# **The application of learning algorithm** 2018

Course Project

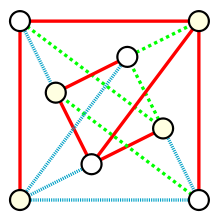
**Graph Edge Partitioning via Neighborhood Heuristic**

**Paper reference:**

*Zhang, Chenzi, et al. "Graph Edge Partitioning via Neighborhood Heuristic." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.‏*

**Submitted by:** ***ליעד נחמיאס***

***מיכאל חבקין***



**https://en.wikipedia.org/wiki/Matroid\_partitioning**

# Algorithm Name

Graph Edge Partitioning via Neighborhood Heuristic

# Reference

Zhang, Chenzi, et al. "Graph Edge Partitioning via Neighborhood Heuristic." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.

# Motivation

Graphs are widely used to represent relations between entities, for example social graphs, web graphs and IOT graphs. Nowadays graphs can grow rapidly to a large extent. Sometimes a graph is too large for a single computer to process it, and hence distributed graph engines are introduced to process these large graphs using cluster of computers. For these systems, a crucial step is to partition the graph to small parts and assign them to different computers. The part should be balanced enough, so that the to achieve maximum parallelism to avoid the scenario where one computer is waiting for another computer to complete its tasks. At the same time, we want to avoid cutting a large portion of the graph, because the network communication cost is high.

Instead of vertex partitioning, the paper proposes to use edge partitions, which partitions edges by cutting vertices, which serves the above-mentioned motivations, especially with graphs whose vertex degree follows power distribution. In addition, the edge partitioning produces smaller cut and more balanced partitioning.

One possible application of such algorithm is in some preprocessing steps of data mining or machine learning. For example, we want to execute some classification ensemble algorithm on a very large graph in a distributed manner, such that each machine receives an equal portion of the graph, while using minimum number of machines.

# Short Description

The paper proposes to partition a given graph iteratively in . Each round produces one edge set for a single computer from the un-assigned edge set that is left from previous iterations. The partitioning algorithm receives a constant the number of edges , and the number of iterations , and produces partitioning. The algorithm uses these main notations:

* Graph with vertices and edges.
* denotes the set of vertices adjacent to .
* denotes the set of vertices covered by .
* denotes the vertex set covered by .
* denotes the core set, from which the edge expansions begin.

The vertices are cut according to a new heuristic called NE (Neighbor Expansion). According to this heuristic, in each step of the algorithm one vertex is selected, such that its number of neighbors that are not in the S is minimal. The adjacent edges of are added to the current partition set , and is added to the core set . The expansion continuous until . The main objective of the algorithm is to produce a disjoint partitioning of its edge set into subsets .

# Pseudo-Code

:



:



:



# Algorithm Explanation

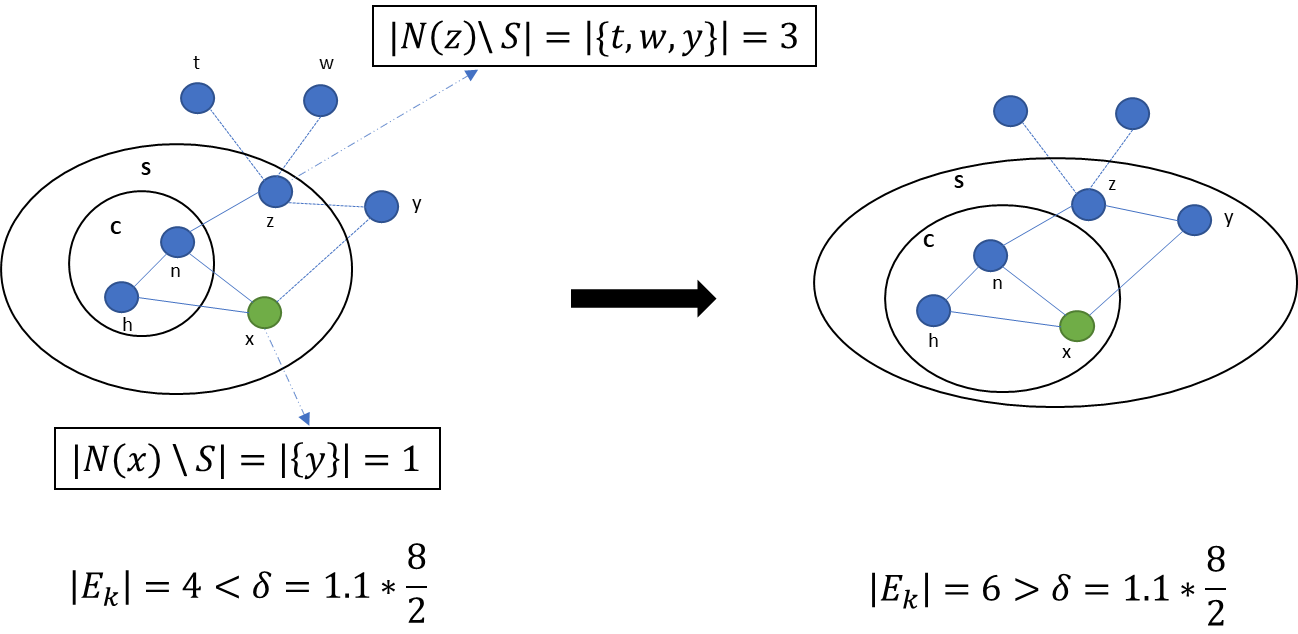
The operation is performed times in lines 3-4 of the main algorithm. In each round , edge set is selected from the so far un-assigned edges in graph , that is . In each iteration of the expansion process in lines 2-6 in the procedure, vertex , called the core vertex, is selected as the vertex which minimizes the number of vertices to be added to the boundary set , that is the vertex with the least amount of adjacent edges. In the first iteration of the expansion, all vertices of are candidates for the core vertex, and hence is selected randomly.

Then, the procedure in line 7 is called and all adjacent edges of are added to , in line 2 of that procedure. is added to the core set in line 1. Each allocated vertex such that is removed from the set of original edges , so that the number of un-assigned edges decreases. This process continues until the number vertices in partition satisfies in line 7 (, for number of original edges , number of partitions and a balance factor ).

# Illustration

The paper illustrates the main steps of the algorithm with an example of a single expansion of the edges. The graph is given in some iteration of the procedure. The value of is , for ).

We note that the edge set contains only the un-assigned edges, denoted as the dashed lines in the left image of figure 1. In this step one of the vertices and in the set should be selected for the next expansion. Two calculations are performed: the number of neighboring edges of and are counted, \ and \ . Since \ < \ , is selected as the next core vertex. The neighboring edges of , , are added to the boundary set and core set , and these removed from the un-assigned edge set, for the next steps. The vertex is added to the boundary set . The updated graph is illustrated in the right image of figure 1. The current partition is ready since its size is greater than .



**Figure 1** Illustration of single step of the expand procedure

# Strengths

1. Because each partition is balanced, that is the number of edges in the partition edge set is similar between partitions, network communication cost is reduced, since smaller chunks of the graph are sent over the communication line.
2. Respectively, the running time of each machine processing part of the graph is reduced, since the graph partitions are smaller.
3. In our opinion, the simplicity of the algorithm is another advantage. The expansion of the edges is intuitive and easy to implement.

# Drawbacks

1. The paper does not state explicitly what the data structure it uses to store the edges. In our implementation, we use python list to store the edges of each partition, but this solution might not be scalable for very large graphs. Hence, the paper leaves the efficient implementation for future work.
2. The NE (Neighbor Expansion) algorithm needs to be loaded the whole graph edge set in the main memory, while other state-of-the-art methods save only part of the data. Again, this can pose a problem for very large graphs.
3. In our opinion, it will be useful to save the edges in a hash-table structure for fast extraction and deletion. This modification will reduce the running time and reduce space complexity.

# Experimental Results

## Datasets

The paper uses undirected graphs from the Stanford large network dataset collection (available at <https://snap.stanford.edu/data/>). All the original experiments are run on super-computers, on very large graphs. Since our resources are very limited, we limit the size of the graphs to maximum of 500,000 edges (for graphs that exceed this number).

All datasets relate to undirected graphs, containing tuples of the form , which denote the edge between the vertices and . The vertices appear as the positive integer ID number.

The list of the datasets with their size are shown in table 1.

|  |  |  |
| --- | --- | --- |
| Dataset | Description | # of edges |
| Facebook1 | Social circles from Facebook (private sector) | 5038 |
| Facebook2 | Social circles from Facebook (business sector) | 69,820 |
| YouTube | YouTube online social network | 500,000 |
| Orkut | Orkut online social network | 5000 |
| LiveJournal | LiveJournal online social network | 500,000 |
| Friendster | Friendster online social network | 5000 |
| Amazon | Amazon product network | 500,000 |
| HepTh | Collaboration network of Arxiv High Energy Physics Theory | 51,971 |
| HepPh | Collaboration network of Arxiv High Energy Physics | 237,010 |
| GrQc | Collaboration network of Arxiv General Relativity | 28,980 |

## Measures

We use two main measures (as reported in the paper):

* Execution time - relates to the elapsed time from the reading of the data to the termination of all partition expansions. We measure this in seconds.
* Replication factor – The replication factor of a partition is defined as

## Hyper-parameters

We test the performance of the algorithm using two setups:

1. Changing the number of desired partitions .
2. Changing the workload balance .

## Baselines

## Results

# User Manual

The implemented code can be easily run using the python interpreter (python 2.7 should be installed on the computer). To execute the partitioning code, run the following command from your local CMD prompt:

python EdgePartitioner.py [edges\_file\_path] [alpha] [p]

* edges\_file\_path – The full path of the data file. The file should be in txt format delimited by one of the following delimiters: [,|;\t]. The file should contain one edge per line of the format <node\_1\_ID> <node\_2\_ID>.

For example, using the file “facebook.edges” and the hyper-parameters and , we execute the command: **python facebook.edges 1.1 30**

Possible output of such execution is:

| Summary of Results

| Replication Factor: 9.00

| Execution Time: 667.999983 miliseconds

In addition, each round of the partitioning produces the following output:

@@@@@@@@@ Started round #: 49 @@@@@@@@@

Start the Edge Partitioning Algorithm

!!!You've gone to a dead end. No more neighbors from this vertex!!!

The Edge Partitioning Algorithm - Successfully Finished

Results Report:

Gamma: 100

#Vertexes in C: 31

#Edges in Ek: 30

#Vertexes in S: 61

Number of Expands: 31