Qualifying Exam Part II
Doctor of Philosophy in Computer Science

By: Reet Barik

August 11, 2020

Abstract

For most parallel graph-theoretic applications, data movement has become a primary bottleneck toward achieving scale in computing. One of the primary contributors to data movement is how the graph is stored or represented in memory. To this end, various vertex reordering schemes have been studied in the past. MinHashing has been a popular way in estimating the similarity between sets. Its use of 'sketches' makes it a powerful data analysis tool. As a result it has the potential to make for a good candidate as a solution for the vertex reordering problem.

1 Introduction

In graph processing algorithms, CPU cache performance is plagued by efficiency issues, so much so that almost half the processing time comprises of cache miss latency [1]. The problem of poor data locality is caused by irregular memory access. Real world graphs present an opportunity to improve the locality of graph applications because of the presence of clusters or hubs. One solution is to come up with something called 'vertex reordering' which is an optimal permutation among all nodes wherein, those that are frequently accessed together, are stored locally in the memory.

Of the vertex reordering schemes around, the Minimum Linear Arrangement (**MinLA**) [17] formulates the problem as one of optimization where the objective is to minimize the sum of the linear arrangement gaps (difference in ranks of vertices which share an edge). The log variant of it, namely the Minimum Logarithmic Arrangement (**MinLogA**) is geared more towards graph compression [8] and is relevant for graph storage purposes. There are various lightweight degree based reordering techniques like **Degree Sort** which is based on the objective of assigning consecutive IDs to vertices with the highest degrees such that they fit into the same cache line. Minor variants of such a degree based approach are **Hub Sort** [19] and **Hub Clustering** [3] which attempts to take advantage of the power law degree distribution of real world graphs. **Slashburn** [12] is another degree based scheme which identifies 'caveman communities' greedily. The current state of the art **Gorder** [18] uses a window based solution and tries to preserve neighborhoods by looking at neighbor and sibling relationships between vertices. Some moderately lightweight approaches are based on community detection like **Rabbit-order** [2] which attempts to map the hierarchical community structures present in graphs to the cache hierarchy. There have also been attempts to reduce the fill of the adjacency matrices of input graphs like Reverse Cuthill-McKee **RCM** [9] and **Nested Dissection** [10].

MinHash or the min-wise independent permutations locality sensitive hashing scheme, was first introduced [6] as a way to estimate the similarity between sets. Numerous variants of it have been proposed down the years like [7]. The intuition behind the technique is that hash functions can be used to map large sets of objects to smaller ones in such a way that if two objects are close to each other semantically, then their hash values are also likely to be the same. MinHash has a lot of applications, some of which are clustering, plagiarism detection [7], eliminating near-duplicates among web documents [15], etc. It has also found widespread application in the field of bioinformatics where MinHash based algorithms like [16], [5], and [13] have found success in the problem of genome alignment and assembly.

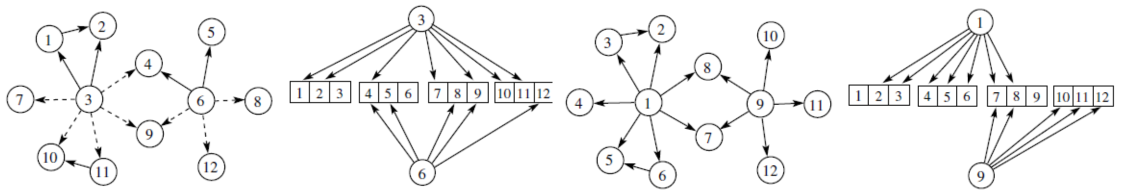
This report is an attempt to try and come up with ideas/ways to solve the vertex reordering problem by using the various transferable skills of the MinHash technique.

2 Background

This section attempts to give a general overview of vertex reordering followed by an overview of the MinHash technique.

2.1 Vertex Reordering Overview

Vertex reordering has long been a popular optimization to reduce cache latency for various graph applications. The simple example shown in Figure 1, taken from [18], shows how ordering fairs significantly better at maintaining locality than naive partitioning in case of real graphs.



(a) Partitioning without vertex re-ordering (b) Partitioning with vertex re-ordering

Figure 1: An illustration of the impact of vertex re-ordering on block partitioning of vertices.

In sub-figure (a), the edges shown as dashed lines are the edges that cross two partitions to be cut (with a total of four partitions). It can also be seen that if an algorithm accesses the out-neighbors of node 3, it needs to access all four partitions which are stored across four CPU cache lines. Similarly, for node 6, there is a need to access three partitions stored across three cache lines. Sub-figure (b) shows the partitioning after reordering. If an algorithm tries to access the out-neighbors of node 1 (topologically the same node as node 3 from the previous case), it needs to access three partitions which are stored across three CPU cache lines. Similarly, for node 9 (topologically the same node as node 6 from the previous case), there is a need to access just two partitions stored across two cache lines. The reduction in latency from naive partitioning to ordering is quite apparent.

2.2 MinHash Overview

The MinHash technique is used to estimate the similarity between sets. The similarity here, is given by the metric called **'Jaccard Coefficient'**. For two sets X and Y, it is given by:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

It is essentially the ratio of the cardinality of the intersection of two sets with that of their union. It follows from the definition that $0 \leq J(X,Y) \leq 1$.

There are two ways the MinHash technique can be carried out:

- **Using k minimum values from a single hash function**: Given two sets X and Y such that high cardinalities of the two sets make the computation of $J(X,Y)$ very expensive,

  1. A universal hash function H is applied on all the elements of X and Y.
  2. After sorting, the top-k elements are obtained from X and Y. These are called *Sketches* of the sets X and Y and denoted by $S(X)$ and $S(Y)$.
  3. Since $J(S(X),S(Y)) \approx J(X,Y)$, calculating the Jaccard Coefficient of the Sketches of X and Y gives us a good estimate of the actual similarity between the two sets.

- **Using k hash functions**:

  1. k universal hash functions are applied to the elements of X and Y.
  2. Sketches $S(X)$ and $S(Y)$ are obtained by collecting the top elements from each $H_i(X)$ and $H_i(Y)$ where $1 \leq i \leq k$, such that $|S(X)| = |S(Y)| = k$.
  3. Compute $J(S(X),S(Y))$ to estimate the Jaccard similarity between X and Y.

Here, k which has a value between 1 and $min(|X|,|Y|)$ is a hyperparameter that can be tuned. Larger the k, more accurate will be the estimation of the Jaccard Coefficient.

3 Using MinHash for Vertex Reordering

This section's objective is to suggest ideas/ways how the MinHash technique can be used to solve the problem of vertex reordering. What follows are two ways suggested by the author on how that might be done.

3.1 Preserving graph neighborhoods via MinHash

Neighborhood or locality preservation is one of the main aims of vertex reordering. If there could be a way to extract neighborhoods of a certain size from an input graph, MinHash could be used to estimate the similarities between such neighborhoods and label the vertices of similar neighborhoods in a way that their IDs are closer to each other in the vertex array. Keeping this approach in mind, the following steps could be carried out for vertex reordering of an input graph using MinHash:

1. In a graph $G(V,E)$ where V is the set of vertices, and E is the set of edges, $\forall v \in V$, do a BFS from v going up to a depth d, where d is a hyperparamter than can be tuned.

   The neighborhood of a vertex v is defined by the set $S(v)$ which contains the vertices visited by the above mentioned BFS traversal.

2. Use MinHash to estimate the similarity between these sets which implicitly gives an estimate of the similarity between the neighborhoods that these sets represent.

3. Greedily order these sets based on the estimated Jaccard coefficient of each pair. This has been illustrated in Figure 2 below. Here, $J1 \approx J(A,D)$, $J2 \approx J(D,B)$, $J3 \approx J(C,B)$, and $J4 \approx J(A,C)$ such that, $J1 > J2 > J3 > J4$. This leads to the ordering where A is followed by D, B, and then C.

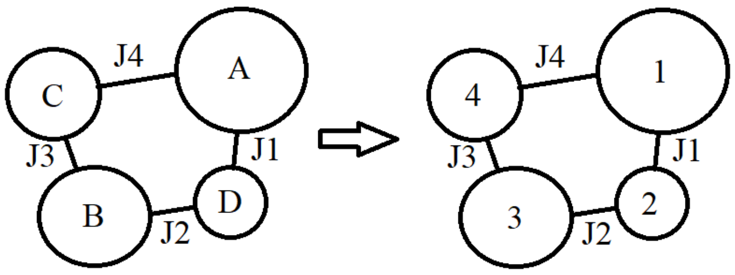


Figure 2: Greedy ordering of sets based on the pair-wise Jaccard coefficient.

4. The ordering of nodes within each set can be done conserving the BFS traversal order. This further helps to preserve the smaller neighborhoods while reordering inside the bigger sets.

3.2 Using MinHash instead of Community Detection

It has been observed as part of the work from a paper [4] under review that community detection based reordering techniques are much more adept at generating orderings that reduce the average linear arrangement score mentioned in [17]. Keeping that in mind, MinHash could be used instead of relatively heavyweight community detection methods to generate the orderings. The following is a way how that might be achieved:

1. In a graph $G(V,E)$ where V is the set of vertices, and E is the set of edges, $\forall v \in V$, do a random walk of length l, where l is a hyperparameter than can be tuned. Inspiration can be taken from 'Node2vec' [11], a random walk based graph algorithm to generate node embeddings while carrying out these walks. The random walk can be parameterized as follows.

   - During a random walk when a transition is made from a vertex u to v, the parameter q could be used to generate the probability of making the transition back from v to u.
   - During a random walk, the parameter p can be used to influence whether the transitions made one hop away will be more in a BFS, or a DFS fashion.

   Here, the random walks must be sufficiently long (l must be large enough) so that vertex coverage is high.

2. Each random walk gives a set of vertices (large enough to make MinHash not be an overkill). These sets can be put through the MinHash treatment so as to cluster the random walks together based on similarity.

3. For pairs of sets representing random walks sharing an estimated Jaccard Coeffient higher than a set threshold, the intersection of such sets have a high probability of having vertices that might otherwise have belonged to the same community generated by a traditional community detection algorithms like [14].

4. After identification of communities, they might be ordered as illustrated in Figure 2. The only difference being, instead of using the Jaccard coefficient, one can use the number of inter-community edges as a way to determine the strength of connection between one community and the other.

5. As far as the intra-community vertex ordering is concerned, the natural order could be preserved, or each community could be used as an input graph to this whole process in a recursive fashion.

4 Discussion

The previous section was an attempt at using MinHash to address the problem of vertex reordering. The following is a discussion of some of the merits of the suggested methods:

- The approach suggested in Section 3.1 consists of a BFS traversal from each node of the input graph to generate the neighborhood. The advantages of this is:

  1. Since the BFS traversals are mutually exclusive for each vertex (because they are read-only), this step is highly parallelizable.

2. Though the size of the sets representing the neighborhoods might be large, using MinHash significantly brings down the complexity of pair-wise comparison to estimate similarity.

- The method suggested in Section 3.2 consists of doing random walks from each vertex of an input graph.

  1. Similar to the other approach suggested, this is a highly parallelizabe step (capped by the total number of threads available) and MinHash makes the pair-wise comparison of the sets representing the random walks significantly cheaper.
  2. The parameterized way to generate the random walks presents the advantage of having the opportunity to tune them based on the end graph-application. For example, if the end application is something like BFS, or Local Triangle Counting, then the random walks could be tuned to follow a more BFS way of traversal and vice versa.

5 Conclusion

To summarize, this report first takes a look at vertex reordering and MinHash separately. This is followed by the description of two suggested approaches that uses the various transferable skills of MinHash to solve the problem of vertex reordering. What follows is a brief discussion that speaks to the merits of the two suggested approcahes in terms of parallelizability and adaptability to applications. In the end, it can be said that given the inexpensive nature of MinHash and its capability of estimating the similarity between large sets of objects, using it in the field of vertex reordering remains a very fertile scope of research.

References

[1] Anastassia Ailamaki, David J DeWitt, Mark D Hill, and David A Wood. Dbmss on a modern processor: Where does time go? In *VLDB'99, Proceedings of 25th International Conference*

on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK, number CONF, pages 266–277, 1999.

[2] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. Rabbit order: Just-in-time parallel reordering for fast graph analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 22–31. IEEE, 2016.

[3] Vignesh Balaji and Brandon Lucia. When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 203–214. IEEE, 2018.

[4] Reet Barik, Marco Minutoli, Mahantesh Halappanavar, Nathan Tallent, and Anantharaman Kalyanaraman. Vertex reordering for real-world graphs and applications: An empirical evaluation. *Submitted to IEEE International Symposium on Workload Characterization*, 2020.

[5] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623–630, 2015.

[6] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[7] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[8] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228, 2009.

[9] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, 1969.

[10] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[11] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[12] U Kang and Christos Faloutsos. Beyond'caveman communities': Hubs and spokes for graph compression and mining. In *2011 IEEE 11th International Conference on Data Mining*, pages 300–309. IEEE, 2011.

[13] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, 2017.

[14] Hao Lu, Mahantesh Halappanavar, and Ananth Kalyanaraman. Parallel heuristics for scalable community detection. *Parallel Computing*, 47:19–37, 2015.

[15] Mark S Manasse. On the efficient determination of most near neighbors: horseshoes, hand grenades, web search and other situations when close is close enough. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 7(5):1–100, 2015.

[16] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):132, 2016.

[17] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics (JEA)*, 8, 2003.

[18] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. Speedup graph processing by graph ordering. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1813–1828, 2016.

[19] Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Matei Zaharia, and Saman Amarasinghe. Optimizing cache performance for graph analytics. *arXiv preprint arXiv:1608.01362*, 2016.