Application of HPC in graphs community detection

Cong Yu

February 23, 2020

1 Problem Introduction

In graph theory, community detection, also known as graph clustering is to partition nodes of the graph into small subsets (or communities) based on certain criteria. One intuition behind this method is that nodes in the same cluster or module may share similar properties or patterns so we can better study the complex network.

Therefore, this method can be applied in many disciplines whose problems can be modeled as networks. Such as in society, people natrurally form different groups like families, organizations, and countries. Also, web resources in the same cluster may contain related or even the same topics. Another example from biology is that in protein-protein interaction, one protein is prone to behave in cells similarly. [1]

2 Needs for HPC

In recent years, this method has been applied in increasingly complex networks such as social networks. Despite polynomial time complexity of mainstream algorithms, the computation time of most algorithms is up to about 10 seconds when the numbers of nodes in the graph reach 20000.

However, a large graph like the followed-by network in Facebook is about 200 million in the US alone. [4] Thus, the cost for a large graph like that would be intolerable in most cases. Even worse, such a large network is almost impossible to store on a single machine.

Hence, the community-detection problem must be solved by parallel, reliable and scalable algorithms where the high-performance computing comes to play.

3 Algorithms

There are many algorithms to do this job. Among all the algorithms, Label Propagation (LP) algorithm [3] is the fasted, most widely used in real practice and also suited to be parallelized.

The basic process of this algorithm goes as follow:

1. Every node in the graph has a unique label.

2. For each iteration, the label is updated by the majority of the neighbors.

2.1 If there is a tie among the choices, a winner is chosen randomly.

3. The algorithm converges when every node has a label that at least half of its neighbors hold.

4 Performance and Scalability

The time complexity of the label propagation algorithm is near-linear [3], and the actual computation takes less than 0.1s for 20000 nodes on a single machine [4].

Furthermore, this method can be easily parallelized since each update for nodes would only require data from their neighbors and could be calculated separately.

The map-reduce version of this algorithm from the paper [2] goes as follow:

```
Input:
    input graph G_in (edge based)
    number of propagation steps p
    number of label instance k
```

```
Output:
    vertex-core group mapping M
    output graph G_out

Procedure:
passed_labels = ReadMap (G, k)
for i = 1 to p - 1
    new_labels = PropagateReduce(passed_labels)
    passed_labels = PropagateMap(new_labels)
final_labels = CoreGroupsExportReduce(newlabels, M)
final_labels = EdgeListPreparationMap(final_labels)
core_group_links = EdgeListPreparationReduce(final_labels)
core_group_links = EdgeListExportMap(coregrouplinks)
EdgeListExportReduce(M, G_out)
```

References

[1] Santo Fortunato. “Community detection in graphs”. In: *Physics reports* 486.3-5 (2010), pp. 75–174.

[2] Michael Ovelgönne. “Distributed community detection in web-scale networks”. In: *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*. IEEE. 2013, pp. 66–73.

[3] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. “Near linear time algorithm to detect community structures in large-scale networks”. In: *Physical review E* 76.3 (2007), p. 036106.

[4] Zhao Yang, René Algesheimer, and Claudio J Tessone. “A comparative analysis of community detection algorithms on artificial networks”. In: *Scientific reports* 6 (2016), p. 30750.