**Project Report**

Group name: CLZZ

Students: Anhua Chen (anhua@uchicago.edu)

Xiang Zhang (snzhang@uchicago.edu)

Xiuyuan Zhang (xiuyuanzhang@uchicago.edu)

David Liu (dliu5@uchicago.edu)

Github repository: <https://github.com/zhangxiang0822/CS123_Project>

**Project Overview**

This project explored the topic of innovation diffusion across industries using patent citation data. Through analyzing the dynamics of the patent citation behaviors across industries, we aimed to find the diffusion patterns of patents within and across industries. To achieve this goal, we performed several large data merges leveraging the Google Cloud Platform for parallel computing; further, we used the Dijkstra algorithm to find the shortest paths between each patent citation pair and aggregated these pairwise computations to the industry level. For this specific project, we focused on patents granted in the U.S due to the magnitude of the citation data. This means that, for a citation pair where A cited B, both A and B were patents granted in the U.S.

**Data Description**

We used a total of three datasets: (1) NBER patent classification data, (2) U.S. patent citation data, and (3) NBER industry classification data. All three datasets were downloaded through the United States Patent and Trademark office website: <http://www.patentsview.org/download/>

**(1) U.S. patent citation data** (98,207,057 rows, 3.505 GB). The U.S. patent citation data provides information on citations of U.S. patents made by U.S. patents. Each row of the dataset contains nine columns: universally unique id, patent number, patent number for which the current patent cites, date of when the cited patent was granted, name of the cited record, kind code from WIPO, country where the cited patent was granted (always U.S.), category (who cited the patent), sequence (order in which this reference is cited by select patent).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| uuid | patent\_id | citation\_id | date | name | kind | country | category | sequence |
| 00000jd7thmiucpaol1hm1835 | 5354551 | 4875247 | 1989-10-01 | Berg | NULL | US | NULL | 11 |
| 00000l0ooxrvfv6jkenobhwis | D674253 | D519335 | 2006-04-01 | Ishii | S | US | cited by examiner | 13 |
| 00001nlwuimui60vu3k1yzjqd | 8683318 | 6642945 | 2003-11-01 | Sharpe | B1 | US | cited by examiner | 6 |
| … | … | … | … | … | … | … | … | … |

Table. 1 U.S. patent citation sample data

**(2) NBER patent classification data** (5,105,937 rows, 110.807 MB).Each row of the NBER patent classification data contains category information for a single patent, including the universally unique identifier for the patent, the patent number, a NBER category id, and a NBER subcategory id.

|  |  |
| --- | --- |
| id | name |
| 11 | Agriculture, Food, and Textiles |
| 12 | Coating |
| 13 | Gas |
| … | … |

|  |  |  |  |
| --- | --- | --- | --- |
| uuid | patent\_id | category\_id | subcategory\_id |
| 000114qfli99qqd9fsbxichy1 | 6243839 | 2 | 22 |
| 0001jsdl1xi7z84rzx9iwvdlh | 4646100 | 2 | 21 |
| 0001qpsb0yts8daudtuf3mbm8 | 7712627 | 6 | 68 |
| … | … | … | … |

**(3) NBER industry subcategory data** (37 rows, 871 bytes). This data contains information of 37 pairs of subcategory IDs and subcategory names. The set of subcategories is developed by the National Bureau of Economic Research (NBER) and widely used for economic research.

Table. 2 NBER patent classification

sample data

Table.3 NBER industry subcategory sample data

**Data Merge & Preprocessing**

**Summary Statistics**

**Finding the Shortest Path**

We implement Dijkstra Shortest Algorithm to find the shortest path between nodes in a graph. Dijkstra's original algorithm found the shortest path between two nodes. In our implementation, we fix a single node as the “source” node and find the shortest paths from the source to all other nodes in the graph. Further, to analyze the network structure of the graph, we find the shortest path between each node A and node B in the graph by iterating over all nodes and set them as our source node.

**Small-data Dijkstra**

In the small-data world, the Dijkstra works as follows (Gass and Fu, 2013)

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbours and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

In our implementation, the we construct a data structure graph with two attributes:

1. Node: A list of node IDs.
2. Edge: A dictionary of list. The key of this dictionary is the node ID, and the value is a list storing ID of all of its neighbors.

The pseudo code is shown below

1 **function** Dijkstra(*Graph*, *source*):

2

3 create unvisited set Q (Now Empty)

4

5 **for each** vertex *v* in *Graph*:

6 dist[*v*] <- INFINITY

7

8 add *v* to *Q*

9

10 dist[*source*] ← 0

11

12 **while** *Q* is not empty:

13 *u* <- vertex in *Q* with min dist[u]

14

15 **for each** neighbor *v* of *u and v in Q*:

16 *alt* <- dist[*u*] + 1 //In our setting, distance are always 1

17 **if** *alt* < dist[*v*]:

18 dist[*v*] <- *alt*

19

20 remove *u* from *Q*

21 **return** dist[], prev[]

**Large-Data Dijkstra**

In the large-data world, we want to paralyze Dijkstra algorithm. In fact, we implement a breadth-first search instead.

Data representation: In large-data world, we slightly modify the graph structure. We use a single line to represent information of a node. A single line is of the following format with four parts:

ID|ID1, ID2, ID3, .. IDn|Color|Distance

The first part represents the node ID; the second part, separated by commas, stores IDs of its neighbor nodes; the third part is an indicator of whether this node have been visited/to be visited and can take three values; the last part stores the distance.

The Pseudo code is shown below

1 **class Mapper**

2 **method Map**(NodeID nid, NodeInfo n):

3 dist <- n.distance

4

5 if n.color = ‘gray’:

5 **for each** NodeID mid in n.neighbors:

6 m.dist <- dist + 1

7 m.color = ‘gray’

8 yield (NodeID mid, NodeInfo m)

9 n.color = ‘black’

10 yield (NodeID nid, NodeInfo n)

11

12 **class reducer**

13 **method Reduce**(NodeID nid, NodeInfo n):

14 dist\_min <- infinity

15 M <- []

16 Color <- ‘white’

17 **for each** m in n.neighbors:

18 if m.distance < dist\_min:

19 dist\_min <- m.distance

20 if m.neighbors != []:

21 M.extend(m.neighbors)

22 if m.color = ‘black’:

23 color = ‘black’

24 if m.color = ‘gray’ and color = ‘white’:

25 color = ‘gray’

26

27 update NodeInfo n using dist\_min, M, Color

28 yield (NodeID nid, NodeInfo n)

**Discussion**

**Reference**

Gass, Saul; Fu, Michael (2013). "Dijkstra's Algorithm". *Encyclopedia of Operations Research and Management Science*. Springer. **1**. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1007/978-1-4419-1153-7](https://doi.org/10.1007%2F978-1-4419-1153-7)