Graph Partitioning Algorithm with LSH

Poster extended abstract

Weidong Zhang*, Mingyue Zhang†
EECS of Peking University, Beijing 100871, P.R.China
{zhangwd*, mingyuezhang†}@pku.edu.cn

***Abstract*—In this paper, we propose a graph partitioning algorithm which is based on the locality-sensitive hashing (LSH). Its time/space complexity is O(n), n is the number of vertices in graph. For all kinds of hyperscale graphs, it works at the speed of random partitioning method approximately. Compared with the mainstream graph partitioning algorithms, the new algorithm owns a simpler processing pipeline and avoids irregular memory access generated by graph traversals. The experimental results show that the new algorithm achieves 10x faster than Metis and 2x faster than label propagation algorithm at the cost of a reasonable precision loss.**

***Index Terms*—Graph Partitioning Algorithm, Locality-sensitive Hashing**

I. Introduction

Many effectual graph partitioning algorithms have been proposed in recent decades, such as Metis and label propagation algorithm (LPA). These methods are good options for the small-to-medium-sized graphs, but when the scale of graph increases to ten million vertices, the existing methods hardly could work out the final results or take a prohibitively cost. In this paper, we propose the locality-sensitive hashing (LSH) based graph partitioning algorithm whose time/space complexity is O(n), n is the number of vertices in graph. To provide an overview, it is compared with some more algorithms in four aspects.

TABLE I: Compare our algorithm with the common graph partitioning algorithms in four aspects:time/space complexities; quality of partitioning result; the pattern of memory access.

Algorithm	Time	Space	Result	Mem access
Random Partition	O(n)	O(m)	poor	regular
Kernighan-Lin	O(n^3) [1]	O(m)	good	irregular
Spectral	$O(n^3)$	$O(n^2)$	good	irregular
LSH-based	O(n)	O(m)	good	regular
Metis	$O(m\log(n/n'))$[2]	O(m)	excellent	irregular
LPA	$O(n+m)+O(nlog_2(n))$	O(m)	excellent	irregular

II. Models and Algorithms

Resorting to LSH, the adjacent vertices are first mapped into the adjacent locations in hash space. Then through partitioning the hash space, a roughly partitioned result is obtained. Three key steps are involved during the processing:

- Choose an appropriate vertex presentation which not only expresses the similarity of adjacent vertices, but also delivers sufficient distinctiveness of non-adjacent vertices.
- Choose an appropriate locality-sensitive hash function cluster.
- Design a suitable hash space partitioning method.

A. Vertex numbering and presentation

The goal is to find a presentation carrying sufficient information to convey:

- Adequate similarity among adjacent vertices.
- Decreasing similarities as the vertices' distance increases gradually.

Here, we use the Jaccard distance to measure the similarity of two vertices.

1) Vertex Numbering: Inspired by text retrieval, we attempt the following schemes to relpace the numbers to encode vertices: (i)use a letter to encode a number; (ii) use multiple distinct letters or digits to encode a number, etc.

2) Vertex Presentation: In order to bring in more positional information and increase the similarity as the vertices' distance decrease, we design several strategies to present the vertices.

Tree-like/Start-like presentation: Vertices are expressed as a set of vertices, which are located in the scope of a tree/star structure with the current vertex as the root node. The location information of vertex can be adjusted by the depth/radius of the tree/star.

B. Mapping adjacent vertices to hash space

As is shown in Figure 1, we first express the graph vertices as text documents, then map them into the hash space by LSH. The hash space of LSH can be considered as a high-dimensional vector space. In spite of the fact that items with low similarities cannot be mapped to the same hash value, partial dimensions of their hash values still share certain similarities. This is the reason why LSH Forest and Multi-prob LSH work.

C. Indexing hash value by kd-tree

Through the mapping of LSH, the adjacent vertices are mapped into the adjacent buckets in hash table, then divide the hash space, we could get a roughly partitioning result. Unfortunately, the LSH's value space is still a high-dimensional vector space, it is hard to partition the

2168-9253/18/$31.00 ©2018 IEEE
DOI 10.1109/CLUSTER.2018.00053

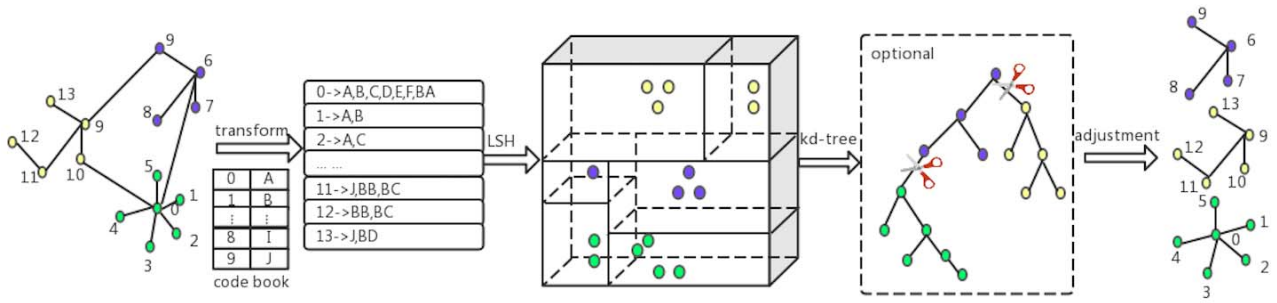

Fig. 1: Pipeline of LSH-based partitioning algorithm

high-dimensional space according to the number of data items. So we further index the hash values with a tree-like structure index. By dividing the linear sequence of the inorder traversal result of the index tree, a preliminary partitioning result can be obtained.

D. Adjusting the preliminary result

Except the first hash table which is used to create kd-tree, the elements in the rest hash tables will be used as the baseline for adjustment in this step.

In the experiment stage, we find that even without indexing by and partitioning on kd-tree, a forthright adjustment on LSH's result can achieve a approximate result to the former. In the following section, the implementation of the LSH-based partitioning algorithm simply consists of vertex presenting, LSH mapping and adjusting.

III. Evaluation and Result Analysis

We compare our algorithm with Metis and LPA mainly in three aspects including: cut edge, balance and partitioning time.

The performance on the graphs with less than one million vertices, is shown as in Figure 2(a,b): the new algorithm slashes the cut edges by 95% from the result of random partitioning algorithm, but still generates 7x-15x cut edges than Metis and LPA. It runs 1.5x-3x faster than Metis and LPA, and almost as fast as random partitioning algorithm.

The performance on the graphs with more than one million vertices, is shown as in Figure 2(c,d): the new algorithm reduces the cut edges by 83% from the result of random partitioning algorithm, and generates about 33% more cute edges than LPA. It works 2.8x-10x faster than LPA, and 30% slower than random partitioning algorithm.

For the balance among partitions, the new algorithm performs the best among these algorithms. As the number of partitions increases, the number of cut edges also tends to increase for all the algorithms. The new algorithm presents the slowest increment speed among them.

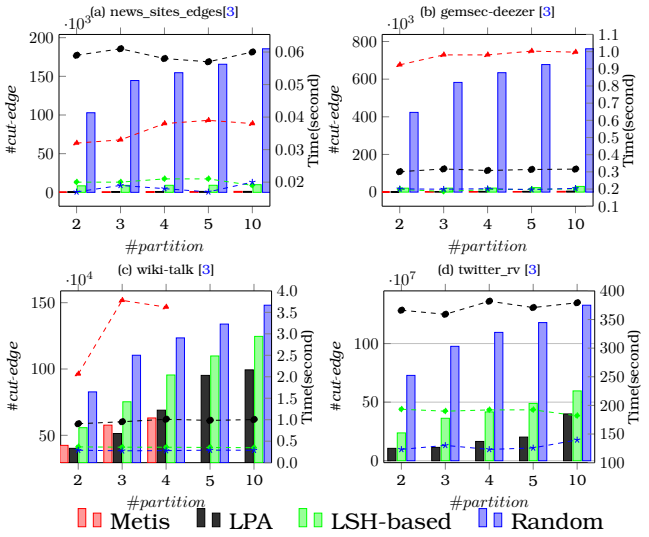

Fig. 2: Partitioning result. The histograms demostrate the numbers of cut edges between partitions, the wave lines denote the partitioning time.

IV. Conclusion

The LSH-based graph partitioning algorithm proposed in this paper owns a very low time complexity of O(n), the partitioning speed is close to that of random partitioning algorithm. For various large scale graphs, the quality of partitioning result is close to that of high-complexity algorithms. At the same time, we find that the strategy of adjustment plays a key role to the quality of the final result. In the future work, more adjustment strategies for different types of graphs will be tested and evaluated to improve the partitioning result.

References

[1] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst.tec.j*, vol. 49, no. 2, pp. 291–307, 1970.

[2] G. Karypis and V. Kumar, "Metis: a software package for partitioning unstructured graphs," *International Cryogenics Monograph*, pp. pags. 121–124, 1998.

[3] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data, June 2014.