Final Report

Jiangnan Zhu

December 2021

1 Problem

This semester, I focus on improving query suggestion in academic search (searching scholarly articles).

After the end user has input a complete query, the search engine will provide related keywords as query suggestions to help the user explore more aspects of the keyword and give the user a direction to look for in the following search. There are two problems in current academic query suggestion.

1. Most mainstream web search engines propose suggestions by utilizing search engine query logs to give popular suggestions. However, in academic search, the user query log information is not very useful. First, the data set is not large enough to analyze. Second every user focuses on unique problems.

2. Most web search engines only provide several related keywords as suggestions and do not allow users to explore further. However, in academic search, the user does not know exactly what he is looking for. He may look for a specific method. For example, when the user enters the "machine learning" query, he might be looking for the "linear regression" method. However, Google Scholar will provide the user with "machine learning algorithms", "machine learning techniques", and "machine learning review" as top suggestions. These suggestions are vague and leads to nowhere. In academic search, we want users to explore through different suggestions and let the user to find the desired keyword.

2 My approaches

1. General Idea
   To help users navigate through keyword suggestions, we create a directed unweighted graph called "Concept Map", where keywords are nodes and edges are relations between keywords, so that users can navigate among keywords through directed edges. Since we cannot use search engine query logs for references, we decide to extract keywords relations directly from corpus. Therefore, we use PMI score to determine the relevance between 2

keywords. We use 83k springer keywords as candidates and use all paper abstracts from ArXiv as corpus. We then calculate PMI score between each pair of keywords and created edges based on PMI. We also trained a language model to evaluate the similarities between query and candidates. When the user enters a query, if the word is a keyword node, the user starts navigating from that node. If not, the language model compares the query with nodes and takes user to the most related keyword node as the starting node to navigate. We use the collapsible tree as the user interface.

2. Improvements
   To improve the concept map, we focus on following aspects:

   (a) The concept map needs to be connected and has nodes of a uniformed degree. It needs to be connected because we want to avoid suggestions that lead to nowhere. Its nodes need to have a uniformed degree because we want to provide the same number of suggestions for each keyword.

   (b) For each keyword, the suggestions need to be representative and lead to distinguished directions because we only provide a limited number of suggestions in each step. For example, there are over 2000 keywords co-occurred with "machine learning" keyword. However, we cannot overwhelm the user with all of them. Therefore, the keywords we selected need to be both highly related to the initial keyword and should have largest semantic differences possible (to ensure they lead to different directions). Moreover, to make the concept map usable, we should generate paths with least overlaps.

   (c) The concept map needs to be sparse and the graph diameter needs to be small. In Facebook network, the average distance between two person is 3.57 steps. Similarly, We want the user to encounter his desired keyword with fewest steps possible. To achieve this, we wish to create a graph with the smallest diameter. We also want the concept map to be sparse and use the least number of edges to avoid overlaps.

   (d) We also need to build a user interface to navigate through the concept map.

3. Process
   We use the co-occurrence dictionary and language model (the model uses gensim word2vec) from $https://github.com/Scott-Huang/Academic_Search$

   (a) We first focused on selecting representative suggestions.
   For each keyword, we want to give types of suggestions with largest semantic differences to help the user explore more directions. We first decided to use k-means clustering to categorize words. K-means clustering transforms each items into vectors and partitions them into clusters. At first, we used all the keywords in the co-occurrence

dictionary as candidates. For each keyword, we picked 100 words with highest PMI score. We then used k-means clustering to separate the 100 keywords into n groups and picked one from each group that's most semantically close to the keyword. We discovered that the graph can be strongly connected when n = 3. We checked the method by printing out paths between different keywords. However, the paths are highly overlapped. We thought uni-gram keywords like "algorithm", "time", "methods" could cause this problem since they appeared in most abstracts. Therefore, we excluded these keywords. We applied k-means clustering again. It turned out the paths are more distinguished, however, the graph diameter became large, and once we reduce the diameter by adding more connections, overlaps reappeared. We then guessed it could be the fault of k-means clustering itself. Moreover, based on observation, we cannot guarantee that k-means clustering provides better results for categorizing than simple co-occurrences. Therefore, we created connections based on PMI scores instead. For each keyword K, we create directed edges from K to m keywords with highest PMI scores. This has two advantages. First, it ensures the high relevance of the keywords. Second, it creates less overlapped paths.

(b) We then focus on getting the minimum diameter of the graph while using the least number of edges.

   i. Initially, we attempted to reduce the graph diameter first by adding edges and then reduce edges and maintain the diameter. To reduce the graph diameter, we used the "Center nodes" method. Intuitively, we create a set of nodes as centers of the graph, and connects them to reduce distance. We first create a set M contains all keywords candidates and an empty set N. We add the first element of M to N, then loop through M. For each iteration, we check the distance between keyword m in M and every keyword in N. If distance d(m,n) $\geq$ x, we add a double directed edges between m and n. The algorithm is described in algorithm 1. According to algorithm 1, we can reduce the diameter to be $2x + 2$, since nodes in set N are at most 2 from each other and nodes in M\N are at most x to any element in N. So together the largest distance would be $2x + 2$. This method guarantees the small diameter. However, it creates a center node c that have very large degree and many unrelated connections. Therefore, we modify the algorithm to connects only co-occurred nodes in the N.

   ii. we then reduce edges
   We use the "Central Hub" method. We preserve nodes with higher in-degree (because we already controlled the out-degree by construction). Nodes with higher degree are more important in preserving graph structure since they have more connections.

Algorithm 1 Center Node

```
M ← keywords
N ← [m]
for m in M do
    for n in N do
        if dist(m, n) > x then
            add m to N
        end if
    end for
end for
c ← n
for n in N do
    add double directed edge e(n,c)
end for
```

In G Lindner, CL Staudt's article, they pinpoints that "Central Hub" method best preserves the undirected graph structures and reduces the most edges. Therefore, we adopted this algorithm. We slightly modifies their algorithm to apply to directed graphs. The algorithm is shown in algorithm 2. The method reduces the edges by 79% and maintains 90% of nodes in the largest strongly connected component. However, it does not preserve the diameter. Moreover, we cannot mathematically prove that this method reduces the graph to the least number of edges and maintains the smallest graph diameter. Therefore, we decided to generate a graph with smallest diameter by directly adding least edges to the null graph (graph with only nodes but no edges).

Algorithm 2 Central Hub

```
N ← nodes
for n in N do
    L ← successors of n
    Rank elements in L according to their in degree from largest to smallest
    Maintain edges from n to top degree(n)^α elements in L
end for
```

iii. We first understood it as degree-diameter problem, which aims at finding the largest graph possible with diameter k such that its largest degree is d. I discovered that most solutions utilize voltage graph, a special branch of Cayley Graphs. To create a Voltage graph, we take some initial graph G and assign a direction to G. We then find some group S (some element set Z, and an operation P) and assign the values of Z to each edge under the operation P such that if the voltage along the edge direction is k then the voltage in the opposite direction is $\frac{1}{k}$. We derived the

voltage graphs by expanding its nodes as the Cartesian Product of the nodes in graph G and elements in Z and add edges under certain restrictions. The key idea is to expand the graph by creating more nodes and edges. This requires super computers to find the optimal solutions. However, our concept map has a fixed number of nodes. Therefore, we cannot use voltage graph. So, we decided to create our own algorithm, which leads to our current approach.

3 Current Approach

1. Method
   First, we discovered that the graph diameter when every keyword connects to co-occurred words is 10. We get the null graph and add edges from each keyword to k words with highest PMI scores. This ensures a uniform degree. We discovered that k = 5 is the smallest k to include over 90% co-occurred keywords in the strongly connected component. Therefore, we use k = 5 as our base graph, where the diameter d = 17. We add more edges to reduce the graph diameter d based on the following method:
   First, we compute the distance between every pair of keywords. We then rank them from largest to smallest and iterate through them. For each path greater than the graph diameter, we label the nodes from start node to end node t from 0 to n. We then check if the start of the path node s has degree more than 15 (we manually set this number to avoid too many suggestions for a single keyword). If not, we check if there are any keyword v in the middle of the path that's co-occurred with s, starting with the index $\lceil \frac{3}{8}n + \frac{7}{6} \rceil$. If so, we create a directed edge from s to v and breaks out of the loop. Otherwise, we oscillatingly check the nodes co-occurrence relation. If we are unable to check nodes oscillatingly, we start checking from the smallest node that haven't been checked. For example, if we have path [0, 1,2,3,4,5...16], we check in the order [8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 15, 16]. Notice that we do not check between 0 and 1 because it only adds duplicated edges. After getting out of the loop, we update the path list and continue on another path until all the path distances are less than 10. We describe the algorithm in Algorithm 3.

2. Reasoning

   Claim: the above algorithm uses the least number of edges when creating the smallest graph diameter

   Intuition: To reduce the diameter of the graph, we want to shorten all the paths that are longer than the graph diameter. For each path, when adding an edge to a path, we want to reduce the sum of distances from the start of the path to every other node in the path the most.

Algorithm 3 Edge Addition

```
L ← paths for all pairs of nodes
sort L by distances from the largest to the smallest
for list l in L do
    label nodes as 0,1,2,3...n, 0 is the start and n is the end
    if n ≥ 11 then
        if Node 0 hasn't been checked and has degree ≤ 15 then
            if node ⌈3/8 n + 7/6⌉ co-occurred with node 0 then
                add directed edge e(0, ⌈3/8 n + 7/6⌉)
                break
            else
                check other nodes as described above
            end if
        end if
        update L to reflect changes
    end if
end for
```

(a) Observation:
if P is the shortest path from s to t, then for each pair of nodes u, v (u appears before v) along this path, uv is also the shortest path.
We prove by contradiction. Suppose u, v are nodes between s and t, we have path from s to u, u to v, and v to t as a shortest path from s to t. Suppose we have a shorter path from u to v, then s to v will be shorter. Contradiction!

(b) Proof
We define sum of distances in path P to be the sum of distance from s to every other nodes in P. For example, if we have a labeled path [0, 1,2,3..9], the sum of the distances would be d(0,9)+d(0,8)+...+d(0,1) = 45.
An edge is efficient if it addition to a path produces the minimum sum of distances.
For a shortest path st, we label the nodes from 0 to n in order with s = 0, t = n. Then the sum of distances for P is $\frac{n}{2}$(1+n). Suppose we want to add an edge from 0 to i, where i is a node between s and t, the sum of distances after addition will be:
Before node i: 1+2+3+..+(i-1) = (i-1)$\frac{i}{2}$
After node i: 1+2+3+...+(n-i+1) = $\frac{(n-i+2)(n-i+1)}{2}$
Sum together, we get:
(n^2+3n+2) + $\frac{3}{2}$(i^2 - ($\frac{3}{4}$n+$\frac{7}{3}$)i)
By algebra, when i = $\frac{3}{8}$n + $\frac{7}{6}$, we get the smallest diameter. Therefore it is the best choice. If node 0 and node i are not co-occurred, we oscillatingly check nodes, this would gradually increase the sum of distances. Since the index $\frac{3}{8}$n + $\frac{7}{6}$ is closer to the first half, after

running out of nodes, we check from the farthest index explored of the path.
Therefore, for each path, we add the most efficient edge. Since we update the list L after each addition, we do not add unnecessary edges. Now there are two situations. First, if paths are internally disjoint (there are no overlap between every path), then we are done. So we assume there are overlaps between some paths. If we do not skip any node s (every start of the path is co-occurred with some internal nodes), we are done. If not, by our observation, for one path, its internal paths are also the shortest paths. So if we take some internal node q as the new start of the path in the next step and continue the process, the statement also holds.

(c) Results
In reality, we cannot update the path list L after every addition, because adding one edge will affect numerous paths and it's expensive to update. So we iterate through list L by distance from d = 17 (the initial graph diameter), 16, 15,...8 (since some s does not co-occur with any internal nodes) instead
There are 3 advantages of this construction. First, the PMI score grantees that the keyword suggestions are related and are not significantly overlapped. Second, the graph achieves the optimal diameter. Every keyword can be reached within 10 steps. Third, the graph is sparse. For each keyword, only 5 related keywords on average are suggested. Comparing to the base graph (each keyword connects to all the co-occurred keywords, with diameter 10), the new graph only uses 12.3% edges. To improve the implementation of this method, we need to find effective ways to update the distance list.

(d) User Interface
First, we adopted noe4j as our user interface. However, neo4j does not give a clear insight on the layers of the suggestions. Therefore, we switched to collapsible tree. For each layer, we suggest some keywords for users to explore. The result is shown below. When user enters a query in the search box, if the query is a keyword node in the concept map, the collapsible tree will treat it as the root of the tree. Otherwise, a language model will pick the most schematically related keyword as the root. When the user find his target keyword, he can search it in the desired keyword box, which is connected to google scholar.

4 Assessment and Reflection

In the beginning of the semester, I only have a rough idea about the concept map. I did not even formulate my problem. This semester I formulated my problem and set the criteria of the concept map. I tried different algorithms to use the least number of edges and achieves the smallest graph diameter possible.

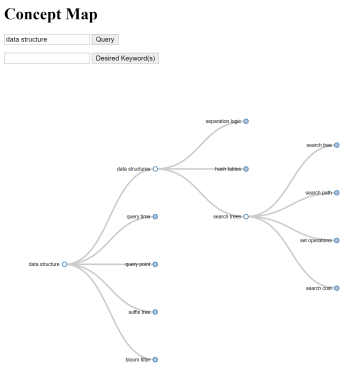

Figure 1: User interface

I also developed an interface to help users navigate through the concept map.

I felt the most difficult part of research is to define the problem and find the direction. I did not have a clear picture in the beginning. Therefore, I did not progress much. I then kept reading articles to do comparisons. I learned lots of other similar problems such as telecommunication, transportation systems, and recommendation systems. I learned many algorithms and ideas from these articles. However, each systems have its own restrictions. I also read articles regarding graph leanings and learned many prototypes. These articles helped me formulate my ideas eventually and everything became easier when I created my basic graph (a directed graph that connects all the co-occurred keywords).

The second challenge in research is to overcome constant frustration. I spent more time on research than any other courses this semester. However, sometimes I cannot get positive feedback in reading papers because not all (if there are any) papers can be directly applied to my project. Moreover, even if some projects (algorithms) are related, most of the papers are dealing with undirected graphs. So I need to do some modifications. But these modifications can lead to unwanted behaviors and the results might not be useful in the end. For example, I modified the center nodes methods several times to make it work properly but I did not adopt the method in the end. Therefore, I decided to break from reading papers every two hours if I find nothing interesting or my implementation has some unwanted but unclear behavior, which helps a lot.

Despite two challenges, I enjoy the biggest merit of learning. I took graph theory this semester. Since I am doing research about graph diameter, I found some graph theory problems in class not hard to understand. I also took numerical methods and found many matrix concepts similar to the ones regarding the robustness of graphs. I also learned to write html code by following tutorials. Since running code on local computer is slow, I learned to connect to the remote,

which accidentally helps me test the network protocol in my CS241 final. In the end, I feel that although many papers cannot be applied to the project directly and been presented in the weekly meetings, they help in other perspectives in a way or another.

5 Future Plan

I will continue to do research next semester and finish my senior thesis. For Concept Map, I think I need to modify the algorithm to make it computationally cheaper. I think there are flaws in my algorithm but I haven't found a counter example that could use less edges yet. But I will keep finding and revise my algorithm. Moreover, I think I may explore more projects or discover more aspects of query suggestions. I found it deficient to focus on a single project. I think if I can do two to three projects at a time, I can switch my thoughts when I get stuck and become more productive.