Project report: PageRank / HITS computation

Sandro Baccega
Ca' Foscari University of Venice
865024@stud.unive.it

Link Analysis is a technique to evaluate the relationship between nodes. In Computer Science it's used in search engines that ranks web pages. PageRank[1], by Sergey Brin and Lawrence Page, and HITS[2], by Jon Kleinberg, are two examples of algorithm that utilize it. We implemented the two algorithms in an efficient way, added the in degree of the nodes to add another data point, and benchmark them together.

I. Introduction

In this report we are going to: analyze our C++ implementation of: *PageRank*[1], *HITS*[2], and the in degree of the nodes, benchmark them and confront them using **Jaccard's Coefficient** (section: I-B). In order to have a real life graph sample, we are going to use the web graphs of the *Stanford Large Network Dataset Collection*[3]. Since the graphs from this database contains thousands of nodes and millions of edges, we cannot use the common matrix representation of a graph, but instead we have to use the **Compressed Sparse Row** (section: I-A) representation.

A. Compressed Sparse Row (CSR)

The CSR is a representation of a matrix that can greatly reduce the memory requirement compared to traditional matrix representation if the matrix is sparse. It accomplish this by avoiding filling values in the matrix that are 0, and in our case it means that a matrix that should be hundreds of GB worth of memory can instead occupy a few MB, making it manageable.

It accomplish this by writing all the non-zero values of the matrix in the *Values* vector, then for every value there is a corresponding index in the *Indexes* vector. The only thing missing is the *Rows* vector, that is used to delimit the when a row of the matrix ends and another one begins (figure: 1).

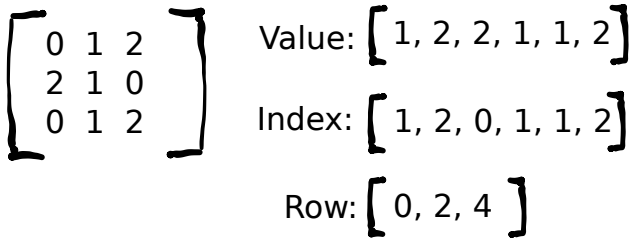

Figure 1: Compressed Sparse Row (CSR)

B. Jaccard's coefficient

The Jaccard similarity coefficient is used to determine the similarity between two sets of data. You can calculate it using formula 1, where A and B are the two sets of data that we need to compare.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1}$$

C. PageRank

PageRank[1] is a link analysis algorithm that assigns a rank to each node of a graph that is a representation of an hyperlinked set of documents.

The core of the algorithm can be represented by formula 2, where:

- P is the probability distribution vector.
- d is the damping factor (teleportation probability).
- n is the number of nodes.

- A^T is the transposed stochastic matrix of the graph.

$$P_{k+1} = (1-d) * \frac{1}{n} + d * A^T * P_k \tag{2}$$

D. HITS

Hyperlink-Induced Topic Search[2] (HITS) is another link analysis algorithm that assigns two different scores to a node: the authority score (this is the one that we are focusing on for this project) and the hub score. A hub score is the sum of the authority values of the pages it points to. An authority score is computed as the sum of the scaled hub scores that point to that page.

The core of the algorithm can be represented by formula 3, where:

- a_k is the authority score at step k.
- L is the normal adjacency matrix of the graph.

$$a_k = L * L^T * a_{k-1} \tag{3}$$

II. Implementation

In this section we will describe in detail our C++ implementation of the algorithms and data structure used for this project.

A. Compressed Sparse Row (CSR)

We implemented CSR (section: I-A) in the file named `csr.h` and it is used in all of our algorithms. In this file we can find the implementation of a class CSR that, beside the constructor, offers only two usable methods: `vectorMultiplication(vector<double> vector, vector<double> destVector)` and it's transposed variant with the same signature. These two methods are used to easily compute the multiplication of the csr matrix with a desired vector.

The core implementation of the CSR logic is inside the constructor of the class, that is resposible for: parsing the raw file of the graph, sorting it, transpose it if it's necessary and creating the two temporary files that will contain the values of the the index and row vectors.

We used the **mmap** (Memory Map) function from the C language to avoid filling the stack memory of our program with the raw data of the matrix in order to further optimize the handling of the graph's data. We used mmap in two functions: `openMMap(string mapFilename, int mapPointerSize)` and `closeMMap(int *pointer, int pointerSize)` that can be found in the file `csr.h` as well.

B. MaxHeap

In the `utils.h` file we implemented a basic MaxHeap that is used throughout the application to pick the topK results of every algorithm efficiently. We choose to represent every pair of *{node;score}* in the MaxHeap using a `pair<double, int>`, in order to simplify the code of the heap, since by deafult, C++ uses the first argument of the pair in comparisons.

C. PageRank

We implemented PageRank[1] (section: I-C) in the file `pagerank.h` using the power iteration method, and it's main flow is in the function called: `getPageRankTopK(string filename, int topK, double dampingFactor)`. The PageRank algorithm is divided in few main steps:

- The stochatization of the csr matrix, done in the specific `stochatization(CSR csr)` function.
- The main loop (that exits if the previous iteration did not cause an update of the probability vector):
  - The current probability distribution is multiplied (using the vector-matrix multiplication algorithm) with the stochastic matrix.
  - Teleportation is taken into account.
- The sorting of the final results using a MaxHeap.

D. HITS

We implemented HITS[2] (section: I-D) in the file `hits.h` using the power iteration method, and it's main flow is in the function called: `getHitsTopK(string filename, int topK)`. Hits, like the PageRank algorithm, is divided in a few main steps:

- The main loop (that exits if the previous iteration did not cause an update of the authority vector):
  - The current authority vector is multiplied (using the vector-matrix multiplication algorithm) with the transposed matrix.
  - Normalize the new authority vector.
- The sorting of the final results using a MaxHeap.

E. In degree

To get the in degree of every node in a CSR matrix, we just had to count how many times a nodes appeared in the index vector. To do this efficiently we used a `map<int, int>` and looped through the index vector. After that we just had to put all the scores (each divided by the total number of nodes to normalize it) into a MaxHeap in order to find the topK nodes with the biggest in degree.

III. Benchmark and output

In this section we'll take a look at the output of our program.

A screenshot of a sample application output is available in figure 2.

This sample's data is collected using a topK of *10* (default value is 20, but it made it difficult to read the screenshot) on the graph *web-NotreDame* by SNAP[3] with a damping factor of *0.85*.

These values are calculated on a Apple MacBook Air 2020 powered by an Apple M1 chip (ARM).

```
PageRank - Hits
------
TopK                <- 10
Damping Factor      <- 0.85
Filename            <- assets/web-NotreDame.txt
------

- PageRank
        - SKIPPING SORTING DATASET
        - COMPUTING CSR START
        - N_NODES: 325729, N_EDGES: 1497134
        - TMP FILES CREATED
        - STOCHASTIZATION
        - PAGERANK
        - Converged after 65 iterations
        - SORTING RESULTS
        - RELEASING MMAP
        - DELETING TMP FILES
        - PageRank top 10 nodes:
          1963      -> 0.00206908
          0         -> 0.00206635
          191267    -> 0.00100891
          10336     -> 0.0018818
          124802    -> 0.00121743
          83606     -> 0.000904845
          1973      -> 0.000872016
          12129     -> 0.0010247
          212843    -> 0.00143804
          32830     -> 0.00100536
- HITS
        - SKIPPING SORTING DATASET
        - COMPUTING CSR START
        - N_NODES: 325729, N_EDGES: 1497134
        - TMP FILES CREATED
        - HITS
        - Converged after 282 iterations
        - SORTING RESULTS
        - RELEASING MMAP
        - DELETING TMP FILES
        - Hits top 10 nodes:
          12129     -> 0.00653519
          235904    -> 0.00216743
          198328    -> 0.00216252
          199031    -> 0.00216743
          193592    -> 0.00216253
          155590    -> 0.00216252
          199029    -> 0.00073684
          199028    -> 0.00073684
          199030    -> 0.000736805
          151241    -> 0.00216253
- InDegree
        - SKIPPING SORTING DATASET
        - COMPUTING CSR START
        - N_NODES: 325729, N_EDGES: 1497134
        - TMP FILES CREATED
        - RELEASING MMAP
        - DELETING TMP FILES
        - InDegree top 10 nodes:
          12129     -> 0.0329139
          0         -> 0.0233906
          31331     -> 0.0132012
          140170    -> 0.0131367
          124802    -> 0.0215701
          155590    -> 0.0109385
          193592    -> 0.0109631
          199031    -> 0.0109815
          151241    -> 0.0109662
          235904    -> 0.0109785
- Jaccard
        - PageRank-HITS: 0.0526316
        - PageRank-InDegree: 0.176471
        - HITS-InDegree: 0.428571
------
All done!
------
Total time elapsed [s]          -> 15.8746
------
PageRank time elapsed [s]       -> 3.80855
HITS time elapsed [s]           -> 9.2009
InDegree time elapsed [s]       -> 2.86511
Jaccard time elapsed [us]       -> 35
```

Figure 2: Sample application output

IV. Conclusions

We have provided an efficient implementation of PageRank[1] and HITS[2] and tested them against a sample of a real world graph in order to benchmark them. The benchmarks unfortunately does not shows if the topK results from our algorithms are relevant, but it shows, as expected, how the PageRank[1] algorithm is almost 2 times faster than HITS in our configuration.

References

[1] Sergey Brin and Lawrence Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: *WWW Int.l Conf.* 1998.

[2] Jon M. Kleinberg. "Authoritative Sources in a Hyperlinked Environment". In: *Journal of the ACM.* 1999.

[3] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection.* http://snap.stanford.edu/data. June 2014.