Group 19 Final Project Report
Spectral Clustering of Wikipedia Articles Using the Edit History

Konstantin Lopyrev
klopyrev@gmail.com

Chip Mandal
chitradip.mandal@gmail.com

Saila Talagadadeevi
saila@stanford.edu

December 10, 2013

1 Introduction

The spectral clustering algorithm is a powerful clustering algorithm that is known to give better clusterings than other algorithms such as k-means. Most spectral clustering scenarios require the computation of the distance between every pair of points in order to construct the similarity matrix that is the input to the spectral clustering algorithm. In fact, this similarity matrix construction step is one of the most expensive steps of the algorithm, sometimes taking significantly longer than the actual clustering.

We study an optimization to this similarity matrix construction step. We apply the locality sensitive hashing algorithm which we use to approximately compute all pairs of near neighbors. We test our algorithm on the Wikipedia edit history dataset on which we define article similarity using the list of editors of each article. We find that using locality sensitive hashing results in a 10- to 20- fold decrease in the running time of the similarity matrix construction step without a significant decrease in the quality of the final clustering.

2 Details of Spectral Clustering Algorithm

2.1 Key Definitions

1. Jaccard Similarity: The Jaccard similarity measures the similarity between 2 sets. It is defined as the size of the intersection divided by the size of the union of the sets: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. In the case of Wikipedia articles each article is associated with the set of editors who have edited the article. Then the Jaccard similarity is defined as the number of common editors between the 2 articles divided by the number of editors who have edited either article.

2. Similarity Matrix: The similarity matrix W gives for each pair of Wikipedia articles the Jaccard similarity between those articles. This matrix is the primary input to the spectral clustering algorithm. One problem with using the full similarity matrix is that it can contain very large numbers of non-zero entries, taking a long time to compute and slowing down all the later steps of the spectral clustering algorithm. Thus, an important step in applying the spectral clustering algorithm is efficiently computing a similarity matrix of reduced size. This step is the primary focus of this project.

3. Normalized Laplacian: The Laplacian matrix is defined as $L = I - W$. This matrix has a number of useful spectral properties and is often used in spectal graph analysis. The normalized Laplacian is defined as $L = I - D^{-1/2} W D^{-1/2}$ where D is the degree matrix defined as $D_{ii} = \sum_{j=1}^{n} W_{ij}$. For practical purposes, however, most implementations actually use $L = D^{-1/2} W D^{-1/2}$. Doing so simply changes the eigenvalues of the matrix to be $1 - \lambda$, instead of λ, without changing the corresponding eigenvectors.

4. Spectral Graph Theory: Spectral graph theory examines the eigenvalues and eigenvectors of a graph's Laplacian matrix and uses them to determine structural properties of the graph

2.2 Steps of the Algorithm

For this project we use the variant of the spectral clustering algorithm proposed in [5]. This variant is described below, with details about our usage of it.

Step 1: Process raw data and build symmetric similarity matrix W: Given a set of points build similarity matrix containing similarities between points in the graph. We build the similarity matrix for Wikipedia articles using the Jaccard similarirty as defined in section 2.1. We also make the matrix sparse by making entries in the matrix that are $> \epsilon$ into 1s and the rest into 0s.

Step 2: Build diagonal degree matrix D: The ith entry on the diagonal of this matrix is defined as the sum of the entries for row i or column i of the similarity matrix. Since the matrix is symmetric these are the same.

Step 3: Build normalized Laplacian matrix L: From W and D, build $L = D^{-1/2}WD^{-1/2}$

Step 4: Find the top k eigenvectors of L: Find the k eigenvectors of the Laplacian matrix L that correspond to the largest eigenvalues. Form matrix $V \in \mathbb{R}^{n \times k}$ containing these eigenvectors.

Step 5: Normalize the rows of V: Normalize each row so that its norm is equal to 1. Form matrix Y such that $Y_{ij} = \frac{V_{ij}}{\sqrt{\sum_l V_{il}^2}}$

Step 6: Cluster Y by treating each row in Y as a point in $\mathbb{R}^k$: Commonly this clustering is done using the k-means algorithm. This is the algorithm that we use.

Step 7: Assign points in the graph to clusters: Each point i of the original graph is assigned to the cluster that the ith row of Y belongs to.

We implemented steps 2 - 7 of the algorithm using the implementation described in [1]. Their code is published online and is also available internally at Google. We ran steps 2 - 7 using roughly 50 machines.

2.3 Algorithm Explanation

There are multiple ways of measuring the quality of a graph clustering. Generally clustering has 2 objectives:

- Minimizing between-cluster similarity: Points in different clusters are dissimilar to each other.
- Maximizing within-cluster similarity: Points in the same cluster are similar to each other.

There are multiple different metrics that capture these objectives. These metrics are generally referred to as minimum cut metrics. Unfortunately, most useful minimum cut problems are NP-hard. However, there are ways of relaxing the minimum cut problem that result in efficient algorithms. [10] shows that the spectral clustering algorithm applied to the normalized Laplacian in fact solves a relaxed version of the minimum cut problem where the 2 objectives defined above are optimized.

You can also look at spectral clustering from a different angle. Computing the top eigenvectors of the Laplacian is similar to computing the SVD of the matrix and choosing the left-singular eigenvectors corresponding to the largest singular values. The result is a low-rank approximation where each point is represented by a low-dimensional embedding. This process can be said to remove noise from the data [7].

2.4 Similarity Matrix Computation

The input to the spectral clustering algoritm is a similarity matrix W. To compute this matrix, we need to calculate the similarity between every pair of articles and then choose only those with similarity $> \epsilon$. There are $\binom{n}{2}$ such pairs, which given the size of Wikipedia is computationally expensive.

One optimization that we make is to filter out all editors who have edited more than 1000 articles. Doing so results in significant decreases in the runtimes of all the steps and offers some quality improvements. This filtering removes a number of editors who have bot-like behavior, editing 1000s of articles and adding noise to the data.

2.4.1 Exhaustive Computation

The standard way of computing the similarity matrix is to exhaustively compute the similarity between every pair of articles, yielding the exact similarity matrix. Using this dataset we ran a full $O(n^2)$ computation of Jaccard similarities between every pair of articles. This computation took 200 machines roughly 4 hours.

2.4.2 Locality Sensitive Hashing (LSH)

To optimize the computation of the similarity matrix, we use locality sensitive hashing to find subsets of articles that are all approximately similar to each other. The locality sensitive hashing algorithm efficiently puts similar items in the same buckets.

The steps to perform LSH are as follows:

Step 1: Each article in represented by the editors that edited the article, with the editors who have edited more than 1000 articles excluded.

Step 2: Each of the editors of a particular article is hashed using $b \cdot r$ different hash functions. Then, for each hash function the minimum value over all the editors is taken as the minhash. It can be shown that the probability that any one minhash of 2 articles is equal is the same as their Jaccard similarity.

Step 3: The minhash vector of each article is divided into b bands of r rows each. A hash is computed from the minhashes in each band. All articles which have the same hash for the same band are placed in the same bucket. Note that each article gets placed in b different buckets.

Step 4: All articles which fall into the same bucket are similar to each other with high probability. A 1 is added to the output similarity matrix for every pair of articles in some bucket.

The probability that 2 articles with similarity s will be hashed to the same bucket is $1-(1-s^r)^b$. Choosing $\left(\frac{1}{b}\right)^{\frac{1}{r}} \approx \epsilon$ ensures that with high probability only articles with similarity ϵ will be hashed to the same bucket in at least one of the bands. Low values of b and r result in less accuracy of the output similarity matrix, thus giving a lower quality of clustering. The choice of b and r represents a tradeoff between computation time and cluster quality. Figure 1 shows the tradeoff between values of b and r, and the accuracy of the resulting output.

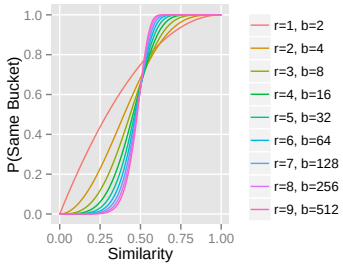

Figure 1: Probability of hashing to the same bucket

Note that a big benefit of using locality sensitive hashing is that the similarity matrix computation is trivially parallelizable. For example, using MapReduce [2] the Mappers compute the buckets for each of the articles in parallel. Then, each bucket becomes the key in the Reduce phase. All the articles that are in the same bucket are joined in the Reducers.

3 Prior Work

Spectral clustering is an algorithm that is known to work very well in practice [5]. However, it is hard to explain exactly why it works and what it does. Several papers provide different explanations. Notable work by Ng et al. [5] uses matrix perturbation theory. Other works use random walks [3] or relaxation of the graph normalized cut problem [8].

A good survey on the current state of spectral clustering is given by Luxburg [10]. It discusses various aspects of spectral clustering including the mathematical concepts and theories. It presents several different ways of doing the spectral clustering. Finally, it gives practical advice, based on a geometric interpretation of the data, on which type of Laplacian to use and how to compute the similarity matrix.

Most of the algorithms and advice from the previous papers are based on small datasets. Chen et al [1] describe attempts to handle large scale data using a distributed system. Their paper provides a thorough analysis of the parallelizability of different steps of the algorithm, showing how the processing time changes as the number of computing machines is increased. However, the computation of the similarity matrix takes a very long time, even with large numbers of machines. The Mahout package includes a spectral clustering implementation that is similar to the one discussed in [1], except for the fact that it uses Hadoop, instead of message passing. However, we find that the implementation does not seem to work in a distributed environment. Even then, the implementation still needs the similarity matrix as input.

More recently, the Spectral Neighborhood (SPAN) algorithm has been described in [9]. This algorithm performs spectral clustering without computing pairwise similarities. It is similar to our work since it uses a technique called *blocking* where points are divided into blocks and the similarities are computed only for points in the same block. Although, an important difference in their application of spectral clustering is that they do spectral biclustering where they repeatedly divide the points into 2 clusters using spectral graph methods.

For this project we use the variant of spectral clustering and its distributed implementation as presented in [1]. We apply the algorithm to the large Wikipedia edit history dataset and attempt to improve the performance of the similarity matrix computation step.

4 Data Collection

We downloaded the Wikipedia edit history dataset containing all edits to the main namespace until some point in 2008 from a link given on the SNAP website. We extracted the compressed 8.1G input file and converted the data into 2202 sharded files totaling 205G. We then used MapReduce [2] to parse the data, drop unnessary fields and put the data into a more structured format, yielding 100 files totaling 5.6G. We also converted the data into a columnar data format for quick adhoc analysis using the Dremel tool [4].

Finally, using the structured data we built the article signature dataset containing for each article the list of editors in sorted order. The resulting file is 99M in size, and is the input to the similarity matrix computation.

5 Dataset Description

Our dataset includes information on 78,437,759 edits of 2,920,796 Wikipedia articles by 1,525,755 users. 2,144,290 of the articles have one or more of 543,502 categories associated with them. On average, each article has 12.5 editors, with the maximum number of editors for some article being 5,913. The degree distribution is shown in figure 2 and seems to follow a powerlaw for the most part.

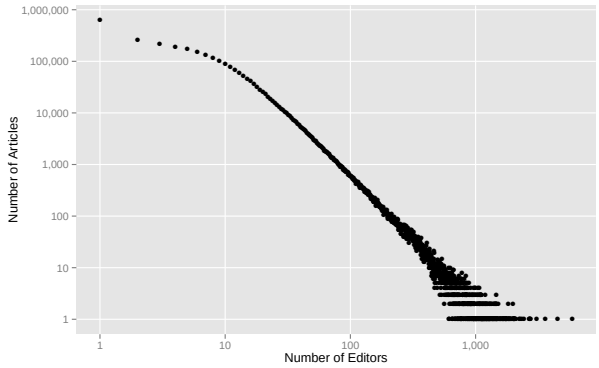

Figure 2: Article degree distribution

A total of 929,752,989,192 pairs of articles has non-zero similarity, with the similarities distributed as shown in figure 3.

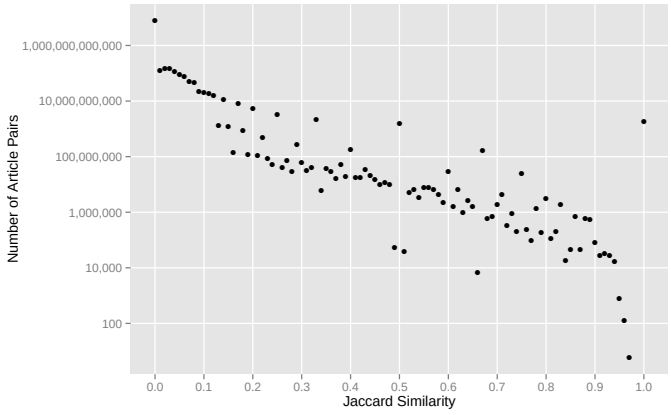

Figure 3: Article similarity distribution

6 Clustering Quality Evaluation

We used the category information attached to each article in order to evaluate the different clusterings. We summarized the category information of each article, giving us a single category for that article. To do so we found the following quantities: number of edits, e, number of edits of article a, e_a, number of edits containing category c, e_c, and number of edits of article a that contained the category c, $e_{a,c}$. Then we found the top category for each article by choosing the category with the highest $tf \cdot idf$ calculated using the formula

$$tf \cdot idf = \left(\frac{e_{a,c}}{e_a}\right) \cdot \left(\log \frac{e}{e_c}\right)$$

We chose a sample of all the articles when calculating the quality metrics. This sampling was done to ensure that the metrics for different clusterings were comparable to each other. The set of articles that were clustered varied with the parameter ϵ since some articles did not have any other articles with similarity $> \epsilon$. The sample we chose included all the articles that had more than 2 editors, that had a category and that had other articles with > 0.832 similarity to them. This sample consisted of a total of 18550 articles.

We implemented 2 metrics: Ncut and normalized mutual information (NMI), described below. We used 2 baselines for our metrics. For the first baseline we randomly partitioned all of the articles into some given number of clusters. For the second baseline we found the connected components of the similarity matrix calculated using $\epsilon = 0.5$. Each of these gave us a clustering that we then compared to the clusterings returned by the algorithm.

6.1 Ncut

The Ncut metric is a minimum cut metric that captures 2 important goals of clustering: minimizing between-cluster similarity and maximizing within-cluster similarity. For a graph $G = (V, E)$ with each edge $(u, v) \in E$ having a weight w_{uv} the between-cluster similarity is measured using the cut which is defined for a cluster A as

$$cut(A, \bar{A}) = \sum_{(u,v) \in E | u \in A, v \in \bar{A}} w_{uv}$$

Between-cluster similarity is minimized when the cut is minimized.

The within-cluster similarity is measured using the difference between the volume and the cut, $vol(A) - cut(A, \bar{A})$, where the volume is defined as

$$vol(A) = \sum_{(u,v) \in E | u \in A, v \in V} w_{uv}$$

Within-cluster similarity is maximized when volume is maximized and cut is minimized.

Ncut captures both of the goals and is defined using the formula

$$Ncut(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{cut(A_i, \bar{A}_i)}{vol(A_i)}$$

where A_i is the set of articles in cluster i. Smaller values of Ncut are better.

We define a graph $G = (V, E)$ on the articles using the top category information. We connect all the articles in a given category to each other. Each edge of an article is given a uniform weight in such a way that the total degree of each article is equal to 1.

6.2 Normalized Mutual Information (NMI)

The normalized mutual information metric is a measure of dependence of 2 random variables, where in our case the random variables are the clusters, CLS and the top categories, CAT. The normalized mutual information achieves a maximum value of 1 when the clusters exactly match the categories. The normalized mutual information is defined as

$$NMI(CAT; CLS) = \frac{I(CAT; CLS)}{\sqrt{H(CAT) H(CLS)}}$$

$I(CAT; CLS)$ captures the mutual information between the clusters and the categories. The category and cluster entropies $H(CAT)$ and $H(CLS)$, respectively, are used to normalize the mutual information to be in the range $[0, 1]$. The formula is implemented by calculating 4 counts: the total number of articles, n, the number of articles in a given category, n_i, the number of articles in a given cluster, n_j, and the number of articles in a given category i and cluster j, n_{ij}. Given these counts, the normalized mutual information is calculated as

$$NMI = \frac{\sum_{i,j} \frac{n_{i,j}}{n} \log \frac{n \cdot n_{ij}}{n_i \cdot n_j}}{\sqrt{\left(\sum_i -\frac{n_i}{n} \log \frac{n_i}{n}\right) \left(\sum_j -\frac{n_j}{n} \log \frac{n_j}{n}\right)}}$$

Larger values of NMI are better.

7 Findings

We conducated 3 experiments in order to evaluate the performance of each of the 2 methods of constructing the similarity matrix. In the first 2 experiments we determined how the quality of the clustering changes as we vary the threshold, ϵ, and the number of clusters, k. We ran these experiments in order to understand how sensitive our metrics are to changes in these parameters. Doing so helped us make sure that we were not misinterpretting the final results. For our last experiment we fixed ϵ and k, and varied the values of the 2 parameters of the locality sensitive hashing algorithm: b and r. In the last experiment we measured the quality of the clustering as well as the runtime of the similarity matrix construction step using locality sensitive hashing. We compared the results to the quality and the runtime of the algorithm on the exact similarity matrix.

Additionally, we manually looked at the final clusterings and verified that they looked reasonable using the manually compiled lists of articles as available on Wikipedia.

7.1 Experiments

7.1.1 Varying ϵ

In our first experiment we fixed the number of clusters, k, to 500. Then, we chose several levels of ϵ. We chose ϵ based on the article similarity distribution graph shown in figure 3. Each ϵ was chosen to capture a particularly frequently occuring value of the Jaccard similarity. The levels we settled on are 0.499, 0.57, 0.599, 0.624, 0.665, 0.713, 0.749, 0.799 and 0.823. The metrics for these values of ϵ are shown in figures 4 and 5.

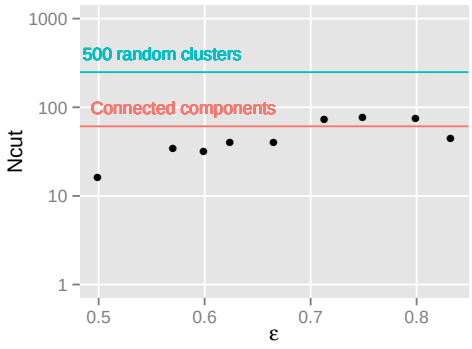

Figure 4: Ncut

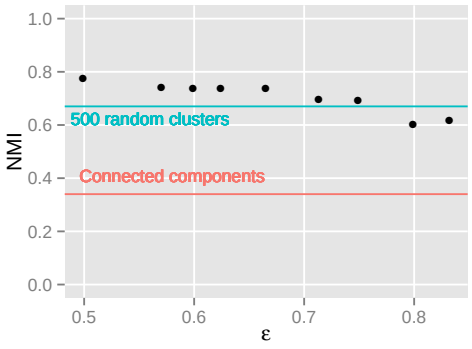

Figure 5: NMI

We find that the lower we choose ϵ the better the final clustering. However, computing the clustering with lower values of ϵ takes more time. For the rest of our experiments we decided to use $\epsilon = 0.499$.

7.1.2 Varying number of clusters k

In the second experiment we wanted to understand how the quality changes as we change the number of clusters. We calculated each of the clusterings with $\epsilon = 0.499$ and varying k. The results are shown in figures 6 and 7.

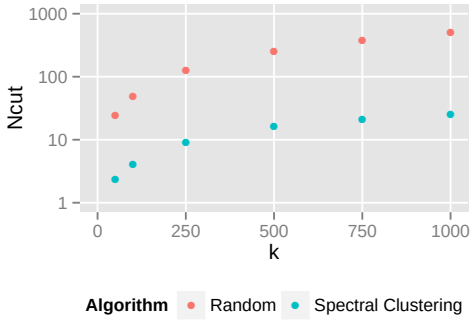

Figure 6: Ncut

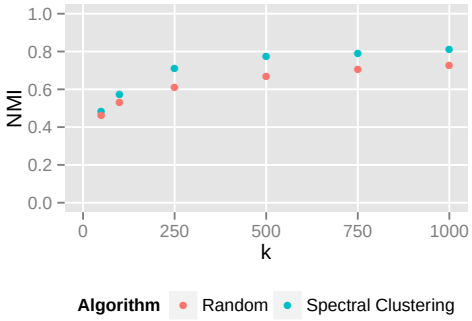

Figure 7: NMI

As expected as the number of clusters is increased the quality of the resulting clustering increases. This experiment proved that the clustering was working as expected and showed how sensitive the metrics are to the number of clusters. However, choosing higher values of k caused the runtime to increase significantly. The slowest step of the algoritm, aside from the similarity matrix computation, was the eigendecomposition step which we ran on 50 machines. For $k = 100$ this step took 20 minutes. For $k = 500$ this step took 3 hours. For $k = 1000$ this step took 10 hours. Since chosing higher values of k caused the runtime to increase significantly, for the last experiment we fixed k to 500.

7.1.3 Locality Sensitive Hashing Evaluation

For our last experiment we compared locality sensitive hashing to the exhaustive matrix computation. We chose 9 different values of the locality sensitive hashing parameters, b and r, all chosen in such a way that $\left(\frac{1}{b}\right)^{\frac{1}{r}} = 0.5$. We looked at the quality of the clusterings resulting from each choice. This data is presented in figures 8 and 9.

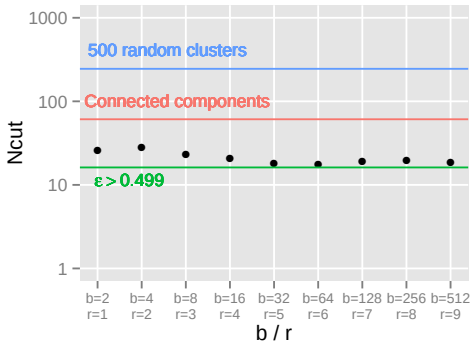

Figure 8: Ncut

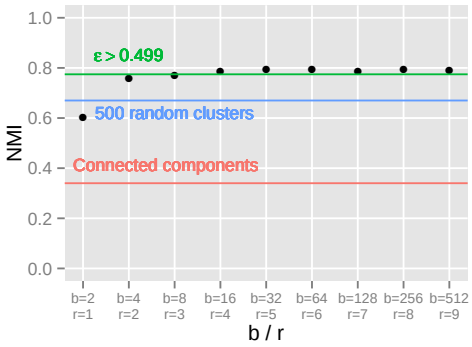

Figure 9: NMI

We can see that the quality of the LSH clusterings is comparable to the quality of the exact matrix clusterings. In fact, for $b \cdot r = 32 \cdot 5 = 160$ or more the Ncut of the LSH clusterings is within 13% of the Ncut of the exact matrix clusterings. What's surprising is that for $b \cdot r = 8 \cdot 3 = 24$ or more the NMI is actually better for the LSH clusterings.

The runtime of the LSH algorithm, however, is significantly lower than the runtime of the exhaustive matrix computation, as shown in figure 10. For both the exact computation and for the locality sensitive hashing computations 200 machines were used.

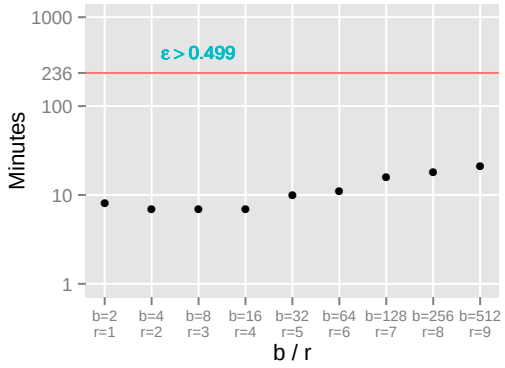

Figure 10: LSH similarity matrix computation timings

This experiment shows that using the approximation of the similarity matrix as calculated using locality sensitive hashing results in a final clustering the quality of which is comparable to the quality of the clustering done on the exact similarity matrix. However, using locality sensitive hashing reduces the runtime of the similarity matrix computation step by 10- to 20 fold.

7.2 Qualitative Evaluation

We handpicked a few lists of articles and checked to see if the articles in those lists were clustered together. The examples that we found interesting were:

- Legislative districts of the Philippines
- Seasons of St. Louis Baseball team
- Seasons of CSI Miami (TV Show)

All the 82 articles about "Legislative districts of Phillipines" were clustered together, as were articles about all the seasons of "CSI Miami". 123 of the 124 of the articles for "Seasons of St. Louis Baseball team" were in the same cluster.

Although most of the elements in the lists were in the same cluster, the clusters had lot of other articles as well. This is because our k (500) was too small for a dataset this size. Choosing higher values of k was prohibitive based on runtime.

We also filtered out articles which were not similar to any other article based on editors. This reduced the number of articles by approximately 50%. We could not validate some lists like "American Presidents".

8 Discussion

We show that using locality sensitive hashing during the computation of the similarity matrix computation step significantly decreases the runtime of the overall algorithm. A major limitation of our work, though, is the fact that not all distance functions have a nice locality sensitive hashing scheme. There exist schemes for a number of other distance functions, such as cosine similarity [6]. However, such schemes are not always easy to work with.

Another limitation of our work is the runtime of the eigendecomposition step. We show that as the number of clusters increases the quality of the clustering increases significantly. However, the runtime quickly becomes unmanageable. One way to address this issue is to do repeated spectral biclustering. This divide and conquer algorithm is much easier to parallelize and may be more numerically stable. A variant of this algorithm is presented in [9].

References

[1] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(3):568–586, March 2011.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[3] Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. 2001.

[4] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pages 330–339, 2010.

[5] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[6] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[7] Delip Rao, David Yarowsky, and Chris Callison-Burch. Affinity measures based on the graph laplacian. In *Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing*, TextGraphs-3, pages 41–48, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[8] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[9] Liangcai Shu, Aiyou Chen, Ming Xiong, and Weiyi Meng. Efficient spectral neighborhood blocking for entity resolution. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1067–1078, 2011.

[10] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.