)
next 2 x A0 lines	author and affiliation id's
next line	(empty)
next line	aminer file id for paper #1
next line	mag file id for paper #1
next line	year for paper #1
next line	# of authors for paper #1 (A1)
next 2 x A1 lines	author and affiliation id's
next line	(empty)
next line	aminer file id for paper #2
next line	mag file id for paper #2
next line	year for paper #2
next line	# of authors for paper #2 (A2)
next 2 x A2 lines	author and affiliation id's
next line	(empty)
...	...

Table 10: merged_structured_papers_info_dataset.txt layout

In merged_structured_papers_refs.txt and merged_structured_papers_keys.txt, I have merged the data from both datasets.

The layout for those 2 text files are as follows:

merged_structured_papers_info_dataset.txt	
Line No	**Line Content**
1	# of papers (N)
next line	(empty)
next line	# of references for paper #0 (R0)
next R0 lines	referenced paper id's
next line	(empty)
next line	# of references for paper #1 (R1)
next R1 lines	referenced paper id's
next line	(empty)
...	...

Table 11: merged_structured_papers_info_dataset.txt layout

merged_structured_papers_info_dataset.txt	
Line No	**Line Content**
1	# of papers (N)
next line	(empty)
next line	# of keywords for paper #0 (K0)
next K0 lines	referenced paper id's
next line	# of fos for paper #0 (F0)
next F0 lines	referenced paper id's
next line	(empty)
next line	# of keywords for paper #1 (K1)
next K1 lines	referenced paper id's
next line	# of fos for paper #1 (F1)
next F1 lines	referenced paper id's
next line	(empty)
...	...

Table 12: merged_structured_papers_info_dataset.txt layout

2.3 Converting to Compressed Row Storage

In this task, I have constructed 3 graphs and converted them into CRS format. Given a graph $G = (V, E)$, CRS consists of 3 arrays as follows:

1. Row Pointer Array:

   Keeps the sum of the number of edges up to and excluding the vertex for each vertex.

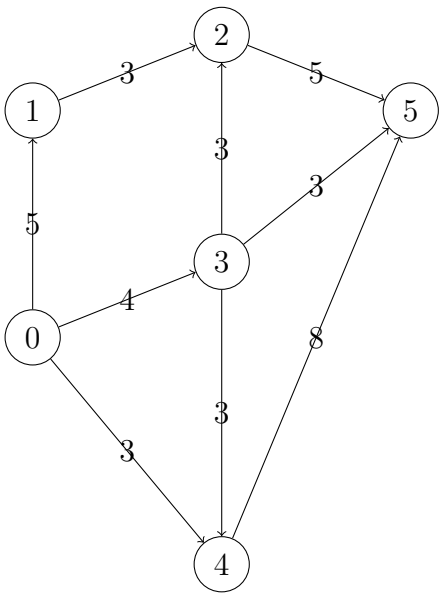
2. Column Index Array:

   Keeps the target of the edges for each vertex.

3. Weights Array:

   Keeps the weights of the edges for each vertex.

Take a graph as below:



Then, corresponding CRS arrays will be as follows:

0	**1**	**2**	**3**	**4**	**5**	**6**
0	3	4	5	8	9	9

Table 13: Row pointer array

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	3	4	2	5	2	4	5	5

Table 14: Column index array

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
5	4	3	3	5	3	3	3	8

Table 15: Weights array

Suppose you want to know all of the outgoing edges from the vertex 3. Then, first, you look at the 3^{rd} and the 4^{th} indexes of the row pointer, which are 5 and 8. Then, you look from 5^{th} to 8^{th} (8 excluded) indexes of the column index array, which are 2, 4 and 5. Similarly, 5^{th} to 8^{th} (8 excluded) indexes of the weights array will give you 3, 3 and 3. This means that there are 3 outgoing edges from 3: to 2 with weight 3, to 4 with weight 3 and to 5 with weight 3.

2.4 Finding Communities

In this project, communities play a very significant role. We can use communities to find and identify the fields, or to detect author or affiliation communities.

There are different methods to find the communities of a network. The Louvain Method [3] is an example of these methods. It is a simple, efficient and easy-to-implement method that optimizes the modularity of a network in a greedy way.

Modularity is one measure of the structure of networks that was designed to measure the strength of division of a network into communities.

The formula of the modularity is as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \Big[A_{ij} - \frac{k_i k_j}{2m} \Big] \delta(c_i, c_j)$$

where $k_i = \sum_j A_{ij}$ is the total link weight that outgoes from node i, $m = \frac{1}{2} \sum_{i,j} A_{ij}$ is the total link weight in the network, $\delta(c_i, c_j)$ is 1 when nodes i and j are assigned to the same community and 0 otherwise.

The Louvain Method is an iterative method which has 2 phases. In the beginning every node is assigned to its own community. In the first phase, for each node, the algorithm checks the neighbors of that node and if changing community will increase the modularity it changes the community. In the second phase, each community is converted into a node and by doing so, a new graph is created.

You can find the pseudocode [4] for the algorithm below:

Algorithm 1 The Louvain Method

```
Let G the initial network
while increase in modularity do
    Put each node of G in its own seperate community
    while previous modularity < new modularity do
        for all nodes do
            Calculate move for node that yields highest increase in modularity
            if there exists a move with positive gain then
                Move the mode to new community
            else
                Let the node stay in its current community
            end if
        end for
    end while
    if the new modularity is higher than the initial then
        Contract G
    end if
end while
```

In the task, I have used the C++ implementation of the Lovain Method by Etienne Lefebvre, who is behind the original idea of the method.

At the end of running the code for the citation network, there have been found 164251600 communities for 256283281 nodes. When we exclude the nodes with no links, there have been found 212553 communities for 92244234 nodes.

2.5 Identifying Fields

In this task, the objective was to identify the top 5 communities using keywords and fields of study. To do so, I have calculated the general distribution of keywords and fields of study. Then, I compared this distribution with the distribution within communities and score each keyword and field of study.

I have used 3 different scoring, which gave similar results:

1. $score = \frac{freq}{expected} \times freq$
2. $score = \sqrt{\frac{freq}{expected}} \times freq$
3. $score = \frac{freq}{expected} \times \sqrt{freq}$

where $freq$ is the frequency of a keywords within the community, and *expected* is the expected frequency obtained by the general distribution and the community size.

According to the results (with respect to the first scoring), top 4 fields of study for top 3 communities are below:

Community #1	
15.2 M / 256 M	
Score	**Field of Study**
15535154	Economics
11746926	Sociology
7907520	Law
7359839	Management

Table 16: Community #1

Community #2	
10.4 M / 256 M	
Score	**Field of Study**
32200218	Computer Science
14740550	Mathematics
11304786	Machine learning
9285702	Mathematical optimization

Table 17: Community #2

Community #2	
9.5 M / 256 M	
Score	**Field of Study**
23167119	Physics
16902072	Materials Science
13837527	Chemistry
11657222	Nanotechnology

Table 18: Community #3

With these top 10 fields of study, I have identified the fields of top 5 communities as follows:

1. Social Sciences
2. Computer Science
3. Physics & Chemistry
4. Biology
5. Medicine

3 Changes in the Plan and Goals

When we look at the current progress, it seems like we are where we have estimated to be. Up to this time, the plan was mainly to explore the data and we have managed to do that. Therefore, at this time, we don't see a necessity to change our plans or goals.

4 Testing Solutions

Up to today, we have explored our data and converted it into various formats we think that will be useful in future tasks. Since we are dealing a huge amount of data, the best way to test the correctness of our solutions is to manually verify some random data. We expect that a verification on some random data will show the correctness of our solutions.

5 Future Tasks

Now that the data exploration part is almost finished, we will move on to find some solutions to the following problems:

1. Evaluation of papers and researchers in general
2. Evaluation of papers and researchers in different fields
3. Extraction of field-specific properties based on evaluations
4. Comparison of different fields
5. Implementation of a system to visualize the results

References

[1] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. (2008). *ArnetMiner: Extraction and Mining of Academic Social Networks.* In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining: p: 990-998.

[2] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. (2015). *An Overview of Microsoft Academic Service (MAS) and Applications.* In Proceedings of the 24th International Conference on World Wide Web: p. 243-246

[3] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre. (2008). *Fast Unfolding of Communities in Large Networks.* In Journal of Statistical Mechanics: Theory and Experiment 2008 (10): P10008 (12pp).

[4] Herman Moyner Lund. (2017). *Community Detection in Complex Networks.* Master Thesis in Department of Informatics, University of Berlin.

1 Appendix: comparedistributions.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <sstream>
4
5  #include <string>
6  #include <vector>
7  #include <map>
8
9  #include <algorithm>
10
11 #include <cstdlib>
12 #include <cmath>
13
14 using namespace std;
15
16 struct key_fos
17 {
18   string kf;
19   double score = 1;
20
21   bool operator<(key_fos const & rhs) const
22   {
23     return score < rhs.score;
24   }
25 };
26
27 int main()
28 {
29   ifstream mkey_in("../0-data/2-merged/merged_keywords.txt");
30
31   string mkey_line;
32   getline(mkey_in, mkey_line);
33   getline(mkey_in, mkey_line);
34
35   vector<string> m_keywords;
36   while (getline(mkey_in, mkey_line))
37     m_keywords.push_back(mkey_line);
38
39   mkey_in.close();
40   cout << "merged_keywords.txt has been read." << endl;
41
42   ifstream mfos_in("../0-data/2-merged/merged_fos.txt");
43
44   string mfos_line;
45   getline(mfos_in, mfos_line);
46   getline(mfos_in, mfos_line);
47
48   vector<string> m_fos;
49   while (getline(mfos_in, mfos_line))
50     m_fos.push_back(mfos_line);
51
52   mfos_in.close();
53   cout << "merged_fos.txt has been read." << endl;
54
55   cout << endl;
56
57   ifstream in("../0-data/2-merged/merged_structured_papers_keys.txt");
58
59   string line;
```

```
60    getline(in, line);
61    istringstream num_iss(line);
62
63    int num_papers;
64    num_iss >> num_papers;
65
66    vector<vector<int>> keywords;
67    vector<vector<int>> fos;
68
69    while (getline(in, line))
70    {
71      vector<int> keys;
72
73      getline(in, line);
74      istringstream key_num_iss(line);
75      int key_num;
76      key_num_iss >> key_num;
77
78      for (int i = 0; i < key_num; i++)
79      {
80        getline(in, line);
81        istringstream key_iss(line);
82        int key_no;
83        key_iss >> key_no;
84        keys.push_back(key_no);
85      }
86
87      keywords.push_back(keys);
88
89      vector<int> foss;
90
91      getline(in, line);
92      istringstream fos_num_iss(line);
93      int fos_num;
94      fos_num_iss >> fos_num;
95
96      for (int i = 0; i < fos_num; i++)
97      {
98        getline(in, line);
99        istringstream fos_iss(line);
100       int fos_no;
101       fos_iss >> fos_no;
102       foss.push_back(fos_no);
103     }
104
105     fos.push_back(foss);
106   }
107
108   in.close();
109   cout << "merged_structured_papers_keys.txt has been read." << endl;
110
111   cout << endl;
112
113   vector<int> keywords_gdist;
114
115   ifstream key_gdist_in("../0-data/5-papercomm/key_dist.txt");
116
117   string key_gdist_line;
118   getline(key_gdist_in, key_gdist_line);
119   getline(key_gdist_in, key_gdist_line);
120
121   while (getline(key_gdist_in, key_gdist_line))
```

```
122     {
123       istringstream key_gdist_iss(key_gdist_line);
124       int key_count;
125       key_gdist_iss >> key_count;
126       keywords_gdist.push_back(key_count);
127     }
128
129     key_gdist_in.close();
130     cout << "key_dist.txt has been read." << endl;
131
132     vector<int> fos_gdist;
133
134     ifstream fos_gdist_in("../0-data/5-papercomm/fos_dist.txt");
135
136     string fos_gdist_line;
137     getline(fos_gdist_in, fos_gdist_line);
138     getline(fos_gdist_in, fos_gdist_line);
139
140     while (getline(fos_gdist_in, fos_gdist_line))
141     {
142       istringstream fos_gdist_iss(fos_gdist_line);
143       int fos_count;
144       fos_gdist_iss >> fos_count;
145       fos_gdist.push_back(fos_count);
146     }
147
148     fos_gdist_in.close();
149     cout << "fos_dist.txt has been read." << endl;
150
151     cout << endl;
152
153     ifstream n2c_in("../0-data/4-louvain/citation_node2comm.txt");
154
155     vector<int> node2comm;
156
157     string n2c_line;
158     while (getline(n2c_in, n2c_line))
159     {
160       istringstream n2c_iss(n2c_line);
161       int node, comm;
162       n2c_iss >> node >> comm;
163       node2comm.push_back(comm);
164     }
165
166     n2c_in.close();
167     cout << "citation_node2comm.txt has been read." << endl;
168
169     ifstream kcomm_in("../0-data/5-papercomm/topkcomm.txt");
170
171     string kcomm_line;
172     getline(kcomm_in, kcomm_line);
173     istringstream kcomm_iss(kcomm_line);
174
175     int k;
176     kcomm_iss >> k;
177
178     getline(kcomm_in, kcomm_line);
179
180     for (int i = 1; i <= 5; i++)
181     {
182       cout << endl;
183       string ii = to_string(i);
```

```
184
185       getline(kcomm_in, kcomm_line);
186       istringstream kcomm_iss(kcomm_line);
187       int comm, comm_size;
188       kcomm_iss >> comm >> comm_size;
189
190       map<int, int> keywords_cdist;
191       map<int, int> fos_cdist;
192
193       for (int node = 0; node < node2comm.size(); node++)
194       {
195         if (node2comm[node] != comm) continue;
196
197         for (int j = 0; j < keywords[node].size(); j++)
198         {
199           int curr_key = keywords[node][j];
200
201           if (keywords_cdist.find(curr_key) == keywords_cdist.end())
202             keywords_cdist[curr_key] = 1;
203           else
204             keywords_cdist[curr_key]++;
205         }
206
207         for (int j = 0; j < fos[node].size(); j++)
208         {
209           int curr_fos = fos[node][j];
210
211           if (fos_cdist.find(curr_fos) == fos_cdist.end())
212             fos_cdist[curr_fos] = 1;
213           else
214             fos_cdist[curr_fos]++;
215         }
216       }
217
218       for (int opt = 0; opt < 3; opt++)
219       {
220         char copt = '0' + opt;
221         string aopt(1, copt);
222
223         vector<key_fos> key_cdist_vec;
224
225         map<int, int>::iterator key_it;
226         for (key_it = keywords_cdist.begin(); key_it != keywords_cdist.end(); ++key_it)
227         {
228           key_fos kf;
229           kf.kf = m_keywords[key_it->first];
230
231           double sizeratio = (double) comm_size / (double) num_papers;
232           double expected = (double) keywords_gdist[key_it->first] * sizeratio;
233           double ratio = (double) (key_it->second) / expected;
234
235           double freq = (double) (key_it->second);
236
237           if(opt == 0)
238             kf.score = ratio * freq;
239           else if(opt == 1)
240             kf.score = ratio * sqrt(freq);
241           else if(opt == 2)
242             kf.score = sqrt(ratio) * freq;
243           else
244             kf.score = 0;
245
```

```
246           key_cdist_vec.push_back(kf);
247         }
248
249         make_heap(key_cdist_vec.begin(), key_cdist_vec.end());
250
251         string key_cdist_filename = "top_keywords_" + ii + "_" + aopt + ".txt";
252         ofstream key_cdist_out(key_cdist_filename);
253
254         key_cdist_out << key_cdist_vec.size() << endl;
255         while (key_cdist_vec.size() > 0)
256         {
257           key_cdist_out << endl;
258
259           key_cdist_out << fixed;
260           key_cdist_out << key_cdist_vec.front().score << endl;
261           key_cdist_out << scientific;
262
263           key_cdist_out << key_cdist_vec.front().kf << endl;
264
265           pop_heap(key_cdist_vec.begin(), key_cdist_vec.end());
266           key_cdist_vec.pop_back();
267         }
268
269         key_cdist_out.close();
270         cout << key_cdist_filename << " has been written." << endl;
271
272         vector<key_fos> fos_cdist_vec;
273
274         map<int, int>::iterator fos_it;
275         for (fos_it = fos_cdist.begin(); fos_it != fos_cdist.end(); ++fos_it)
276         {
277           key_fos kf;
278           kf.kf = m_fos[fos_it->first];
279
280           double sizeratio = (double) comm_size / (double) num_papers;
281           double expected = (double) fos_gdist[fos_it->first] * sizeratio;
282           double ratio = (double) (fos_it->second) / expected;
283
284           double freq = (double) (fos_it->second);
285
286           if(opt == 0)
287             kf.score = ratio * freq;
288           else if(opt == 1)
289             kf.score = ratio * sqrt(freq);
290           else if(opt == 2)
291             kf.score = sqrt(ratio) * freq;
292           else
293             kf.score = 0;
294
295           fos_cdist_vec.push_back(kf);
296         }
297
298         make_heap(fos_cdist_vec.begin(), fos_cdist_vec.end());
299
300         string fos_cdist_filename = "top_fos_" + ii + "_" + aopt + ".txt";
301         ofstream fos_cdist_out(fos_cdist_filename);
302
303         fos_cdist_out << fos_cdist_vec.size() << endl;
304         while (fos_cdist_vec.size() > 0)
305         {
306           fos_cdist_out << endl;
307
```

```
308           fos_cdist_out << fixed;
309           fos_cdist_out << fos_cdist_vec.front().score << endl;
310           fos_cdist_out << scientific;
311
312           fos_cdist_out << fos_cdist_vec.front().kf << endl;
313
314           pop_heap(fos_cdist_vec.begin(), fos_cdist_vec.end());
315           fos_cdist_vec.pop_back();
316         }
317
318         fos_cdist_out.close();
319         cout << fos_cdist_filename << " has been written." << endl;
320       }
321     }
322
323     kcomm_in.close();
324
325     return 0;
326   }
```