Evolutionary Community Discovery Using Minimum Description Length Principle

Anna Leontjeva, Irene Teinemaa

STACC Software Technology and Applications Competence Center

Abstract

In this work we explore an approach from the article “GraphScope: parameter-free mining of large time-evolving graphs” by Sun, J. et al. [1]. The goal is to detect communities in large datasets and discover change-points over time. We implement the proposed algorithm for dynamic community detection and evaluate its performance using several artificial and real data sets.

Introduction

A dynamic graph can be seen as a set of multiple graph states over time. Each of the states can be represented as an adjacency matrix and retained as a binary string. However, compression of this string is not optimal as the values are unevenly distributed. We can achieve better compression by reordering the matrix. The groups in the reordered graph correspond to communities, i.e. they contain nodes that are densely connected with each other.

The main idea behind the algorithm is to combine two consecutive states of the graph into one segment if there is a compression benefit. Communities are discovered by assigning each node to the partition that incurs the smallest encoding cost.

Methods

Algorithm 1: PartitionGraph

```
input  : graph, initial partitioning
output: new partitioning
// split
partition ← partition with largest average entropy
foreach node in partition do
    if average entropy of partition reduces without node then
        assign node to new partition
// update
foreach node in sourceNodes do
    assign node to partition with minimal cost
// merge
foreach partition pair (p1, p2) do
    if total encoding cost decreases when p1 merged with p2 then
        merge (p1, p2)
```

Algorithm 2: SegmentGraphStream

```
input  : segment, new graph
newGraphCost ← totalCost(newGraph);
segmentCost ← totalCost(segment);
unionCost ← totalCost(segment ∪ newGraph);
if unionCost- segmentCost <newGraphCost then
    segment ← segment ∪ newGraph
```

The cost of assigning a node to a partition is calculated as

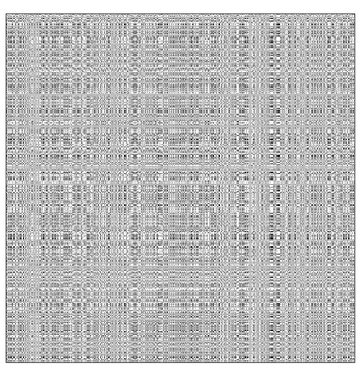
$$C(\mathbf{p}, \mathbf{q}) = (t_{s+1} - t_s)m \sum_{i=1}^{\ell} H(\mathbf{p}_i, \mathbf{q}_i)$$

where m is the number of nodes, l is the number of partitions and H(p,q) is the cross-entropy of the edge distribution of the node (p) and partition (q).

The total encoding cost of a graph depends on the entropy of partitioning distribution

$$H\left(\mathcal{G}_{p,q}^{(s)}\right) = -\left(\rho_{p,q}^{(s)} \log \rho_{p,q}^{(s)} + \left(1 - \rho_{p,q}^{(s)}\right) \log \left(1 - \rho_{p,q}^{(s)}\right)\right)$$

Results



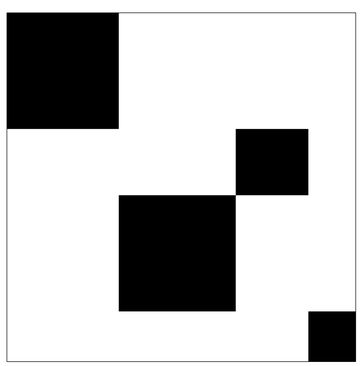


Figure 1. An artificial graph with full bipartite subgraphs. Left figure illustrates the initial adjacency matrix and right figure the matrix reordered by GraphScope.

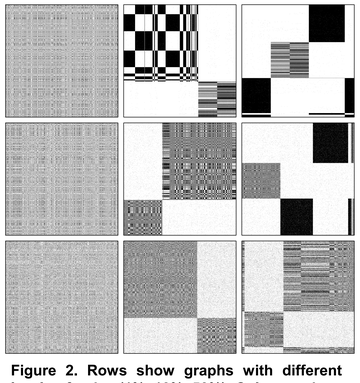


Figure 2. Rows show graphs with different levels of noise (1%, 10%, 50%). Columns show initial graphs and two iterations of graphScope respectively.

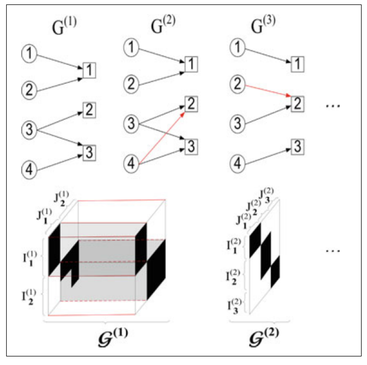


Figure 3. A bipartite graph stream with three graphs in two segments. Figure originated from [1].

The GraphScope method assumes a bipartite graph with separate groups of source and destination nodes. We experimented with artificial graphs of 1000 source and 1000 destination nodes and ~ 279K edges. The graphs were constructed in such a way that there are clearly distinguishing modules or (nearly) complete subgraphs. The data was distorted with different levels of noise (from 0 to 50%).

With few iterations the algorithm was able to partition the nodes into dense modules. However, the performance of the algorithm decreased when the amount of noise was increased.

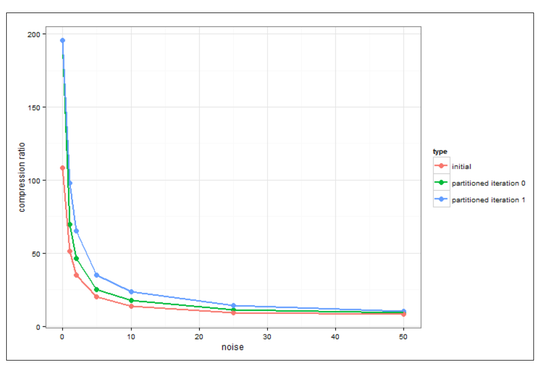


Figure 4. Compression ratios (initial file size divided by compressed file size) of the initial graph file and partitioned graph with 1 and 2 iterations of GraphScope. 7-zip was used for compression.

After partitioning, the graph was retained as homogenous subgraphs (close to either fully connected or fully disconnected modules), which enabled higher compression than the initial graph.

Namely, we created the initial graph files that contain the number of source nodes (m), destination nodes (n) and the adjacency matrix in the format of a binary string. Additionally, we created the partitioned graph files, containing m, n, the number of source partitions (k), destination partitions (l), membership of each node and k * l submatrices.

Therefore, the total encoding cost of the partitioned graph can be calculated as the sum of the partition encoding cost

$$C_p^{(s)} := \log^\star m + \log^\star n + \log^\star k_s + \log^\star \ell_s + mH(P) + nH(Q),$$

and graph encoding cost

$$C_g^{(s)} := \sum_{p=1}^{k_s} \sum_{q=1}^{\ell_s} \left(|E|_{p,q}^{(s)} + |\mathcal{G}_{p,q}^{(s)}| \cdot H\left(\mathcal{G}_{p,q}^{(s)}\right)\right),$$

Both the initial and partitioned graph files were compressed with 7-zip compression tool. Figure 4 shows that the partitioned graph files enabled better compression than the initial graph file.

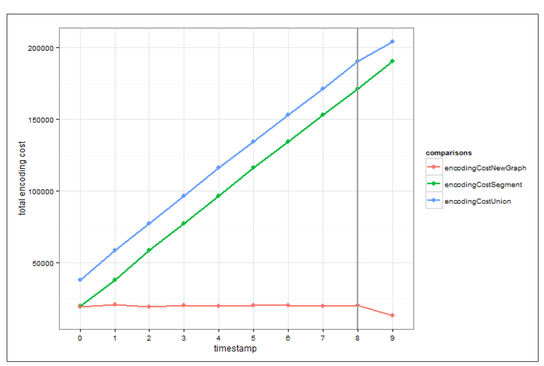


Figure 5. Encoding cost of graph streams across 11 time periods on a real social network. GraphScope detected two segments of graphs, or one change-point.

For change-point detection analysis we collected a subset of a real social network where source nodes are users and destination nodes are countries. There is an edge between a user and a country if that user added at least one friend from this country in a particular month. The sample contains 1000 users and 264 countries with 40537 edges divided between 11 different time periods.

The explored approach could potentially reveal the users' level of internationality and clusters of similar users in that regard. Unfortunately, the GraphScope algorithm was not able to produce meaningful results, discovering partitionings with only a very small compression gain. The final output had two segments, with 10 and 1 graphs respectively.

Conclusion

The idea behind GraphScope method seems to be very appealing. However, one of the major drawbacks is its inability to scale to large number of nodes. The method seems to be suitable for data sets with relatively small number of nodes and frequent updates.

References

1. Sun, Jimeng, et al. "GraphScope: parameter-free mining of large time-evolving graphs." Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007.
2. Code is available at https://github.com/irhete/graphScope