Big Data Technologies and Analytics Assignment 3: Graph Clustering

Anatoliy Zigangirov, Razim Garipov, Vyacheslav Karpov

May 2019

1 Introduction

Graph clustering is an unsupervised learning problem. Given a dataset, the goal of clustering is to divide the dataset into clusters such that the elements assigned to a particular cluster are similar or connected in some predefined sense. In our particular case, we view a cluster as a group where nodes are densely inter-connected and sparsely connected to other parts of the network. Graph clustering has applications in quite a few areas: it is used to understand the structure and derive all kinds of interesting findings in World Wide Web, social networks, biological networks, molecular groups, etc. Many instruments have been developed over the years for graph storage and processing.

2 Problem statement

Many algorithms have been proposed to find clusters in graphs [3]. For example, hierarchical clustering, minimal spanning tree, betweenness clustering, etc. One of the main problems of clustering algorithms is the running time due to the fact that most of the algorithms have $O(m \log(n))$ or $O(n^2)$ complexity, where n is count of vertexes and m is count of edges. In order to make the computations reasonable for large graphs, algorithms should not grow faster than $O(n)$. There is also a problem of storing entire graphs in memory: it is either impossible or cost of swapping memory contents during the running time is critical. To cope with such issues, local clustering algorithms were introduced. They require access only to a subset of edges and vertices at a time, which is also parallelizable. We are given a task of implementing a graph clustering algorithm using one of the local clustering algorithms—label propagation, which has proved to be simple and efficient. It must be implemented with Spark.

3 Dataset description

Our graph's vertices are Wikipedia pages (documents) and edges between them indicate that one page has a hyperlink to another.

Wikipedia network of top categories [1] is presented in several interrelated parts:

- Adjacency list. List of edges in the format `source_id`, `destination_id`. It means that a source page has a link to a destination page.
- Categories. The file provides the name of a category and corresponding document ids.
- Page names. This file associates a document's id and its title.

Each document can belong to multiple categories at once.

4 Implementation

In our implementation clusters are disjoint. That is, each document belongs to only one cluster.

4.1 Dataset preprocessing

The original dataset consists of 1,791,489 nodes and 28,511,807 edges. Processing it entirely is unviable on a single machine. We decided to shrink the dataset to make the computations on our laptop computers feasible. A graph built using a specified number of edges and vertices picked randomly or sequentially following the files' layout does not preserve a cluster structure of the original graph. That is why we use a different approach:

1. Randomly pick c categories
2. Take all the vertices that fall into these categories
3. Add all the chosen vertices, their adjacent vertices, and edges between them to our graph.

Using the above method, we guarantee the building of a sufficiently connected subgraph which inherits the cluster structure from the categories. Fixing the number of categories $c = 1200$ and *random seed* for reproducibility, we obtain the graph with 279,291 vertices and 1,539,027 edges which meets subgraph requirements from the task.

4.2 Label propagation algorithm

We roughly follow an algorithm described in "Near linear time algorithm to detect community structures in large-scale networks" [2]. It has following steps:

1. Initialize the labels at all nodes in the network. For a given node x, $C_x(0) = x$, where $C_x(t)$ is the label of node x at time t.
2. Set $t = 1$.
3. Arrange the nodes in the network in a random order and set it to X.
4. For each $x \in X$ chosen in that specific order, let $C_x(t) = f(C_{x_{i1}}(t), ..., C_{x_{im}}(t), C_{x_{i(m+1)}}(t-1), ..., C_{x_{ik}}(t-1))$. f here returns the label occurring with the highest frequency among neighbors and ties are broken uniformly randomly.
5. If every node has a label that the maximum number of their neighbors have, then stop the algorithm. Else, set $t = t + 1$ and go to (3).

Our implementation is different from the algorithm in a few ways:

- Step 3. It is synchronous. Node x at iteration t updates its label based on the labels of its neighbors at iteration $t - 1$. $C_x(t) = f(C_{x_1}(t-1), ..., C_{x_k}(t-1))$
- Step 2. There is no need to shuffle the nodes as we wait for all the nodes to be updated on the current time step.
- Step 5. We run our algorithm for a specified number of steps without the stopping criterion.

4.3 Implementation details

We took `org.apache.spark.graphx.lib.LabelPropagation` as a starting point for our implementation. We follow the idea of large-scale graph processing using the *Pregel* computation model with synchronous *LPA* implementation.

At first, we did not implement "breaking the ties" from step 4. It led to poor results: there were many very small clusters and after just a few iterations labels oscillated between the same nodes. "Breaking the ties" model proposes the random break of edges between a node and its neighbors. Theoretically, it eliminates the oscillation between algorithm iterations and connects the marginal elements into groups.

We also tried two options: undirected and directed edges. By default, label propagation algorithm assumes that edges are undirected.

5 Results

All the results below are presented for the same graph, which we obtained after shrinking the original one, with 279,291 vertices and 1,539,027 edges, built over 1200 initial categories. Table 1 contains cluster metrics for graphs obtained using different algorithm implementations and the metrics of the ground-truth graph.

Table 1: Metrics

		Implementation				
		Undirected		Directed		Graph
Metrics	True graph	No	Yes	No	Yes	"Breaking the ties"
Unfiltered clusters	1200	37667	32184	44975	43704	
Clusters	1200	4102	2860	3046	4001	
Min	4	2	2	2	2	
Max	8270	175505	73718	216650	57283	
Mean	264	60	87	78	60	
Median	150	3	3	2	2	
Standard deviation	480	2743	1748	3925	1035	
Execution time, sec	-	213	257	30	48	

Label propagation for a *directed* graph has a noticeably better performance time-wise but suffers from an inability to clusterize sparsely connected elements. Implementations of the approaches differ in `activeDirection` parameter in *Pregel*.

This paragraph describes findings for the undirected graph. Presence of "breaking the ties" decreases the largest cluster size more than twice. Number of clusters decreases by 1.4 times. "Breaking the ties" also leaves a smaller number of *Unfiltered clusters* of size one. It also increases the *Mean* cluster size, so it might be an evidence of a better clusterization.

For the directed graph random edges removal decreases the largest cluster almost four times. Number of clusters actually increases. We speculate both of these phenomena happen due to breaking ties of an already highly fragmented graph introduced by added directions to its edges.

5.1 Anomaly detection

Number of unfiltered clusters shows how many unique labels are left after the run of the algorithm. Number of clusters is a size of cluster space after filtering the *marginal* elements that do not belong to any constructed cluster. The biggest cluster of the maximal size usually represents a *hub*—a cluster containing

a large number of edges between dense groups. Usually it does not contain much sense and may be split into different meaningful sub-clusters.

To detect the hubs we represent each cluster in terms of a probability distribution with respect to underlying categories. In theory, the hub would be primarily described by a wide category, such as **Living people** with a relatively high probability and a set of sub-categories with low probability. The cluster 302755 from Listing 2 is an example of the hub. Its probability distribution is presented in section "Cluster category probability distribution" of Appendix.

5.2 Clusters interpretation

The following listings contain an interpretation of the constructed clusters with respect to the original categories, and a coverage metric—a fraction of the category elements covered by a specific cluster. The categories in the listings are sorted by their relevance to the specific cluster expressed in terms of percentage of cluster elements belonging to a category.

Listing 1 contains an analysis for two randomly chosen clusters built by the undirected *LPA* with no breaks of ties. The first cluster #662344 consisting of 2667 elements may be expressed in terms of its five most relevant categories. Analyzing the listing we may conclude that the cluster is a set of articles about Australian football. Particularly, it sufficiently covers the categories of three different Australian football team categories with the coverage metric more than 0.7. The second cluster #727244 contains articles about vulnerable Californian flora. It does not have a high coverage metric, so we conclude that this cluster is only a subset of the given categories and may be described by a specific flora variety.

Listing 2 is the same analysis but for the undirected algorithm with breaks of ties. The cluster #1296554 contains articles about rural communities. Cluster #302755 has a substantial size of 10749 elements. Its categories are not strongly correlated: it may be a consequence of a vulnerability of the "break the ties" approach. The algorithm covered poorly connected sub-clusters by intentionally breaking their neighborhood. The cluster itself may be interpreted as a set about florists dealing with vulnerable plants.

Listing 1: Cluster analysis for undirected algorithm with no breaks of ties

```
Given cluster:
Label: 662344, number of elements: 2667.0
Comparison to the categories:
(Category:Australian_rules_footballers_from_Victoria;, number of elements: 2843, coverage
    by cluster: 0.7801618)
(Category:Living_people;, number of elements: 418223, coverage by cluster: 0.0039954763)
(Category:Melbourne_Football_Club_players;, number of elements: 519, coverage by cluster:
    0.98651254)
(Category:Year_of_death_missing;, number of elements: 7851, coverage by cluster:
    0.053241625)
(Category:Carlton_Football_Club_players;, number of elements: 494, coverage by cluster:
    0.72672063)

Label: 727224, number of elements: 792.0
Relevant categories:
Comparison to the categories:
(Category:Vulnerable_plants;, number of elements: 1645, coverage by cluster: 0.27598783)
(Category:Flora_of_California;, number of elements: 1168, coverage by cluster: 0.1489726)
(Category:Flora_of_California_chaparral_and_woodlands;, number of elements: 644, coverage
    by cluster: 0.10403727)
(Category:Endemic_flora_of_California;, number of elements: 483, coverage by cluster:
    0.13250518)
(Category:Flora_of_Cuba;, number of elements: 172, coverage by cluster: 0.2616279)
```

Listing 2: Cluster analysis for bidirectional algorithm with breaks of ties

```
Given cluster:
Label: 1296554, number of elements: 2147.0
Comparison to the categories:
(Category:Villages_in_Grjec_County;, number of elements: 376, coverage by cluster:
    0.7898936)
(Category:Villages_in_Zgierz_County;, number of elements: 258, coverage by cluster:
    0.9147287)
(Category:Villages_in_Kutno_County;, number of elements: 352, coverage by cluster:
    0.6051136)
(Category:Bam_Province;, number of elements: 255, coverage by cluster: 0.7137255)
(Category:Populated_places_in_the_Centre-Nord_Region;, number of elements: 251, coverage by
     cluster: 0.7250996)

Given cluster:
Label: 302755, number of elements: 10749.0
Relevant categories:
Comparison to the categories:
(Category:Living_people;, number of elements: 418223, coverage by cluster: 0.0063363323)
(Category:Vulnerable_plants;, number of elements: 1645, coverage by cluster: 0.37082067)
(Category:Flora_of_California;, number of elements: 1168, coverage by cluster: 0.22431506)
(Category:Harvard_University_alumni;, number of elements: 6154, coverage by cluster:
    0.03639909)
(Category:Year_of_death_missing;, number of elements: 7851, coverage by cluster:
    0.026111323)
```

6 Additional tasks

6.1 Neo4j

We loaded our graph to Neo4j graph database using snippets below:

Listing 3: Vertices creation

```
LOAD CSV FROM 'file:///part-00001' AS line FIELDTERMINATOR '\#'
CREATE (node:Vertex {id: toInteger(line[0]), name: line[1], label: 'no'})
```

Listing 4: Edges creation

```
LOAD CSV FROM 'file:///part-00001' AS line FIELDTERMINATOR ',' MATCH (a:Vertex),(b:Vertex)
WHERE a.id = toInteger(line[0]) AND b.id = toInteger(line[1])
CREATE (a)-[r:r]->(b)
```

Listing 5: Label annotation

```
LOAD CSV FROM 'file:///part-00001' AS line FIELDTERMINATOR ',' MATCH (a:Vertex)
WHERE a.id = toInteger(line[0]) set a.label = toInteger(line[1])
```

6.2 Clusters correctness and similarity to original categories

To estimate the correctness of clusterization, we introduced the metric of *category coverage* by a cluster and the *cluster probability distribution* w.r.t. categories. Both metrics evaluations are described in section 5.2, and Appendix.

7 Conclusion

We reviewed and implemented the label propagation algorithm using Spark's GraphX library and specifically its Pregel graph computational model. We tried combinations of different approaches—directed/undirected edges and their random removal relative to a node—and estimated metrics for the resulting graphs.

References

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

[2] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks". In: 76.3, 036106 (Sept. 2007), p. 036106. DOI: `10.1103/PhysRevE.76.036106`. arXiv: `0709.2938 [physics.soc-ph]`.

[3] Satu Elisa Schaeffer. "Graph clustering". In: *Computer Science Review* 1.1 (2007), pp. 27–64. ISSN: 1574-0137. DOI: `https://doi.org/10.1016/j.cosrev.2007.05.001`. URL: `http://www.sciencedirect.com/science/article/pii/S1574013707000020`.

Appendix

Neo4j visualization

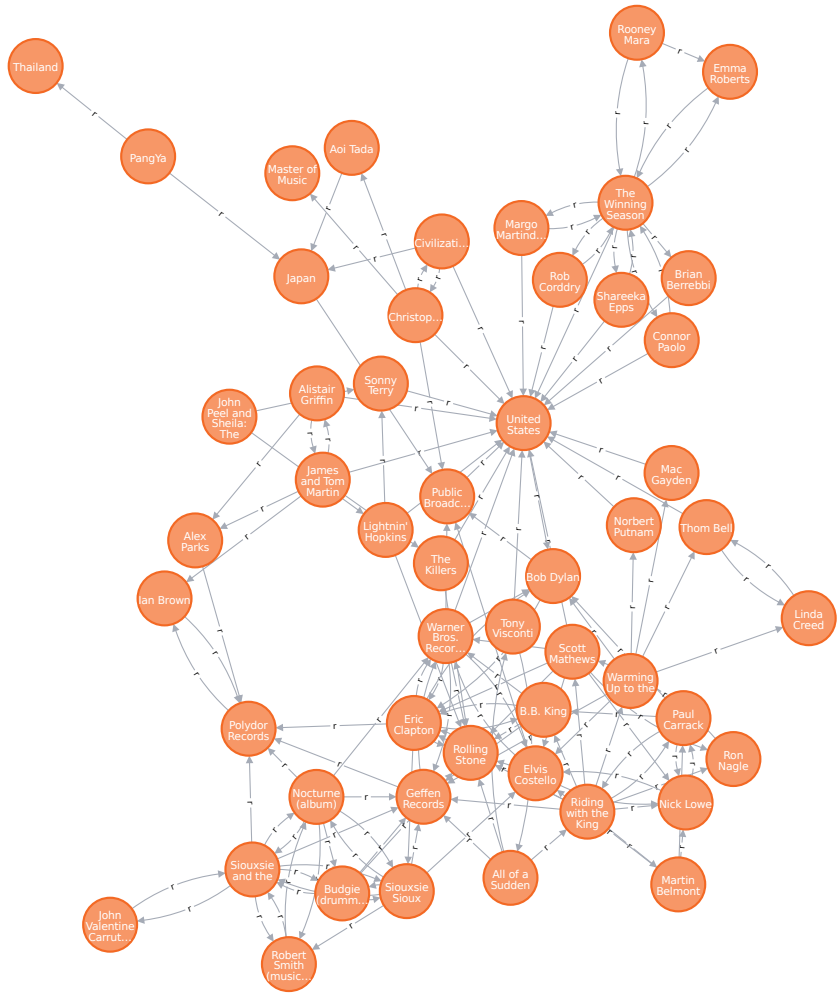

Figure 1: Part of the graph in Neo4j

Cluster category probability distribution

The listing contains the category probability distribution of cluster 302755 having 10749 between the categories presented in a list. The floating point number correspondent to each category is its relevance probability, calculated by collecting all the categories of cluster elements and reducing them by a key.

Listing 6: Probability distribution w.r.t. categories of cluster 302755

```
(302755,10749.0,List((Category:Living_people;,0.05946503904496583),
    (Category:Vulnerable_plants;,0.013688178799030696),
    (Category:Flora_of_California;,0.005879184992370504),
    (Category:Harvard_University_alumni;,0.005026478772103037),
    (Category:Year_of_death_missing;,0.004600125661969303),
    (Category:Municipalities_in_vila;,0.00415133291446011),
    (Category:Populated_places_in_vila;,0.00415133291446011),
    (Category:Year_of_birth_missing_(living_people);,0.004061574364958271),
    (Category:Recipients_of_the_Legion_of_Merit;,0.0039718158154564325),
    (Category:Rivers_of_Romania;,0.003859617628579133),
    (Category:United_States_Army_generals;,0.003792298716452752 5))
```