Clustering for Graph Partitioning

Andrea Smith

May 3, 2018

About the Author

The author has earned her Bachelor's of Science degree in Computer Science from Missouri University of Science and Technology. She is working towards her doctorate degree in Computer Science, with a focus on graph data mining, from the same university, anticipating completing 2022.

Contents

1 Executive Summary

This project aims to explore the potential of clustering algorithms for aiding in the partitioning of large graphs for frequent subgraph mining. Partitioning is important as many graphs of interest cannot be held in the memory of a single machine, and so must be spread across many machines in order to be processed. Partitioning risks losing data that spans across multiple partitions, requiring an intelligent partitioning process in order to minimize.

This project explores k-means clustering, distance-k clustering, and multilevel kernel clustering algorithms, as they apply to partitioning large transaction graphs in data mining. The clusters produced by the algorithms are evaluated for quality based on cluster density and the maximal shortest path within the cluster. The algorithms were tested on a toy data set built from publicly available Arabian horse pedigree papers, such that each edge represents a child-parent relationship. The algorithms were written for undirected, weighted graphs, but could be modified for directed graphs.

Further research needs to be done to create and evaluate parallelized or distributed versions of these algorithms, so that they can process graphs that do not fit within the memory of a single machine.

2 Introduction

Stuff to say [1, 2].

3 Project Specifications

This project focuses on comparing the quality of the clusters produced by each algorithm. Quality is a combined score of the cluster density and the maximum shortest path within the cluster. By comparing the clustering ability of the algorithms on a small graph, inferences about the quality of partitions made on a larger transaction graph can be made. Note that further research should be done into parallelized and distributed versions of these algorithms, but that is beyond the scope of this project.

All code was written in the C++ programming language. The program accepts as input a properly formatted document file that describes the graph to execute. The program outputs the basic statistics of the clusters formed by each of the three algorithms, as well as a list of which nodes are in which cluster. The statistics reported include the number of clusters formed, the number of elements in each cluster, the density of each cluster, and the diameter of each cluster.

In the future, the program should be modified to accept a parameter file that can affect the execution of the program. Parameters of interest would include the number of clusters that the k-means algorithm should search for, and the sigma value for the kernel clustering algorithm.

4 Detailed Design

The program was designed in a combination of the object-oriented method and the functional method of programming.

4.1 Class Objects

The driving force of the program is a UndirectedGraph class, which is built from a SymmetricalMatrix class. The data is represented as an undirected graph because the project is looking for relationships between the horses, and we do not care about which horse is the parent. This then allows us to use a SymmetricalMatrix class, which is much more memory efficient to use than a full matrix class would be, as only either the upper or lower triangle of the matrix actually needs to be stored.

While C++ has a basic vector class, it does not have any linear algebra functions attached to that class. Instead, a custom vector class was developed for use, which the basic Matrix class was then built on top of. This combination allows for full flexibility to implement any of the needed linear algebra operations directly into the class. This was designed to be used by a spectral clustering algorithm, but that algorithm has instead been removed from the scope of this project. The SymmetricalMatrix class inherits from the Matrix class, but was given more efficient memory management, to take advantage of the fact the half of the values in the matrix are identical. A simple UML diagram for the relationship between the custom classes exists in Appendix A.

The UndirectedGraph class includes a SymmetricalMatrix object as a private member. This matrix is used to store the adjacency matrix of the undirected graph. The advantage to this method is realized in very dense graphs, as it becomes more efficient to store and to work with than other representations, which often involve long lists of node id pairs. Unfortunately, the graph used in the testing was not a particularly dense graph, and so this particular benefit did not come to light. However, dense graphs are a prioritized area of interest in the data mining field, and so it was important to consider that beyond this project.

The C++ data type std::set was chosen to represent the clusters within the functional algorithms. The class type has a strict enforcement of no duplication within a set and supports simple insertion, deletion, and search operators. Each node was represented in the cluster by it's id in the original graph, not by it's string label, which is more efficient as string operations are cumbersome and slow.

4.2 Algorithms

4.2.1 K-Means

The k-means function was broken into five different sub-functions. This helps improve readability of the code and allowed some sections to be reused for the other algorithms. The pseudo-code for the k-means algorithm is represented in Appendix B and the pseudo-code for the k-means centroid initialization is presented in Appendix C [3].

4.2.2 Distance-K

4.2.3 Kernel Clustering

Kernel clustering is done in three steps. The first step is coarsening the data. This step lends itself well to recursion, as each level of coarseness does not need to know about any previous or next level.

The second step is initial clustering. Since any clustering method can be done here, k-means was selected for this project. While k-means is reliable, it does potentially change the data quality. It would have been more complete to run the Kernel Clustering algorithm twice, once with k-means and once with distance-k, or to have selected a third clustering algorithm entirely for kernel clustering.

The third step of the algorithm is refinement. This requires a memory of the different levels of coarseness, in order to step back through them. Unfortunately, we did not think of a better way to solve this issue other than to hold in memory the different graphs created and a record of how the nodes were merged to reach that step. For large graphs, this could be a prohibitive amount of memory, especially with many iterations of coarsening.

At each level of refinement of nodes, the clusters are also refined. The algorithm presented for this was a modified k-means algorithm [3], which is expected to converge quickly at every iteration because the centroids received a good initialization due to the coarsening steps. The pseudo-code for the refinement of the clusters can be found in Appendix D.

5 Experimental Results

Appendices

A UML Diagram for Custom Classes

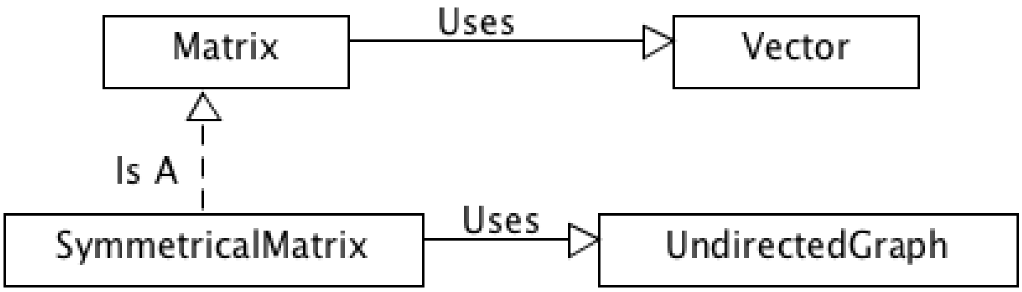

Figure 1: UML Diagram for Relationships between Class Types

B Pseudo-Code for the K-Means Algorithm

Input:
D = {d1, d2,......,dn} // set of *n* data items

k // Number of desired clusters

Output:

A set of *k* clusters.

Steps:

Phase 1: Determine the initial centroids of the clusters by using Algorithm 3.

Phase 2: Assign each data point to the appropriate clusters by using Algorithm 4.

Figure 2: Pseudo-Code for the K-Means Algorithm

C Pseudo-Code for the K-Means Cluster Initialization Algorithm

Input:

D = {d1, d2,......,dn} // set of *n* data items

k // Number of desired clusters

Output: A set of *k* initial centroids .

Steps:

1. Set m = 1;
2. Compute the distance between each data point and all other data- points in the set D;
3. Find the closest pair of data points from the set D and form a data-point set Am (1<= m <= k) which contains these two data- points, Delete these two data points from the set D;
4. Find the data point in D that is closest to the datapoint set Am, Add it to Am and delete it from D;
5. Repeat step 4 until the number of data points in Am reaches 0.75*(n/k);
6. If m<k, then m = m+1, find another pair of datapoints from D between which the distance is the shortest, form another data-point set Am and delete them from D, Go to step 4;
7. For each data-point set Am (1<=m<=k) find the arithmetic mean of the vectors of data points in Am, these means will be the initial centroids.

Figure 3: Pseudo-Code for the K-Means Cluster Initialization Algorithm

D Pseudo-Code for the Kernel K-Means Refinement Algorithm

Weighted_Kernel_kmeans(K, k, w, t_{max}, $\{\pi_c^{(0)}\}_{c=1}^k$, $\{\pi_c\}_{c=1}^k$)
Input: K: kernel matrix, k: number of clusters, w: weights for each point, t_{max}: optional maximum number of iterations, $\{\pi_c^{(0)}\}_{c=1}^k$: optional initial clustering
Output: $\{\pi_c\}_{c=1}^k$: final clustering of the points
1. If no initial clustering is given, initialize the k clusters $\pi_1^{(0)}, ..., \pi_k^{(0)}$ randomly. Set $t = 0$.
2. For each row i of K and every cluster c, compute

$$d(i, \mathbf{m}_c) = K_{ii} - \frac{2\sum_{j\in\pi_c^{(t)}} w_j K_{ij}}{\sum_{j\in\pi_c^{(t)}} w_j} + \frac{\sum_{j,l\in\pi_c^{(t)}} w_j w_l K_{jl}}{(\sum_{j\in\pi_c^{(t)}} w_j)^2}.$$

3. Find $c^*(i) = \text{argmin}_c d(i, \mathbf{m}_c)$, resolving ties arbitrarily. Compute the updated clusters as

$$\pi_c^{(t+1)} = \{i : c^*(i) = c\}.$$

4. If not converged or $t_{max} > t$, set $t = t + 1$ and go to Step 3; Otherwise, stop and output final clusters $\{\pi_c^{(t+1)}\}_{c=1}^k$.

Figure 4: Pseudo-Code for the Kernel K-Means Refinement Algorithm

E Code

The Git repository for this project can be found at:

https://github.com/AriellaRomanov/Clustering

References

[1] I. Dhillon, Y. Guan, and B. Kulis, "A Fast Kernel-based Multilevel Algorithm for Graph Clustering," *KDD*. Chicago, Ill.: 2005.

[2] J. Edachery, A. Sen, and F. Brandenburg, "Graph Clustering Using Distance-k Cliques," Arizona State University, 1999.

[3] K. Nazeer and M. Sebastian, "Improving the Accuracy and Efficiency of the k-means Clustering Algorithm," *Proceedings of the World Congress on Engineering.* London, U.K.: 2009.