A Divide-and-Merge Methodology for Clustering

David Cheng
MIT
drcheng@mit.edu

Ravi Kannan
Yale University
kannan@cs.yale.edu

Santosh Vempala
MIT
vempala@mit.edu

Grant Wang
MIT
gjw@mit.edu

Abstract

We present a divide-and-merge methodology for clustering a set of objects that combines a top-down "divide" phase with a bottom-up "merge" phase. In contrast, previous algorithms use either top-down or bottom-up methods for constructing a hierarchical clustering or produce a flat clustering using local search (e.g. k-means). Our divide phase produces a tree whose leaves are the elements of the set. For this phase, we suggest an efficient spectral algorithm. The merge phase quickly finds the optimal partition that respects the tree for many natural objective functions, e.g., k-means, min-diameter, min-sum, correlation clustering, etc. We present a meta-search engine that clusters results from web searches. We also give empirical results on text-based data where the algorithm performs better than or competitively with existing clustering algorithms.

1 Introduction

The rapidly increasing volume of readily accessible data presents a challenge for computer scientists: find methods that can locate relevant information and organize it in an intelligible way. This is different from the classical database problem in at least two ways: first, there may neither be the time nor (in the long term) the computer memory to store and structure all the data (e.g. the world-wide web or a portion of it) in a central location. Second, one would like to find interesting patterns in the data without knowing what to look for in advance.

Clustering refers to the process of classifying a set of data objects into groups so that each group consists of similar objects. The classification could either be flat (a partition of the data set usually found by a local search algorithm such as k-means [17]) or hierarchical [19]. Clustering has been proposed as a method to aid information retrieval in many contexts (e.g. [12, 31, 28, 23, 15]). Document clustering can help generate a hierarchical taxonomy efficiently (e.g. [9, 35]) as well as organize the results of a web search (e.g. [33, 32]). It has also been used to learn (or fit) mixture models to data sets [18] and for image segmentation [30].

Most hierarchical clustering algorithms can be described as either divisive methods (i.e. top-down) or agglomerative methods (i.e. bottom-up) [5, 19, 20]. Both methods create trees, but do not provide a flat clustering. A divisive algorithm begins with the entire set and recursively partitions it into two pieces, forming a tree. An agglomerative algorithm starts with each object in its own cluster and iteratively merges clusters. We combine top-down and bottom-up techniques to create both a hierarchy and a flat clustering. In the divide phase, we can apply any divisive algorithm to form a tree T whose leaves are the objects. This is followed by the merge phase in which we start with each leaf of T in its own cluster and merge clusters going up the tree. The final clusters form a partition and are tree-respecting clusters, i.e., subtrees rooted at some node of T. For a large class of natural objective functions, the merge phase can be executed optimally, producing the best tree-respecting clustering.

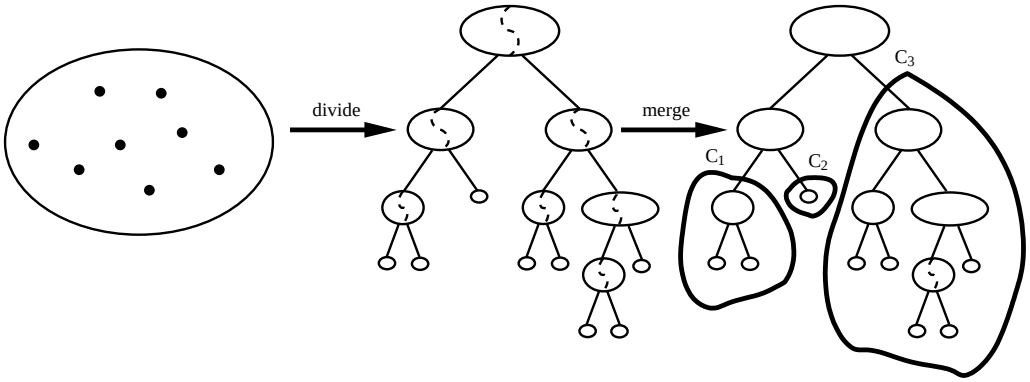


Figure 1: The Divide-and-Merge methodology

For the divide phase we suggest using the theoretical spectral algorithm studied in [21]. There, the authors use a quantity called *conductance* to define a measure of a good clustering based on the graph of pairwise similarities. They prove that the tree constructed by recursive spectral partitioning contains a partition that has reasonable worst-case guarantees with respect to conductance. However the running time for a data set with n objects could be $O(n^4)$. We describe an efficient implementation of this algorithm when the data is presented in a document-term matrix and the similarity function is the inner product. For a document-term matrix with M nonzeros, our implementation runs in $O(Mn \log n)$ in the worst case and seems to perform much better in practice (see Figure 2(a)). The data need not be text; all that is needed is for the similarity of two objects to be the inner product between the two vectors representing the objects.

The class of functions for which the merge phase can find an optimal tree-respecting clustering include standard objectives such as k-means [17], min-diameter [11], and min-sum [25]. It also includes correlation clustering, a formulation of clustering that has seen recent interest [6, 10, 14, 16, 29]. Each of the corresponding optimization problems is NP-hard to solve for general graphs. Although approximation algorithms exist, many of them have impractical running times. Our methodology can be seen as an efficient alternative.

We show promising empirical results for the methodology. The first application is a meta-search engine (called EigenCluster [2]) that clusters the results of a query to a standard web search engine. EigenCluster consistently finds the natural clustering for queries that exhibit polysemy, e.g. a query of `monte carlo` to EigenCluster results in the identification of clusters pertaining to the car model, the city in Monaco, and the simulation technique. We describe EigenCluster and show results of example queries in Section 3. We also apply the methodology to clustering text-based data whose correct classification is already known. In Section 4, we describe the results of a suite of experiments that show that a good clustering exists in the tree built by the spectral algorithm.

2 The Divide-and-Merge methodology

As mentioned in the introduction, there are two phases in our approach. The divide phase produces a hierarchy and can be implemented using any algorithm that partitions a set into two disjoint subsets. The input to this phase is a set of objects whose pairwise similarities or distances are given (or can be easily computed from the objects themselves). The algorithm recursively partitions a cluster into two smaller sets until it arrives at singletons. The output of this phase is a tree whose

leaves are the objects themselves; each internal node represents a subset, namely the leaves in the subtree below it. We can apply graph partitioning algorithms when the objects are represented as vertices in a graph (and their pairwise similarities/distances form the edges) [15]. There are also divisive algorithms known when the objects are represented as vectors in high dimensional space [9]. In Section 2.1, we use a spectral algorithm for the divide phase when the objects are represented as a document-term matrix and the similarity between the objects is the inner product between the corresponding vectors.

The merge phase is applied to the tree T produced by the divide phase. The output of the merge phase is a partition $C_1, \ldots, C_k$ where each C_i is a node of T. The merge phase uses a dynamic program to find the optimal tree-respecting clustering for a given objective function g. The optimal solutions are computed bottom-up on T; to compute the optimal solution for any interior node C, we *merge* the optimal solutions for C_l and C_r, the children of C. The optimal solution for any node need not be just a clustering; an optimal solution can be parameterized in a number of ways. Indeed, we can view computing the optimal solution for an interior node as computing a Pareto curve; a value on the curve at a particular point is the optimal solution with the parameters described by the point. A specific objective function g can be efficiently optimized on T if the Pareto curve for a cluster can be efficiently computed from the Pareto curves of its children. In Section 2.2, we describe dynamic programs to compute optimal tree-respecting clusterings for several well-known objective functions: k-means, min-diameter, min-sum, and correlation clustering.

2.1 Divide phase

The spectral algorithm given here deals with the common case in which the objects are given as a sparse document-term matrix A. The rows are the objects and the columns are the features. We denote the ith object, a row vector in A, by $A_{(i)}$. The similarity of two objects is defined as the inner product of their term vectors: $A_{(i)} \cdot A_{(j)}$. The algorithm can easily be applied to the case when the pairwise similarities are given explicitly in the form of a similarity matrix. However, when the similarity function is the inner product, computation of the similarity matrix can be avoided and the sparsity of A can be exploited.

The algorithm constructs a hierarchical clustering of the objects by recursively dividing a cluster C into two pieces through a cut $(S, C \setminus S)$. To find the cut, we compute v, an approximation of the second eigenvector of the similarity matrix AA^T normalized so that all row sums are 1. The ordering of the objects in v gives a set of cuts, and we take the "best" one. The algorithm then recurses on the subparts. To compute the approximation of the second eigenvector, we use the power method, a technique for which it is not necessary to explicitly compute the normalized similarity matrix AA^T. We discuss this in Section 2.1.1. The algorithm is given below.

Algorithm Divide
Input: An $n \times m$ matrix A.
Output: A tree with the rows of A as leaves.

1. Let $\rho \in \mathbb{R}^n$ be a vector of the row sums of AA^T, and $\pi = \frac{1}{(\sum_i \rho_i)}\rho$.
2. Let $R = \text{diag}(\rho)$, and $D = \text{diag}(\sqrt{\pi})$ be diagonal matrices.
3. Compute the second largest eigenvector v' of $Q = DR^{-1}AA^TD^{-1}$.
4. Let $v = D^{-1}v'$, and sort v so that $v_i \leq v_{i+1}$.
5. Find the value t such that the cut $(S,T) = (\{v_1, \ldots, v_t\}, \{v_{t+1}, \ldots, v_n\})$ minimizes the conductance:
$$\phi(S,T) = \frac{c(S,T)}{\min(c(S), c(T))}$$
where $c(S,T) = \sum_{i \in S, j \in T} A_{(i)} \cdot A_{(j)}$, and $c(S) = C(S, \{, 1 \ldots, n\})$.
6. Let $\hat{A}_S$, $\hat{A}_T$ be the submatrices of A whose rows are those in S, T. Recurse (Steps 1-5) on the submatrices $\hat{A}_S$ and $\hat{A}_T$.

In Step 5, we use the cut that minimizes the conductance. Informally, the less weight crossing the cut and the more even the size of S and T are, the lower the conductance and the "better" the cut is. The conductance of a cluster is the minimum conductance achieved by any cut, which we denote (S^*, T^*). The cut the algorithm finds using the second largest eigenvector is not much worse than the cut (S^*, T^*) in terms of conductance. Theoretical guarantees on the quality of the clustering produced can be found in [21].

For a document-term matrix with n objects and M nonzeros, Steps 1-5 take $O(M \log n)$ time. Theoretically, the worst-case time to compute a complete hierarchical clustering of the rows of A is $O(Mn \log n)$. Empirical experiments, however, show that the algorithm usually performs much better (see Section 2.1.2).

2.1.1 Details

Any vector or matrix that the algorithm uses is stored using standard data structures for sparse representation. The main difficulty is to ensure that the similarity matrix AA^T is not explicitly computed; if it is, we lose sparsity and our running time could grow to m^2, where m is the number of terms. We briefly describe how to avoid this.

Step 1: Computing row sums Observe that

$$\rho_i = \sum_{j=1}^{n} A_{(i)} \cdot A_{(j)} = \sum_{j=1}^{n} \sum_{k=1}^{m} A_{ik} A_{jk} = \sum_{k=1}^{m} A_{ik} \left(\sum_{j=1}^{n} A_{jk} \right).$$

Because $\sum_{j=1}^{n} A_{jk}$ does not depend on i, we can compute $u = \sum_{i=1}^{n} A_{(i)}$ so we have that $\rho_i = A_{(i)} \cdot u$. The total running time is $\theta(M)$ and the space required is $\theta(n+m)$.

Step 3: Computing the eigenvector The algorithm described in [21] uses the second largest eigenvector of $B = R^{-1}AA^T$, the normalized similarity matrix, to compute a good cut. To compute this vector efficiently, we compute the second largest eigenvector v of the matrix $Q = DBD^{-1}$. The eigenvectors and eigenvalues of Q and B are related; if $Bv = \lambda v$, then $Q(Dv) = \lambda Dv$.

Q is a symmetric matrix; it is easy to see this from $D^2B = B^TD^2$. Therefore, we can compute the second largest eigenvector of Q using the power method, an iterative algorithm whose main computation is a matrix-vector multiplication.

Power Method

1. Construct a random vector $v \in \mathbb{R}^n$ orthogonal to $\pi^T D^{-1}$.
2. Repeat $k = \ln(n \ln(\frac{1}{\delta}))/2\epsilon$ times:
   - Normalize v, i.e. set $v = \frac{v}{|v|}$.
   - Set $v = Qv$.

Step 1 ensures that the vector we compute is the second largest eigenvector. Note that $\pi^T D^{-1}Q = \pi^T D^{-1}$ so πD^{-1} is a left eigenvector with eigenvalue 1. To evaluate $Qv = v$ in Step 3, we only need to do four sparse matrix-vector multiplications, since $Q = (DR^{-1}AA^TD^{-1})$, and each of these matrices is sparse. Note that we do not form Q explicitly. The following lemma shows that the power method takes $\Theta(\log n)$ iterations to convert to the top eigenvector. Although stated for the top eigenvector, the lemma and corollary still hold when the starting vector is chosen uniformly over vectors orthogonal to the top eigenvector $\pi^T D^{-1}$; in this case, the power method will converge to the second largest eigenvector. The proof appears in the Appendix.

Lemma 1. *Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix, and let $v \in \mathbb{R}^n$ be chosen uniformly at random from the unit n-dimensional sphere. Then for any positive integer k, the following holds with probability at least $1 - \delta$:*

$$\frac{||A^{k+1}v||}{||A^kv||} \geq \left(n \ln \frac{1}{\delta}\right)^{-\frac{1}{2k}} ||A||_2.$$

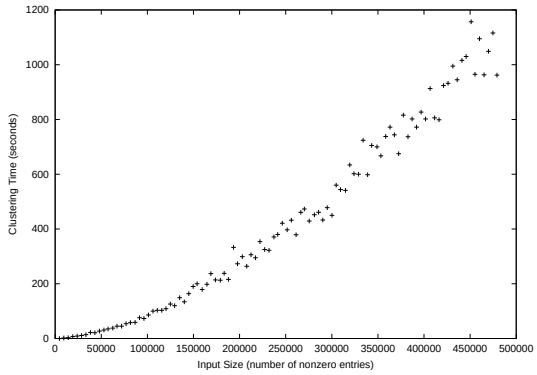
For the choice of k stated in the description of the power method, we have the following guarantee:

Corollary 1. *If $k \geq \frac{1}{2\epsilon} \ln(n \ln(\frac{1}{\delta}))$, then we have:*

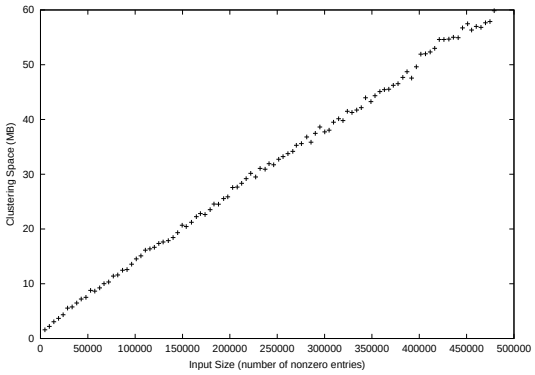
$$\frac{||A^{k+1}v||}{||A^kv||} \geq (1 - \epsilon)\lambda_1.$$

2.1.2 Time and space requirements

In practice, our algorithm seems to perform quite well. Figures 2(a) and 2(b) show the results of a performance experiment. In this experiment, we chose N random articles from each newsgroup in the 20 newsgroups data set [1] and computed a complete hierarchical clustering. Initially, N is 10, and was increased in increments of 10 until N is $1,000$. When we chose 1,000 documents from each of the newsgroups (for a total of 20,000 news articles and 500,000 nonzero entries in the document-term matrix), we were able to compute a complete hierarchical clustering in 20 minutes on commodity hardware.



(a) Time as a function of input size



(b) Space as a function of input size

Figure 2: Performance of spectral algorithm in experiments

2.2 Merge phase

The merge phase finds the optimal clustering in the tree T produced by the divide phase. In this section, we give dynamic programs to compute the optimal clustering in the tree T for many standard objective functions. The running time of the merge phase depends on both the number of times we must compute the objective function and the evaluation time of the objective function itself. Suppose at each interior node we compute a Pareto curve of k points from the Pareto curves of the node's children. Let c be the cost of evaluating the objective function. Then the total running time is $O(nk^2 + nkc)$: linear in n and c with a small polynomial dependence on k.

k-means: The k-means objective function seeks to find a k-clustering such that the sum of the squared distances of the points in each cluster to the centroid p_i of the cluster is minimized:

$$g(\{C_1, \ldots, C_k\}) = \sum_i \sum_{u \in C_i} d(u, p_i)^2.$$

The centroid of a cluster is just the average of the points in the cluster. This problem is NP-hard; several heuristics (such as the k-means *algorithm*) and approximation algorithms exist (e.g. [17, 22]). Let $\mathsf{OPT}(C, i)$ be the optimal clustering for C using i clusters. Let C_l and C_r be the left and right children of C in T. Then we have the following recurrence:

$$\mathsf{OPT}(C, i) = \begin{cases} C & \text{when } i = 1 \\ \text{argmin}_{1 \le j < i}\ g(\mathsf{OPT}(C_l, j) \cup \mathsf{OPT}(C_r, i-j)) & \text{otherwise} \end{cases}$$

By computing the optimal clustering for the leaf nodes first, we can determine the optimal clustering efficiently for any interior node. Then $\mathsf{OPT}(\mathsf{root}, k)$ gives the optimal clustering. Note that in the process of finding the optimal clustering the dynamic program finds the Pareto curve $\mathsf{OPT}(\mathsf{root}, \cdot)$; the curve describes the tradeoff between the number of clusters used and the "error" incurred.

Min-diameter: We wish to find a k-clustering for which the cluster with maximum diameter is minimized:

$$g(\{C_1, \ldots, C_k\}) = \max_i \mathsf{diam}(C_i).$$

The diameter of any cluster is the maximum distance between any pair of objects in the cluster. A similar dynamic program to that above can find the optimal tree-respecting clustering. This objective function has been investigated in [11].

Min-sum: Another objective considered in the literature is minimizing the sum of pairwise distances within each cluster:

$$g(\{C_1, \ldots, C_k\}) = \sum_{i=1}^{k} \sum_{u,v \in C_i} d(u, v).$$

We can compute an optimal answer in the tree T by a similar dynamic program to the one above. Although approximation algorithms are known for this problem (as well as the one above), their running times seem too large to be useful in practice [13].

Correlation clustering: Suppose we are given a graph where each pair of vertices is either deemed similar (red) or not (blue). Let R and B be the set of red and blue edges, respectively. Correlation clustering seeks to find a partition that minimizes the number of blue edges within clusters plus the number of red edges between clusters:

$$g(\{C_1 \ldots C_k\}) = \sum_i |\{(u, v) \in B \cap C_i\}| + \frac{1}{2}|\{(u, v) \in R : u \in C_i, v \in U \setminus C_i\}|.$$

Let C be a cluster in the tree T, and let C_l and C_r be its two children. The dynamic programming recurrence for $\mathsf{OPT}(C)$ is:

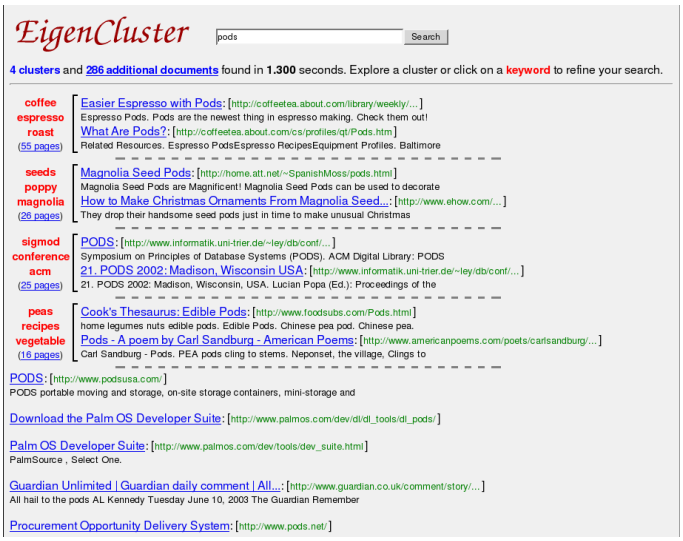
$$\mathsf{OPT}(C) = \text{argmin } \{g(C), g(\mathsf{OPT}(C_l) \cup \mathsf{OPT}(C_r)).$$

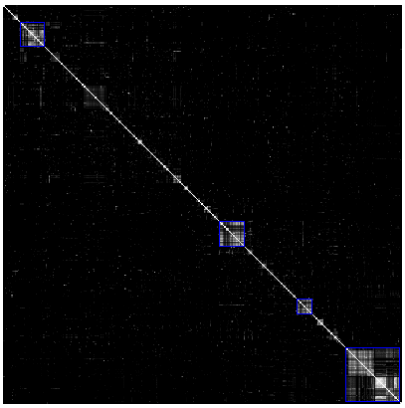
If, instead, we are given pairwise similarities in $[0, 1]$, where 0 means dissimilar and 1 means similar, we can define two thresholds t_1 and t_2. Edges with similarity greater than t_1 are colored red and edges with similarity less than t_2 are colored blue. The same objective function can be applied to these new sets of edges $R_{(t_1)}$ and $B_{(t_2)}$. Approximation algorithms have been given for this problem as well, although the techniques used (linear and semidefinite programming) incur large computational overhead [6, 10, 14, 16, 29].

3 Application to web searching: EigenCluster

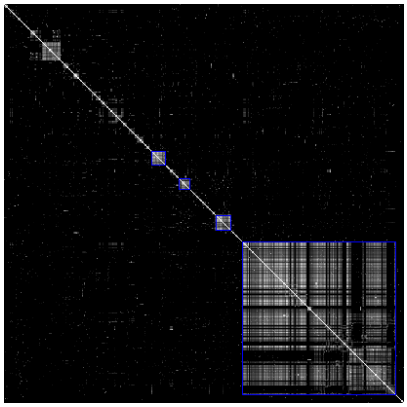
In a standard web search engine such as Google or Yahoo, the results for a given query are ranked in a linear order. Although suitable for some queries, the linear order fails to show the inherent clustered structure of the results for queries with multiple meanings. For instance, consider the query `mickey`. The query can refer to multiple people (e.g. Mickey Rooney and Mickey Mantle) or even a fictional character (e.g. Mickey Mouse).

We have implemented our methodology in a meta-search engine that discovers the clustered structure for queries and identifies each cluster by its three most significant terms. The website can be found at `http://eigencluster.csail.mit.edu`. The user inputs a query which is then used to find 400 results from Google, a standard search engine. Each result contains the title of the webpage, its location, and a small snippet. We construct a document-term matrix representation of the results; each result is a document and the words in its title and snippet make up its terms. Standard text pre-processing such as TF/IDF and removal of too frequent/infrequent terms is applied. The similarity between two results is the inner product between their two term vectors.





(a) Query: pods



(c) Query: mickey

The divide phase was implemented using our spectral algorithm. For the merge phase, we used the correlation clustering objective function with a threshold. A number of other natural objective functions seem to do comparably well. For instance, we have seen similar performance for minimizing the following objective function (for appropriate choice of α, β):

$$\sum_i \alpha \left(\sum_{u,v \in C_i} (1 - A_{(u)} \cdot A_{(v)}) \right) + \beta \left(\sum_{u \in C_i, v \notin C_i} A_{(u)} \cdot A_{(v)} \right).$$

The benefit of using these objective functions is that they do not depend on a predefined number of clusters k. This is appropriate for our application, since the number of meanings or contexts of a query is not known beforehand.

Sample queries can be seen in Figure 3; in each example, EigenCluster identifies the multiple meanings of the query as well as keywords corresponding to those meanings. Furthermore, many results are correctly labeled as singletons. In Figure 3, the pictures on the left are screenshots of EigenCluster. The pictures on the right are before and after depictions of the similarity matrix. In the before picture, the results are arranged in the order received from Google. In the after picture, the results are arranged according to the cuts made by the spectral algorithm. Here, the cluster structure is apparent. EigenCluster takes roughly .7 seconds to cluster results on a Pentium III 700 megahertz with 5 gigabytes of RAM. The total time to return clustered results from a query is 2 seconds (roughly 1.3 seconds are needed to fetch results from Google).

4 Experiments on text-based data

The appropriate objective function for an application will naturally depend on the specific application. To show the applicability of our methodology, we show experimental evidence that a *good* clustering exists in the hierarchical clustering constructed by the spectral algorithm. Finding the good clustering in the merge phase amounts to determining the right objective function to use. We used our spectral algorithm to create a hierarchical clustering for different data sets of text-based data. In each of the data sets, there was a pre-defined correct classification. We found the partition in the hierarchy that "agrees" the most with the correct classification. The amount of agreement was evaluated using three standard measures: F-measure, entropy, and accuracy. Descriptions of the measures can be found in the Appendix.

We performed experiments on the Reuters, SMART and 20 newsgroups data sets as well as data sets that were used in experiments for other clustering algorithms [9]. We compare the performance of the spectral algorithm in these experiments with known results of other algorithms on the data sets. In all of the experiments, we perform better or competitively with known results. The rest of this section describe the data sets and results.

4.0.1 20 newsgroups

The 20 newsgroups resource [1] is a corpus of roughly 20,000 articles that come from 20 specific Usenet newsgroups. We performed a subset of the experiments in [34]. Each experiment involved choosing 50 random newsgroup articles each from two newsgroups.[1] The results can be seen in Table 1. Note that we perform better than p-QR, the algorithm proposed in [34] on all but one of the experiments. We also outperform K-means and a variation of the K-means algorithm, p-Kmeans.

[1] We used the BOW toolkit for processing the newsgroup data. More information on the BOW toolkit can be found on `http://www-2.cs.cmu.edu/~mccallum/bow`.

In each of these experiments, the measure of performance was accuracy. Since the experiment involved choosing 50 random newsgroup articles, the experiment was run 100 times and the mean and standard deviation of the results were recorded.

data set	Spectral	p-QR	p-Kmeans	K-means
alt.atheism/comp.graphics	93.6 ± 2.6	89.3 ± 7.5	89.6 ± 6.9	76.3 ± 13.1
comp.graphics/comp.os.ms-windows.misc	81.9 ± 6.3	62.4 ± 8.4	63.8 ± 8.7	61.6 ± 8.0
rec.autos/rec.motorcycles	80.3 ± 8.4	75.9 ± 8.9	77.6 ± 9.0	65.7 ± 9.3
rec.sport.baseball/rec.sport.hockey	70.1 ± 8.9	73.3 ± 9.1	74.9 ± 8.9	62.0 ± 8.6
alt.atheism/sci.space	94.3 ± 4.6	73.7 ± 9.1	74.9 ± 8.9	62.0 ± 8.6
talk.politics.mideast/talk.politics.misc	69.3 ± 11.8	63.9 ± 6.1	64.0 ± 7.2	64.9 ± 8.5

Table 1: 20 newsgroups data set (Accuracy)

4.0.2 Reuters

The Reuters data set [3] is a corpus of $8,654$ news articles that have been classified into 135 distinct news topics. We performed same two experiments on this data set as were conducted in [8, 23, 24]. The first experiment, performed by [8, 23], constructed a complete hierarchical tree for a document-term matrix that includes all $8,654$ news articles. In the second experiment, a complete hierarchical tree was produced for a document-term matrix containing only $6,575$ news articles from 10 of the 135 largest news topics. This experiment was conducted by [24]. Our algorithm outperformed the results of prior experiments under the F-measure (see Table 2).

data set	Spectral	BEX02	LA99	NJM01
8,654 articles	.713	.57	.63	N/A
6,575 articles	.733	N/A	N/A	.665

Table 2: Reuters data set (F-measure)

4.0.3 Web pages

Boley [9] performs a series of experiments on clustering 185 webpages that fall into 10 distinct categories. In each of the 11 experiments (J1-J11), the term vector for each webpage was constructed in a slightly different way (the exact details can be found in [9]). A comparison of results under the entropy measure can be found in Table 3(b). In 7 of the 11 experiments, our algorithm performs better.

4.0.4 SMART data set

The SMART data set is a set of abstracts originating from Cornell University [4] that have been used extensively in information retrieval experiments. The makeup of the abstracts is as follows: 1,033 medical abstracts (Medline), 1,400 aeronautical systems abstracts (Cranfield), and 1,460 information retrieval abstracts (Cisi). We performed the same four experiments as those found in [15]. In the first three experiments, the data sets were the mixture of abstracts from two classes.

In the fourth experiment, the data set was the set of all abstracts. We perform competitively in the entropy measure (see Table 3(a)).

data set	Spectral	Dhillon 2001
MedCran	.032	.026
MedCisi	.092	.152
CisiCran	.045	.046
Classic3	.090	.089

(a) SMART data set (Entropy)

data set	Spectral	B97
J1	.77	.69
J2	.81	1.12
J3	.54	.85
J4	1.12	1.10
J5	.81	.74
J6	.81	.83
J7	.63	.90
J8	.84	.96
J9	.65	1.07
J10	1.77	1.17
J11	.90	1.05

(b) Webpage data set (Entropy)

Table 3: SMART and Webpage data sets

References

[1] 20 Newsgroups Data Set. `http://www.ai.mit.edu/people/jrennie/20Newsgroups/`.

[2] Eigencluster. `http://eigencluster.csail.mit.edu`.

[3] Reuters Data Set. `http://www.daviddlewis.com/resources/testcollections/reuters21578/`.

[4] SMART Data Set. `ftp://ftp.cs.cornell.edu/pub/smart`.

[5] M. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[6] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *Proceedings of IEEE Foundations of Computer Science*, 2002.

[7] Daniel Barbara, Yi Li, and Julia Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 582–589. ACM Press, 2002.

[8] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining*, 2002.

[9] Daniel Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[10] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. In *In Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.

[11] C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *In Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[12] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329. ACM Press, 1992.

[13] W.F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani. Approximation schemes for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, 2003.

[14] E.D. Demaine and N. Immorlica. Correlation clustering with partial information. In *In Proc. of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*.

[15] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Knowledge Discovery and Data Mining*, pages 269–274, 2001.

[16] D. Emanuel and A. Fiat. Correlation clustering–minimizing disagreements on arbitrary weighted graphs. In *In Proc. of the 11th European Symposium on Algorithms*, 2003.

[17] J.A. Hartigan and M.A. Wong. A k-means clustering algorithm. In *Applied Statistics*, pages 100–108, 1979.

[18] Thomas Hofmann. The cluster-abstraction model: Unsupervised learning of topic hierarchies from text data. In *IJCAI*, pages 682–687, 1999.

[19] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[20] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. In *ACM Computing Surveys*, volume 31, 1999.

[21] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad, and spectral. In *Journal of the ACM (JACM)*, volume 51, pages 497–515, 2004.

[22] A. Kumar, S. Sen, and Y. Sabharwal. A simple linear time $(1+\epsilon)$-approximation algorithm for k-means clustering in any dimensions. In *In Proceedings of the 45th Annual IEEE Foundations of Computer Science*, 2004.

[23] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–22. ACM Press, 1999.

[24] Adam Nickerson, Nathalie Japkowicz, and Evangelos Milios. Using unsupervised learning to guide re-sampling in imbalanced data sets. In *Proceedings of the Eighth International Workshop on AI and Statitsics*, pages 261–265, 2001.

[25] S. Sahni and T. Gonzalez. P-complete approximation problems. In *JACM*, volume 23, pages 555–566, 1976.

[26] C. E. Shannon. A mathematical theory of communication. In *Bell Systems Technical Journal*, volume 27, pages 379–423, 1948.

[27] Noam Slonim and Naftali Tishby. Document clustering using word clusters via the information bottleneck method. In *Research and Development in Information Retrieval*, pages 208–215, 2000.

[28] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *In KDD Workshop on Text Mining*, 2000.

[29] C. Swamy. Correlation clustering: Maximizing agreements via semidefinite programming. In *In Proc. of ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[30] J. Theiler and G. Gisler. A contiguity-enhanced k-means clustering algorithm for unsupervised multispectral image segmentation. In *Proceedings of the Society of Optical Engineering*, pages 108–111, 1997.

[31] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[32] W. Wong and A. Fu. Incremental document clustering for web page classification. In *IEEE 2000 Int. Conf. on Info. Society in the 21st century: emerging technologies and new challenges*, 2000.

[33] Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of web documents. In *Knowledge Discovery and Data Mining*, pages 287–290, 1997.

[34] H. Zha, C. Ding, M. Gu, X. He, and H. Simon. Spectral relaxation for k-means clustering. In *Neural Info. Processing Systems (NIPS 2001)*, 2001.

[35] Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 515–524. ACM Press, 2002.

5 Appendix

5.1 F-measure, Entropy, and Accuracy

For a data set let the correct classification be $C_1 \dots C_k$. We refer to each C_i as a *class*. Let the nodes of a hierarchical clustering be $\hat{C}_1 \dots \hat{C}_l$. We refer to each $\hat{C}_i$ as a *cluster* – the subset of nodes in the tree below it.

F-measure: For each class C_i, the F-measure of that class is:

$$F(i) = \max_{j=1}^{l} \frac{2P_j R_j}{P_j + R_j}$$

where:

$$P_j = \frac{|C_i \cap \hat{C}_j|}{|\hat{C}_i|}, R_j = \frac{|C_i \cap \hat{C}_j|}{|C_i|}$$

The F-measure of the clustering is defined as:

$$\sum_{i=1}^{k} F(i) \cdot \frac{|C_i|}{|C|}$$

The F-measure score is in the range $[0, 1]$ and a **higher** F-measure score implies a better clustering. For a more in-depth introduction and justification to the F-measure, see e.g. [31, 23, 8, 24].

Entropy: For each cluster $\hat{C}_j$, we define the entropy of $\hat{C}_j$ as:

$$E(\hat{C}_j) = \sum_{i=1}^{k} -\left(\frac{|C_i \cap \hat{C}_j|}{|\hat{C}_j|}\right) \log\left(\frac{|C_i \cap \hat{C}_j|}{|\hat{C}_j|}\right)$$

The entropy of a cluster is a measure of the disorder within the cluster. As such, a **lower** entropy score implies that a clustering is better; the best possible entropy score is 0. Entropy was first introduced in [26] and has been used as a measure of clustering quality in [9, 15, 7].

The entropy of a k-clustering $\hat{C}_1 \dots \hat{C}_k$ is the weighted sum of the entropies of the clusters. The entropy of a hierarchical clustering $\{\hat{C}_1 \dots \hat{C}_l\}$ is the minimum entropy of any choice of k nodes that partition C.

Accuracy: The accuracy of a cluster $\hat{C}_j$ is:

$$A(\hat{C}_j) = \max_{i=1}^{k} \frac{|C_i \cap \hat{C}_j|}{|\hat{C}_j|}.$$

As before, the accuracy of a k-clustering $C_1 \dots C_k$ is the weighted sum of accuracies. The accuracy of a hierarchical clustering is the maximum accuracy of any choice of k nodes that partition C. Note that the range of an accuracy score is between 0 and 1; the **higher** the accuracy score, the better.

Accuracy, which has been used as a measure of performance in supervised learning, has also been used in clustering (see [27]).

5.2 Convergence Proof

Proof (of Lemma 1). Since A is symmetric, we can write

$$A = \sum_{i=1}^{n} \lambda_i u_i u_i^T,$$

where the λ_i 's are the eigenvalues of A arranged in the order - $|\lambda_1| \geq |\lambda_2| \dots |\lambda_n|$ and the u_i are the corresponding eigenvectors. Express v in this basis as $v = \sum_i \alpha_i u_i$, where $\sum_i \alpha_i^2 = 1$. Since, v is random, we have that with probability at least $1 - \delta$, $\alpha_1^2 \geq 1/(n \ln(1/\delta))$. Then, using Hölder's inequality (which says that for any $p, q > 0$ satisfying $(1/p) + (1/q) = 1$ and any $a, b \in \mathbb{R}^n$, we have $\sum_i a_i b_i \leq (\sum_i |a_i|^p)^{1/p} (\sum_i |b_i|^q)^{1/q}$), we have

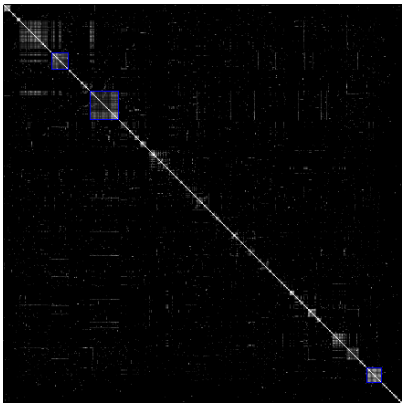
$$||A^k v||^2 = \sum_i \alpha_i^2 \lambda_i^{2k} \leq \left(\sum \alpha_i^2 \lambda_i^{2k+2}\right)^{k/(k+1)}$$

where the last inequality holds using Hölder with $p = 1+(1/k) \quad q = k+1 \quad a_i = \alpha_i^{2k/(k+1)}\lambda_i^{2k} \quad b_i = \alpha_i^{2/(k+1)}$. Note that:

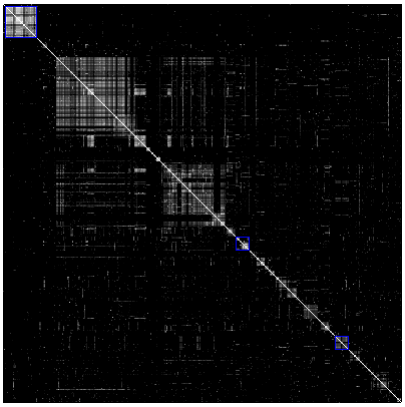
$$\left(\sum \alpha_i^2 \lambda_i^{2k+2}\right)^{k/(k+1)} \leq \left(\sum \alpha_i^2 \lambda_i^{2k+2}\right) / \lambda_1^2 \alpha_1^{2/(k+1)}$$

from which the lemma follows. □

5.3 EigenCluster example searches



(e) query:trees



(g) query:raleigh