COMPARISON OF CLUSTERING ALGORITHMS AND ITS APPLICATION TO DOCUMENT CLUSTERING

Jie Chen

A DISSERTATION

PRESENTED TO THE FACULTY

OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE

BY THE DEPARTMENT OF

COMPUTER SCIENCE

May 2005

Abstract

We investigate the methodology to evaluate and compare the quality of clustering algorithms. We study the issues raised in evaluation, such as data generation and choice of evaluation metrics. We give head-to-head comparison of six important clustering algorithms from different research communities.

We generate data using an extended mixture of Gaussian model that controls data characteristics such as shape, volume and size of a class. The added control facilitates more thorough exploration into the parameter space of the generative model and therefore makes the algorithmic evaluation more comprehensive.

We summarize datasets by their data characteristics. Previous comparison conclusions based on specific datasets are hard to be applied to other datasets. In contrast, conclusions based on data characteristics can be easily generalized. Therefore, we can predict the performance of an algorithm. The prediction is a significant step forward for comparison studies.

We compare three evaluation indices from different research fields. We recommend the f-score measurement if a single evaluation index is used. However, it is more desirable to apply multiple indices and vote among them to determine the ranking of algorithms.

We isolate the objective function (sum-of-squares) and the optimization approach (greedy) used in the popular k-means algorithm. We investigate the effectiveness and efficiency in the four algorithms resulting from substituting another objective function (min-max cut) and another optimization approach (Kernighan-Lin). We illustrate by a quantitative study that contrary to conventional wisdom, k-means is not only fast but also produces quality clusters. It achieves 95% of the best quality among the candidate algorithms running in time an order of magnitude faster.

We present detailed studies of the algorithmic behavior in response to data characteristics. We give an alternative definition of cluster separation and show that the new definition measures the degree of cluster difficulty for spherical and ellipsoidal clusters more consistently.

We develop and systematically evaluate a practical version of a spectral clustering algorithm originally specified for provable guarantees of correctness. We observe that the modified algorithm can find perfect solutions when the clusters are well separated, where iterative algorithms such as k-means tend to miss the perfect solution.

Acknowledgement

Part of this dissertation was supported by the Dean of Engineering School fellowship from Princeton University. Another smaller part was supported by two fellowships from DIMACS, the center for Discrete Mathematics and Theoretical Computer Science.

I received so much help from people around me that sometimes I wonder what I do in this life or the past one that I deserve all this. I couldn't thank Andrea enough. This thesis would not come into being without her. She discussed with me. She offered constructive suggestions. She proofread at least seven times. She encouraged me when I lost focus. She exemplified perfect ethics and discipline. I cannot describe how much I have learned from her, both professionally and personally. Words are too impotent when it comes to those indelible marks left on heart and soul.

I thank members of my committee: Rob, Moses, Brian and Mona. Rob offered so much expertise on machine learning. He gave comments and identified conferences where I should send my papers. He made time in his busy schedule to finish reading my thesis so that I can catch the commencement. The same happens with Moses. He observed the deadline imposed by me and trusted me completely with revisions. Brian and Mona enlightened me with their insightful comments. My gratitude for the gracious assistance of Melissa to put everything together, not only graduation but also grad school life.

Special loving thanks to so many friends who make my life in Princeton so memorable. The trips and weekends we spent together were so care free that they are etched in memory forever.

My deep and heartfelt thanks to my parents. They gave me endless love and unflagging support. They encouraged me to follow my heart and do whatever I want to do with life. They taught me sharing, tolerance and forgiveness. I cannot think of anything better than the values they have and they instilled in me.

Lastly, to Poon. He is always there to quench my anxiety. He argued with me for points in this thesis even though he is not a computer science major. He taught me not to be shy and ask great questions. I am so fortunate to meet him in the second formative years in my life. He led me to where I am now. For all the joy and sorrow, thank you.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Clustering has many applications. Document collections are clustered offline to improve the retrieval performance [57, 82]. Customer records are clustered so that future marketing can direct to more likely targets [51]. Researchers cluster epidemic time and place to discover the path along which diseases spread [40]. Gene expression data are clustered to find new protein functions and pathways [11, 94]. In short, clustering is versatile and often used in exploring datasets.

The clustering problem can be described as dividing a given set of objects into groups so that objects inside a group are more similar to each other than objects belonging to different groups. Because of the popularity of clustering as a data exploration tool, it is not surprising that clustering algorithms are like mushrooms after spring rain. Some of them are new variants of older algorithms [14, 29, 50, 52, 60, 71, 108, 117]. Others are completely new inventions [4, 10, 13, 16, 18, 23, 54, 64, 76, 107, 113, 114, 116].

Considering the output structure, these algorithms can be divided into two categories: hierarchical and partitional algorithms [58]. Hierarchical algorithms produce a hierarchy of clusters, which is a sequence of nested partitions. At the bottom level of the hierarchy, each object is an individual cluster. At the root all objects belong to a single cluster. The

desired clusters are picked out by choosing a particular level of the hierarchy, according to some rule.

The partitional algorithms give a flat partition of the input dataset instead of a tree-like hierarchy. The partitional algorithms generally run faster compared to hierarchical ones because they do not compare every pair of objects. The partitional algorithms employ a wide range of techniques. K-means is arguably the most popular partitional algorithm because of its simplicity and ease of implementation [78]. However, it suffers the criticism of favoring spherical shapes of clusters [40, 58]. Therefore new partitional techniques are on the rise.

Clustering algorithms come from diversified research communities. The area of information retrieval works with document clustering algorithms, which focus on the ability to handle high dimensionality [3, 16, 23, 33, 46, 55]. Machine learning and statistics often focus on model-based clustering [8, 9, 62, 102, 103]. The field of databases works on clustering algorithms that can deal with a large amount of records [19, 38, 50, 51, 64, 86, 118]. Recently, computational biology and bioinformatics have witnessed the rising usage of clustering algorithms to reveal candidate hypothesis to be proven by biological experiments [11, 95].

It is an interesting question why there are so many clustering algorithms [39]. Clustering is not a well-defined problem. Most definitions of clustering lead to an NP-hard problem [1]. Therefore, we have to resort to heuristics. Heuristics seek a local optimum instead of a globally optimal solution. Moreover, the desirable properties of an ideal heuristic may not be satisfied all at the same time [68]. This leads to a

proliferation of heuristics because there would be an algorithm for each property that is left out. Not to mention the disagreement on what constitutes desirable properties.

People have been quite imaginative in inventing heuristics. There are the more traditional iterative hill-climbing procedures [41, 78]. There are more innovative heuristics, such as hash-based clustering [55], suffix-tree clustering [116], spectral cut [30, 96], and mixture solving [9], which we introduce in Chapter 2. Moreover, these heuristics often tailor to a specific application. For example, in large databases, the records that need to be clustered may not fit in the main memory of a computer all at once. There is a technique that builds up a tree to represent an initial clustering of the data, where the tree contains parameters that can be tuned so that it fits into the main memory [118].

Because of the diversity and specificity of the heuristics, we see quite a collection of clustering algorithms in the literature. Given a clustering problem, how do we choose among the candidates? Usually there are three factors to consider: time complexity, space requirements and quality. The size of datasets is crucial in choosing an algorithm. If the datasets are huge, which is not unusual in the Web age, any running time more than linear is unacceptable, which completely rules out the hierarchical algorithms, because they have at least quadratic complexity. In addition to the time factor, some techniques used inside a clustering algorithm pose severe requirements on space complexity. For example, a class of clustering methods called spectral clustering call a singular value decomposition (SVD) procedure to calculate the singular vectors of the input matrix. The SVD takes such a significant amount of memory that it imposes

limitations on the size of an input matrix. However, the space problem can be solved by sampling, which runs the algorithms on a subset of the huge dataset. A few sampling based algorithms can be found in these references [24, 56, 69].

Quality of clusters is the third factor when considering candidates. A clustering algorithm always produces clusters given an input. Different algorithms give different clusters with the same input dataset. We call these output clusters from an algorithm a clustering. It is a tricky question to answer which of these clusterings is the most appropriate for the input dataset. The goal of applying clustering to datasets is usually to explore. There is no objective answer as to which clusterings is more appropriate because semantic knowledge of the datasets is incomplete.

People make two assumptions to judge the quality of a clustering algorithm. First, they assume the performance of an algorithm on all possible datasets can be predicted from a selected subset of these datasets. Secondly, they assume the structure of these selected datasets can be obtained by manual checking. Therefore, the quality of clustering algorithms is determined by running the algorithms on a few datasets with known structure.

For example, in document clustering, the structure of a dataset usually means the categories of the documents. Articles from newspapers can be grouped into categories such as sports and entertainment. After collecting articles, a human reads these documents and gives them a category label. These human-defined category labels constitute the inherent structure of the dataset. These categories are called classes in the

classification literature. In contrast, the groups that are found by a clustering algorithm are called clusters. We stick to this tradition in this thesis.

In this setup of the problem, quality of a clustering algorithm is defined to be how well it can discover the inherent structure of a dataset. There are all kinds of inherent structures. For example, describing objects by vectors in high dimensional space, the classes can be hyper-spheres, hyper-ellipsoid or any arbitrary shapes. The ability of a clustering algorithm to reveal these inherent structures is crucial in determining which algorithm to apply. An evaluation index is used in these situations to gauge the degree to which the output from an algorithm matches the inherent structures.

In this thesis, we focus on the quality of a clustering algorithm. We investigate the methodology that evaluates and compares the quality of clustering algorithms. We study the issues raised in the evaluation of a clustering algorithm, such as data generation and choice of evaluation indices. We give head-to-head comparison of six popular clustering algorithms, coming from different research communities.

When evaluating the quality of a clustering algorithm, even given the category labels, there are still issues to resolve, relating to the methodology to evaluate an algorithm. The common way to compare two or more algorithms begins by picking one or more real-world datasets. These datasets are usually benchmark collections with category labels so that the results from an algorithm can be compared to the labels using some criterion. However, such practice of comparison draws severe criticism. The opponents to the methodology argue that each algorithm is better by some criterion. The conclusion can change when substituting another dataset, even for the same algorithms.

Even worse, the conclusion can change if the evaluation index changes to another. Therefore, there are few comparison studies in the literature because the conclusions are not portable to other datasets. Recent papers that propose new algorithms are going to the other extreme by rejecting real-world comparisons all together. Most of them offer a comparison with older and more popular algorithms using only a few hand-crafted examples, showing where the old fails and the new succeeds [19, 38, 52, 54, 55, 64, 86, 95, 118].

We would like to compare algorithms on a wide range of possible situations so that the behavior of the algorithms can be understood as much as possible. Comparing algorithms by showing a few hand-crafted examples is not enough because these examples only reveal limited amount of information about the algorithmic behaviors. Selected examples do not offer systematic comparison of the algorithms on a wide range of parameters. By systematic we mean a fairly exhaustive sampling of the corresponding parameter space. Only when we compare algorithms systematically, we can predict their performance on previously unseen datasets. If we don't have confidence that we have observed the algorithms on enough datasets, it is hard to conclude anything.

In a sense, the key problem of the common comparison methodology is also comparing by examples. Only this time, the examples are the real-world datasets. Using only a handful real-world datasets to compare algorithms is equivalent in the spirit of limited exposure to using a few hand-crafted two dimensional examples. The few real-world datasets serve the same role as the hand-crafted examples by showing how the

algorithms perform in selected situations. It is not clear how these real-world datasets represent the vast amount of datasets out there. It is not clear how the algorithms perform on other datasets given the performance on these few selected sample real-world datasets. This problem leads to the key criticism of the current evaluation and comparison methodologies.

If datasets are described using the vector space model, the parameter space to explore is the *d*-dimensional space for all datasets, where *d* is the number of dimensions for all the data points. It is impossible to exhaustively enumerate all cases in this space. Some kind of sampling has to be applied. Random sampling does not work because chances are most samples would be structureless, making our attempt to evaluate the ability of an algorithm to recover structure fruitless.

The natural thought goes to a generative model in this case. We propose a generative model that describes the underpinnings of the structures that we are seeking to recover. Then use this generative model to generate numerous cases by exhausting its parameter space. In this sense, we are investigating systematically the algorithmic behavior under all possible cases generated by this generative model.

In this thesis, we present a mixture of Gaussian model, extended from the literature, to describe the classes in d-dimension. The key idea behind using a generative model is to describe the spectrum that the real datasets are in. It could be that several descriptions fit the spectrum well. However, what is important is the idea that by coming up with a reasonable model, we can fairly exhaustively enumerate all possible situations on which to test the algorithms. Given the variety of applications clustering algorithms can be

applied to, it could happen that in different fields of research, another generative model is more appropriate, although a precise, intuitive and succinct descriptive model is hard to come by. Once the data spectrum can be thoroughly investigated, we see no problem of predicting the algorithmic behavior of an algorithm.

In this thesis, we give the most extensive and most exhaustive experimental, comparative study of clustering algorithms to date. We extend an existing generative model and control parameters of the model that correspond to data characteristics that no one else has controlled. The size of our synthetic datasets reflects realistic settings instead of toy cases found in the literature. We introduce a notion of separation, from the area of theoretical machine learning, to distinguish the level of difficulty to cluster, so that the algorithm behavior can be compared at the difficulty level most similar to those observed in real datasets. We also give a brief description, under the parameters of the generative model, of the data characteristics of the real datasets we use.

In short, we use both synthetic and real datasets to evaluate and compare clustering algorithms. We use synthetic datasets to test the algorithmic behavior exhaustively, which may not be evident when using only a few selected real datasets. The real datasets reveal if the generative model is an appropriate abstraction of real data characteristics. By using both kinds of datasets, we hope the conclusions and the observed phenomena of our study can be found on other unseen datasets.

Almost all comparisons we have found in the literature use a single evaluation index to compare. It is an interesting question how many conclusions drawn in the literature depend on the evaluation index used. In information retrieval, the most-used index

traditionally is the corrected Rand Index [58], which measures the percentage of pairs of objects that an algorithm places correctly, either into the same cluster or different clusters. Recently, f-score has become more popular in studies of clustering algorithms for document clustering [72, 119]. Sometimes accuracy is used, which comes from the classification literature [97]. If we were to change the evaluation index for a study, would the conclusion change? Would the dominance of one algorithm disappear with a change of an evaluation index? Given a few indices, if we have to choose a single one, which one should we pick?

These are hard questions. To answer, we give a series of experimental analyses on three most popular indices. We identify desirable properties an evaluation index should possess, such as smoothness of the value distribution so that no sudden change of judgment is possible, and distinguishability among clusterings with similar quality. We calculate the value distribution of indices under random labelings and observe the empirically expected value for an index if labels are randomly assigned. We evaluate algorithm results for synthetic datasets with three indices and make various plots to show where these indices differ in the ranking of algorithms. Our study of evaluation indices goes well beyond previous studies; we identify only two such studies in the literature, which only compare pairs of indices by calculating the correlation coefficient between them [80, 81].

In this thesis we focus on document clustering algorithms. The desiderata for this focus are two fold. First, documents are usually represented by high dimensional vectors, which are hard to visualize. It makes systematic and scientific evaluation and

comparison of document clustering algorithms necessary. In fact, as far as we know, we only find comparison studies of clustering algorithms in this area [90, 100, 119], indicating it is the only application area that has given the problem of evaluation and comparison serious thought. Secondly, with the advent and development of the Web, there is urgent demand for good clustering algorithms, in terms of both time complexity and quality. Initially people are surprised to find that clustering algorithms can facilitate searches and reveal unexpected patterns in electronic commerce. However, later on they start to ponder on the relative goodness of the numerous algorithms in the field.

Whatever the application area we focus on, we want to make it clear that many ideas and arguments in this thesis apply equally well to other application areas. In fact, many of the arguments in this introduction suit the general area of clustering algorithms. Just like ideas behind algorithms can be carried over different application fields, ideas behind evaluation and comparison can apply across fields too. We focus on a particular area to specify concrete contexts for the problem of evaluation and comparison of clustering algorithms. It is our hope that similar framework can be established in other active clustering research areas.

Clustering being such a universal problem, there are many variants of it. For example, each object in the input dataset can belong to a single output cluster or it can have a degree of membership to several output clusters. If an object can only belong to a single output cluster, this is called “hard” clustering. Otherwise, it is named “soft” clustering. In this thesis, we focus only on the hard clustering problem, even though we agree that soft clustering sometimes gives a more intuitive interpretation of a real world problem.

However, when first attacking a problem, we would like to keep its form as simple as possible.

We work with a vector space model in this thesis. This model is the most popular one in document clustering [91]. It uses vectors to represent documents. Each dimension of a vector corresponds to a term. The number of dimensions of a vector is the number of unique words appearing in the total collection. The value for each component indicates the frequency of this term appearing in the document. Behind these application-specific interpretations, a document vector is a real-valued feature vector, which many other classification and pattern recognition algorithms work with. In contrast, there are vectors whose components are categorical values. For example, attributes of an object can have non-continuous values, such as color and material. We can translate these values of attributes to integers. However, the salient feature of algorithms for categorical values is their robustness to the sparseness of vector components [51]. In this sense, algorithms for categorical values share common features with document clustering algorithms. However in this thesis, we focus only on vectors with continuous real-valued components, which are called ratio data in the literature.

We compare two families of clustering algorithms: iterative partitional algorithms and spectral clustering algorithms. We focus on partitional algorithms because their linear time complexity or potential for linear time complexity is more desirable, considering the large amount of data available nowadays.

By iterative partitional algorithms we mean the family of traditional iterative hill-climbing algorithms, such as k-means. K-means has many variants in terms of

implementation details [88, 119]. However, it has three key components: initial assignment of a rough clustering, iterative refinement by moving objects to a more suitable cluster and the objective function to minimize, which is the sum of distances from every object to its closest cluster center, i.e., "sum of squares". K-means suffers criticism of favoring spherical shapes and splitting large clusters when dramatic cluster size difference is present [51, 58]. It is believed that k-means sacrifices quality for speed [111]. Therefore, there are other criteria which are claimed to have shape and size independence. We choose to compare the sum-of-square objective function to a minimum-cut based objective, the min-max cut, which shows signs of moderate success in the literature [30, 95, 96]. Besides the substitution of the objective function, we also substitute for the greedy optimization approach used in k-means, using an approach that accepts certain worse moves in order to achieve a better local optimum than greedy, the Kernighan-Lin approach [65]. By these substitutions, we hope to reveal the trade-off between quality and time complexity quantitatively.

However, as our experiments revealed, the aforementioned criticism of k-means is misleading. First of all, when given datasets with different sized classes, the min-max cut objective gives much worse clusters than sum-of-squares. Secondly, we generate datasets with spherical and non-spherical shapes, some of which are rather elongated. We would expect the min-max cut objective, which claims shape independence, does not give significantly different performance in terms of quality between the spherical and non-spherical datasets. In fact, it gives better f-scores in the non-spherical cases than the spherical cases. This suggests shape is a difficult issue for these objectives to

overcome. The new objectives, such as the min-max cut, are not meeting the expectations.

Furthermore, when comparing the four iterative partitional clustering algorithms, the algorithm of k-means turns out to be the one with best trade-off between quality and time complexity. It produces clusters with 95% of the quality achievable by the best candidate, sum-of-squares with Kernighan-Lin. But the running time of k-means is an order of magnitude faster than that of sum-of-squares with Kernighan-Lin. Therefore we conclude that k-means is a fairly good algorithm, with little sacrifice in terms of quality, given its linear running time.

Spectral methods are rising stars in recent literature [5, 33, 61, 63, 67, 87, 96, 109, 113]. We compare two spectral clustering algorithms, both with each other and with the iterative clustering algorithms. The first spectral algorithm is designed to optimize a minimum-cut based objective similar to min-max cut, that is, normalized cut. It recursively partitions the input graph according an eigenvector of the graph. We show that the algorithm cannot produce as good clusters as those by k-means.

The second spectral algorithm is borrowed from theoretical machine learning. The original algorithm makes assumptions about input data [109]. We make certain modifications to make it applicable to general clustering problems. The modified algorithm, which we give the name Cookie Cutter, gives perfect clusterings when the classes are well separated, in contrast to the almost-there solutions given by the aforementioned iterative algorithms. However, when the classes are overlapped, the modified spectral algorithm does not work as well as the iterative family of algorithms.

The procedure of singular value decomposition used in the algorithm poses severe memory requirements on the size of input matrix. Therefore the algorithm can be applied to only a single real dataset. We have to reduce the dimensions of the real datasets to apply the Cookie Cutter algorithm to them and compare with other algorithms. In the end, the f-scores for Cookie Cutter running on these dimension reduced datasets are not as good as those obtained for k-means.

In the remainder of this thesis, we introduce some background and previous literature in Chapter 2. The comparison framework and issues arising within the framework are presented in Chapter 3. In both Chapter 4 and 5, we present the details of our experimental studies, investigate the experimental results, and analyze the relationships among objective functions, optimization approaches and data characteristics. Spectral clustering algorithms are the topic of Chapter 6. Conclusion and future work can be found in Chapter 7.

Chapter 2 Literature Background

In this chapter we review related work in the literature. The chapter begins with some summarization of the clustering literature. After the overview, we delve into four different areas that are closely related to the topics we cover in this thesis: (1) methodology of comparison; (2) data generative model; (3) evaluation indices; and (4) spectral clustering algorithms. The first three areas are closely related to comparison of clustering algorithms. They touch on the methodology of and the issues arising in a comparison framework. The fourth area introduces a family of algorithms, which we would compare with some other algorithms using the comparison framework we develop.

2.1 Overview

The area of clustering is teeming with publications. It is an important tool in exploratory data analysis. The field accepts the following description of the problem. Clustering is a problem that tries to divide objects into groups, such that objects in the same group are more similar to each other than objects in different groups.

The description leaves an important question open. There is no definition of what is similar and what is not. Moreover, most similarity definitions lead to optimal solutions that are NP-hard to find [1]. Therefore, people focus on developing efficient heuristics. Different cluster definitions and various heuristics cause vast number of algorithms in this field.

By no means do we claim that our review here is exhaustive due to the sheer amount of literature. Given the cross-disciplinary nature of the problem, interested reader can refer to textbooks in several fields [6, 40, 58, 89, 104]. In this thesis, we focus on the fields of information retrieval, machine learning and databases.

Clustering algorithms differ in the data representation, the similarity measurement and the methodology to find the clusters. Input datasets can be spatial data consisting of location coordinates and values [54, 86]. Or they can be time series data that contain a time stamp and a value [47]. Different algorithms are developed catering to these different data representations. In this thesis, we assume the input data can be translated into a vector format. Each vector corresponds to an object to be clustered. Each component of a vector represents an attribute. The values of each component represent the values for each attribute. We assume the values are continuous and ordered. Certain normalization schemes can be applied so that they become unit vectors. We will return to the topic of preprocessing later in this chapter.

There are many similarity measurements floating around. For ratio data, whose attribute values are defined on the domain of real numbers, the most common is Euclidean distance [58]. The cosine similarity measure is also popular [58]. Algorithms based on

graph theory concepts often use minimum-cut or maximum flow to measure the similarity between two subsets of nodes [30, 42, 64, 95, 96, 107]. Sometimes people count the number of length-2 paths between two nodes, which is the number of common neighbors they have [51]. There is an old experimental study that concludes the similarity measurement does not affect the retrieval performance much [112]. However, it is not clear whether the conclusions of that study can carry to the current world, because computing has certainly improved in terms both of resources and the ability to analyze. We leave as an open problem to revisit the effects of similarity measurements on the performance of clustering algorithms.

The methodology to find clusters is the focus of this thesis. It is the most portable part of a clustering algorithm. Even if some specific algorithm cannot be carried over to another context, an algorithm designer can still borrow the idea.

Given vectors as input, there are mainly four different interpretations.

1. Points in High Dimensional Space

The first interpretation sees the vectors as points in high dimensional space. People come up with succinct mathematical definitions to define a cluster. For example, the popular k-means algorithm defines a cluster by assigning points to its closest center [78]. To find these clusters, traditionally there are two types of methodologies: hierarchical and partitional [58]. Hierarchical methods can produce a dendrogram, which is a tree-like structure, of the input data. The final clusters are produced by selecting a certain level in this dendrogram. The dentrogram can be generated either top-down or bottom-

up. The most popular method is the agglomerative hierarchical method, which builds up the dendrogram by repeatedly merging the most similar pair of clusters, starting from individual clusters containing a single point [111]. Some popular hierarchical agglomerative algorithms include single-link, complete-link and UPGMA [111]. In contrast, the partitional method seeks a flat partition, instead of a hierarchy, of the dataset. It first defines an objective function to optimize and then develop a heuristic to find a good enough solution to approximate the global optimal solution to the objective function. One family of heuristics starts from a rough partition of the data and then iteratively moves points among partitions to achieve a better objective value.

2. Samples Drawn from a Probability Distribution

The second interpretation assumes these vectors are samples drawn from a distribution, which in turn combines several other distributions. In the literature, two types of underlying distributions can be found: Gaussian and Poisson [9]. However, Gaussians are the dominant choice because of its widespread usage in the physical sciences and the central limit theorem [74, 105]. In this interpretation, each class in a dataset comes from a single Gaussian distribution. It is the job of an algorithm to estimate the parameters, that is, means and variances, of the distributions of all classes and the proportional presence of each distribution in the dataset. In the end, all the objects are assigned to clusters by the Bayesian rule. This is called model-based clustering, which is used frequently in the area of machine learning [8, 9, 14, 62, 102, 103]. The most common algorithm for model-based clustering is the EM (Expectation Maximization) algorithm [28]. It can get stuck in local optima and can potentially be very time

consuming. If we assume the underlying Gaussian distributions have the same co-variance matrix that is a multiple of the identity matrix, it is equivalent to optimize the minimum variance criterion used in k-means [9].

3. Vertices in a Graph

The third interpretation approaches the problem from the graph theory perspective. One example is to treat the input vectors as nodes in a graph. The edge weight between two points is their similarity value. Then some graph theory concepts are applied to define clusters. For example, we can define a cluster in terms of the ratio between its cut value with respect to the outside world and the total edge weight within the cluster. Recently there are several papers that utilize minimum-cut based ideas to find clusters [30, 64, 95, 96]. They report moderate success in terms of quality of clusters. Another example uses maximum-flow to define the boundary between two clusters [42, 107], which is similar to the above minimum-cut based definition. There are connections between the algorithms using graph-theory notions and the hierarchical agglomerative algorithms. For example, the single link algorithm, which defines the similarity between two clusters as the closest distance between the member objects, gives equivalent clusterings as the minimum spanning tree clustering algorithm [48].

4. Columns in a Sample Matrix

The fourth interpretation forms a matrix using these input vectors as columns. The singular vectors of the input matrix are used to reveal the structure of the input data. Some use the negative and positive components of the largest singular vector to bi-

partition the input vectors [30]. Some use the top ranking components of singular vectors to define multiple clusters [63, 67]. There are even hybrid algorithms that first project input vectors into the spectral space defined by the top k singular vectors and then using traditional clustering algorithms, such as k-means, to cluster the projected points in spectral space [61].

K-means is arguably the most popular algorithm, because of its simplicity and ease of implementation [78]. Starting with a rough partition of the data, in a single round, it moves each object to other clusters so that to lower the objective function, that is, sum-of-squares, it optimizes. The running time of k-means depends on the number of rounds and the time spent in each round. In practice, K-means converges rather fast and therefore only a few rounds, usually irrelevant to the number of input objects, are needed. The time spent in each round is linear to the number of objects. Thus most times, people treat k-means as a linear time algorithm. There is more detailed introduction of the algorithm in Chapter 4.

There are two main complaints about k-means: it favors a spherical shape of clusters and it does not do well for different size clusters [51, 58]. K-means seeks to find a clustering that minimizes the sum of the squared distance from each point to its closest cluster center. Using the distance to a cluster center to indicate the edge of a cluster favors a convex shape, therefore missing those clusters that can be concave or even linear. When the clusters have significant size difference, points belonging to the large cluster can be closer to the cluster centers of a nearby small cluster than the center of

the large cluster. In fact, many newly developed algorithm cites these two shortcomings to boost the one they are proposing [38, 50, 51, 64, 86, 118].

In this thesis we are evaluating how much data characteristic factors of this kind contribute to the quality of clusters produced by different objective functions, including the one that k-means uses. We compare the objective function that has such shortcomings with other objectives that claim they do not, to see the degree of quality that algorithms are sacrificing.

2.2 Comparison Methodology

In the nineteen seventies, when cluster analysis was enthusiastically explored in psychology, biology and library sciences, researchers attempted to compare the quality of the clustering algorithms on different data sets [12, 22, 34, 70, 80, 83, 84]. The comparison process typically was broken into three stages: (1) data was generated or hand-crafted; (2) a set of clustering algorithms were chosen and clusters were produced; and (3) an evaluation index determined how well each clustering recovered the original structures.

The earliest study we found uses 6 hand-crafted data sets, each of which contains 20 2-dimensional points [22]. Later studies start to use Gaussian distributions to model points inside a cluster [12, 70, 80]. The details of their generation process appear in the next section. The set of clustering algorithms these studies choose is the family of agglomerative hierarchical clustering algorithms, including single link, complete link,

median link, centroid link, group average and Ward's method [58]. The evaluation indices are variants of the Rand index [58]. They all conclude single-link is the worst in terms of cluster quality. Two of the studies conclude Ward's method is the best when cluster sizes are equal [70, 80] while the other study concludes it is the best across all data sets [12].

Recently agglomerative hierarchical and partitional hierarchical algorithms have been compared using real data [119]. The study experiments with several objective functions besides the sum of squares, including group average and min-max cut. F-score is used to measure the degree to which the clustering process recovers the original class labels. Twelve real-life document datasets are clustered. The conclusion is that the partitional hierarchical algorithm with sum-of-squares objective function is the best in terms of the quality of clusters.

We search the literature and find these are the few comparison studies available. The reason for so few studies is that the conclusion of a comparison study depends on the choice of evaluation indices and the choice of datasets. (We discuss the choice of evaluation indices in detail in later sections of this chapter.) If we apply the same algorithms on different datasets, their relative ranking in terms of performance can change. It makes the conclusions from one study hard to generalize. Therefore there is doubt on any comparison study.

Even though there are few systematic comparisons of clustering algorithms, people keep inventing new algorithms and adapting older ones to a new environment. Papers on new algorithms usually include some experimental results on its performance

comparing to some older algorithms the authors deem relevant. The algorithms they choose are often the ones that they find fault with. The comparison between the two algorithms focuses on the point where their own algorithm is better. Most of the comparisons are conducted using hand-crafted datasets with two dimensional points so that the results of different clustering algorithms can be visualized. To illustrate this point, we list in Table 2.1 some recent papers with original ideas and experimental studies in the related field, together with the evaluation methodology, and comparison details, such as the datasets they use and the other algorithms they compare.

Table 2.1 Details of Comparison Results for Recent Clustering Algorithms

Algorithm name/year	Datasets Used to Compare	Evaluation Methodology	Other Algorithms They Compare to	Criticism of Other Algorithms
Spectral Cut(2001)	3 2-cluster datasets (each with 2 newsgroups)	Accuracy (undefined)	NCUT; Use first principle component to divide	Skewed cut when large overlap between clusters
ROCK (2000)	2-cluster data sets; 10-cluster synthetic datasets	List cluster contents; misclassified transactions (undefined)	Single link	Sum-of-squares split large clusters
Chameleon (1999)	5 datasets with 2-d points	Visual inspection	CURE DBSCAN	k-means cannot handle different sized and non-sphere clusters
CURE (1998)	2-d points, 2 datasets	Visual inspection	Single link BIRCH	Sum-of-squares split large clusters

Suffix-Tree(1998)	There own snippet collections	Precision/Recall	K-means, Group Average,	k-means favors spheres, group average slow
NCUT (1997)	1 2-d datasets, 4 images	Visual inspection	None	Previous objectives based on local properties
BIRCH (1996)	Generated datasets with 2-d points, real example in image segmentation	Visual inspection	CLARANS	Memory assumption impractical
DBSCAN (1996)	Datasets with 2-d points	Visual inspection	CLARANS	Input params not known; limited on spherical shapes; not efficient on large datasets;
CLARANS (1994)	4 datasets with at most 100 points in 5 clusters	Average distance to cluster center	k-means on samples from original datasets	Quality relies stopping rules
Model-based Clustering (1993)	1 3-cluster 2-d generated dataset; 2 real datasets	Visual inspection	Single-link k-means	k-means favor spherical shapes
Scatter/Gather (1992)	5000 articles from new york times	Show a sample session with the system	n/a	Hierachical methods slow

We consider the evaluation and comparison methodology of current practice to be less than ideal. First of all, we discover in our experiments that performance of algorithms

on datasets with two clusters is far from representative. Some conclusions are even reversed when applying the algorithms to datasets with more than two clusters. Therefore it is not appropriate to use two-cluster datasets to compare clustering algorithms. Secondly, the technique of visual inspection involves human examination. It is more suitable for a few examples in the initial prototyping stages of an algorithm. For systematic evaluation of large amount of results, it is a better idea to have some objective measurements that can be calculated from the results. Thirdly, visual inspection requires two-dimensional datasets, or projection of high-dimensional datasets to the x-y plane. For applications such as document clustering, there can be hundreds of dimensions involved. Projecting to two dimensions to test the clustering performance is too limited and can even be misleading.

We advocate testing the algorithms under investigation using a systematic evaluation framework. To be specific, if synthetic datasets are used, we prefer a thorough and extensive collection of datasets with characteristics spanning the parameter space. The majority of recent comparison studies use realistic datasets [100, 119]. In this paper we argue both synthetic data and realistic data are necessary for quality comparison. Synthetic data sets offer thorough exploration of the data characteristic space while realistic data sets can help check the validity of the generative model assumptions and confirm the conclusions from theoretical studies. When judging which algorithm produces clusterings with better quality, an objective measure is desirable such that a computer can determine the quality of a clustering. This facilitates the evaluation of large numbers of test datasets.

Not only do we test using both synthetic and real datasets, we tie the algorithms under study with the data characteristics of the datasets we use to test. In this thesis, we summarize characteristics of datasets so that datasets can be described in more general terms. The behavior of an algorithm is analyzed by its response to the generalized data characteristics. Therefore we can predict the performance of an algorithm even for previously unseen datasets.

2.3 Data Generative Model

There are many ways to generate datasets. The most naïve way is to hand-craft a few datasets according to the points a publication wants to make. A few recent proposed clustering algorithms to cluster large spatial datasets in the database community resort to this approach [19, 38, 50, 52, 64, 86, 118]. Sometimes, for example, the generator used to test the algorithm CURE increases the number of points per cluster for a hand-crafted dataset to test the scalability of the algorithm [50]. However, the generator does not alter the number and placement of clusters.

A second option is to define an attribute pool and then randomly determine how many attributes are non-zero. The values for the attribute that are non-zero can either be pre-specified or random numbers. For example, the test for the algorithm ROCK, which produces clusters of categorical data, generates market transaction records this way [51]. When the number of transactions is large, the method is fairly exhaustive in terms of

covering different cases. However, its shortcoming lies in the inability to find the strength and weakness in the response of an algorithm to different situations in datasets.

Earlier studies in cluster analysis have used generative models based on mixture of Gaussians. The details of their generation process appear in Table 2.2. Note the Milligan study does not allow overlapping clusters.

Table 2.2 Details of Previous Generative Model

	Mean	Variance	Overlap
Blashfield [12]	Uniformly random from a range	Diagonal entries uniformly random from a range	Yes
Kuiper [70]	Compactly situated,	Identity matrices	Yes
Milligan [80]	Uniformly random from a range	Not mentioned in paper	No

To generate data sets we adopt the mixture of Gaussian model from this earlier work [74]. There are three main problems with the previous experiments using the model. First, they control limited parameters of the model. Previous work usually uses identity covariance matrices and the clusters generated are spherical. We generate positive definite covariance matrices and therefore include non-spherical clusters in the experiments. Second, the previous experiments generate small data sets due to limitations of available computing resources. For example their datasets contain dozens of data points in at most ten clusters. In contrast we generate large data sets that resemble realistic conditions with 1000 points and 20 clusters. Third, previously there has been no characterization of the level of difficulty for a data set, and people generate the parameters from a uniformly random distribution from an arbitrary range. We

introduce the notion of separation to represent the level of difficulty to cluster and devise a method to place the component Gaussians of the mixture in the space to satisfy the separation requirement.

2.4 Evaluation Indices

In this section, we review the past studies on how to evaluate a given clustering. An evaluation index is a function of the given clustering. Usually it produces a single number by which the user can tell which clustering is better.

Clustering algorithms produce clusters regardless of whether a dataset has inherent structure. (An example of a dataset with inherent structure is a collection of newspaper articles: it is natural that the articles belong to categories such as sports, entertainment and healthcare.) Therefore, some evaluation studies are dedicated to cluster tendency tests that determine whether a given dataset has any unusual internal structure, to justify the use of clustering algorithms on the dataset. In this thesis, we work with datasets with pre-specified labels and therefore do not have to test the datasets for its randomness. We do not discuss cluster tendency further, but interested readers can refer to the literature for related work [58, 75].

2.4.1 Categories of Evaluation Indices

Depending on the type of clusterings to evaluate, there are indices to evaluate a hierarchy and a flat partition [58]. Algorithms producing partitions are more popular in

recent studies because of their time complexity. Therefore we focus on evaluating partitions in this thesis.

External indices compare clusterings with the class labels so that a match between clusters and classes can be calculated. In contrast, internal indices only utilize the information within datasets. For example, internal indices usually compare cluster labels to the similarity matrices calculated from datasets, instead of class labels.

An internal evaluation index has a close relationship with an objective function. In fact, an internal evaluation index can be an objective function and vice versa. An objective function refers to the criterion optimized by an algorithm. For example, the popular K-means algorithm produces clusters by minimizing the sum of distances from each data point to its closest cluster centers. An internal evaluation index is the function calculated on the resulting clustering. One example is the trace of the sum of the covariance matrices for each cluster.

We can certainly treat the objective function used in an algorithm also as an internal index to evaluate the resulting clusterings. The problem is that when comparing clusterings from different algorithms, using the objective function of one of them would favor the particular algorithm and therefore produce misleading conclusions. On the other hand, given an internal evaluation index, we can always devise an algorithm to optimize the function of the internal evaluation index. However, the problem is that some internal indices are too expensive to compute repeatedly inside the execution of an algorithm. Hence, internal indices and objective functions remain two separate concepts.

It is hard to define quality of a clustering because of its subjectivity. People try to capture it by succinct mathematical definitions, which may not be accurate for all the cases [39, 43]. The objective function an algorithm optimizes reflects this definition. Similarly, an internal evaluation index defines what clusters are considered good. The goal of evaluation is to tell how well the clusters produced by algorithms recover original structures. However, it is not clear how well the definition of clusters used by an internal evaluation index captures the original structure. Therefore using an internal index measures only how close found clusters are to clusters pre-defined by the evaluation index, not how close found clusters are to the original structures in the data.

Using external indices can avoid this problem because in this case the original structures are given by the classes in input datasets, which are previously labeled or constructed by human beings. The comparison happens between the original structures and clusters. Therefore, in this thesis, we focus on external indices.

However, we have to note that external labels are expensive to obtain because of the significant amount of human examination involved. We download datasets with labels from Project Cluto [21] and characterize the performance of the algorithms by summarizing the characteristics of these datasets. In this way, we hope to predict the performance when algorithms are applied to datasets without labels.

2.4.2 Using a Single Evaluation Index

The conclusions drawn from comparison using a single index are subject to criticism of the choice of the evaluation index. If it is an internal index, it is not clear whether the

type of clusters that it defines to be good is the same as the type defined by the objective functions used in the algorithms. If it is an external index, the problem is less severe, however the argument still arises that the index favors a particular type of match between clusters and classes.

Unfortunately, almost all studies in the literature use a single evaluation index to compare algorithms, as far as we know. The only exception is the study conducted by Milligan et al in 1981 [81]. In this study, they gather 30 internal indices and evaluate four hierarchical agglomerative clustering algorithms on 400 generated datasets. The internal indices are then ranked by the Pearson correlation values with regard to two external indices. They conclude the top six indices are suitable choices because the correlation exceeds a certain level.

There is another comparison of evaluation indices involving the same author [80]. Four external indices are calculated on the same generated datasets. Correlation is calculated between pairs of indices. The conclusion is the four indices are fairly consistent, with correlation at least 0.85.

In this thesis, we choose three external indices from different backgrounds and apply them together to evaluate the clustering experiments we conduct on synthetic datasets. We compare the rankings induced by these indices and investigate the disagreement on rankings among the indices. It is an interesting study in the sense that the analysis is the first as far as we know to measure the effects of using different indices on the comparison conclusions.

2.4.3 Significance of an Index Value

The technique of hypothesis testing was the most used tool to evaluate clustering algorithms in the first twenty years of research on cluster analysis. It reflects the perspective that people were using to approach the problem of cluster evaluation. The technique zeroes in on the validity of applying clustering algorithms to datasets, instead of the goodness of the clusters. It came into existence so that other people would accept the usage of the algorithms. A cluster validity study assumes the problem of cluster validity is inherently statistical. We give a brief introduction on the vocabulary and mechanism of hypothesis testing. Please refer to statistics textbook for more details [58].

Given a statistic T and a null hypothesis H_0, and the distribution of T under this null hypothesis, we would like to determine whether H_0 is an appropriate description for the data at hand. In cluster validity studies, we would like to know if the fit of a hierarchy or a partition to the given data is unusually good. To be unusual, the fit must at least be better than the fit of a hierarchy or partition to a random data set.

Assume the larger the T value, the better fit. Let $P(T \geq t \mid H_0)$ be the probability of the event $T \geq t$ under H_0, where t is a fixed number called a threshold. Let α be a small number, such as 0.05 or 0.01, called the size or level of a test. Given the distribution of T under H_0, we can place a threshold t_α on T by solving the following equation:

$$P(T \geq t_\alpha \mid H_0) = \alpha$$

In summary, the hypothesis testing approach to cluster validity involves several steps. A null hypothesis expressing the idea of no structure must be defined. A statistic, or index, sensitive to the presence of structure in the data must then be selected and the distribution of the statistic under the null hypothesis must be established. A threshold can then be found that defines how large is "large" for the statistic selected. The threshold establishes a formal test of hypothesis. The power of the test evaluates the ability of the statistic to recognize the presence of a structure specified in an alternative hypothesis.

Early cluster validity studies focus on whether a given clustering reflects any structure. It is a yes or no question, without pondering on the relative quality of each clustering. Clusters produced by an algorithm either are valid, or not, because people focus more on the usability of clustering algorithms, instead of their performance. This reflects in the use of the framework of hypothesis testing. Only the distribution for the null hypothesis is used to make a binary decision.

Moreover, the hypothesis testing framework compares the fit of a given clustering to the original data to the fit of the same clustering to some randomly generated data. If the statistic is unusually large or small, the null hypothesis is decided against. The conclusion is that the fit is unusually good for an unstructured dataset. It is a detoured way of saying the fit is good.

The last problem of hypothesis testing is that the distribution of the index values under the null hypothesis is hard to establish. The distribution depends on each instance of the problem, such as the size of the input, and other parameters. When the size of input is

rather large, computing the distribution can be prohibitive. The distribution can be approximated by Monte Carlo analysis or bootstrapping, or even by estimating the moments of the distribution and approximating it by the normal distribution. However, it consumes time and resources. Worse, sometimes it is hard to determine which null hypothesis to use.

In this thesis, we focus on the relative goodness between clusterings. Our input data are human-labeled documents. Thus there is inherent structure in the input set. There is no need to test the validity of the clusters because the dataset is not random. Moreover, we feel the framework of hypothesis testing does not answer the key question for the relative comparison: how much more a gain in the observed values is necessary to declare a better algorithm. Therefore, we do not use the hypothesis testing approach.

However, we feel the setup can still be useful in addressing one open question in this thesis: given two evaluation values, how significant is the difference between them. In other words, we would like to know how much difference there should be between two values such that we can declare one is better than the other. Given the distribution of the index under the alternative hypothesis, the significance can be interpreted as the area under the curve between the two values. It would be an interesting problem to approximate the distribution and the significance value.

2.5 Spectral Clustering

Spectral clustering algorithms use eigenvectors or singular vectors of an input matrix to find clusters. The input matrix A can have many interpretations. Its columns can be the objects being clustered and rows their properties. For example in document clustering, the input matrix A is the document-term matrix where each column represents a document and each row is a word. The entry *A(i, j)* is the occurrence information of the *i*th word in the *j*th document. The input matrix could be square, such as the pair-wise similarity values for the objects to be clustered, or the adjacency matrix of a given graph.

There are several types of spectral clustering algorithms. We give brief introduction in the following paragraphs.

Spectral graph partitioning algorithms date back to the seventies [20]. They treat the objects to be clustered as the vertices of a graph. A similarity function is defined between two objects and used as the weights on edges. The input matrix to the algorithms is usually the adjacency matrix. If the input matrix contains no negative entries the non-principle eigenvectors have both positive and negative components. The assumption is that the nodes with positive components would not be densely connected with the nodes with negative components. Therefore the non-principle eigenvectors can partition the nodes of a graph.

An example of recent studies uses the spectral cut algorithm to separate foreground objects from background in computer vision and reports moderate success [96]. It uses the Fiedler vector, which is the second smallest vector of a generalized eigenvalue

system, to split the given graph into two parts. The split procedure is then applied recursively on the two induced partitions until certain stopping conditions are met.

We experiment with the spectral cut algorithm and discover that the components of the Fiedler vector fluctuate around the origin and therefore sometimes making the partition rather arbitrary. In our experiments the quality of clusters using the spectral cut algorithm cannot match those from other algorithms such as k-means.

In the popular hubs and authorities algorithm of Kleinberg [67], the input matrix is the adjacency matrix A of the underlying directed link graph. The principle eigenvector of A^TA is the authority score of all the nodes in the graph while the principle eigenvector of AA^T is the hub score. Moreover in their experiments they find the large positive and negative end of non-principle eigenvectors respectively form communities that illustrate the more obscure meanings of the search term. However the communities gleaned from different non-principle eigenvectors overlap each other. Therefore the algorithm requires human intervention to list all the clusters.

Kannan and Vempala propose a simpler approach to utilize the eigenvectors [63]. They work with the object-property matrix, which is rectangular. The algorithm groups the rows of the input matrix A into clusters. The top k singular vectors of A form another matrix S, with each column a singular vector. Instead of checking into the content of the singular vectors, they assign the ith row to the jth cluster if S(i, j) is the largest for the ith row of S. Ng et al reports that this algorithm does not find satisfactory clusters on a few hand-crafted datasets they devise [85].

In the aforementioned algorithms the components of an eigenvector or a singular vector are used to reveal the underlying graph structure. Some even argue the singular vectors themselves convey a "generalized clustering", where each component is the degree to which each node belongs to this cluster and each node can belong to more than one cluster [33]. The allowance of overlapping and fuzziness of clusters are useful in some applications. For example, it is natural for a document to have multiple topics. However, we focus on the algorithms for "hard" (versus "soft") clustering in this thesis, where the input set of objects is partitioned into exclusive groups [52].

Traditionally, calculating the eigenvectors or the singular vectors takes at least quadratic time [45]. Fortunately there are fast monte-carlo methods that calculate the decomposition of a submatrix sampled from the original large one according to a certain probability distribution [44], so that linear time clustering algorithms are possible using spectral methods. This becomes an important advantage for these algorithms since linear time complexity is a must for algorithms dealing with large datasets, which are ubiquitous nowadays.

Not only the singular vectors themselves can be used for clustering, projecting the input points to the subspace spanned by the top k singular vectors helps also. We denote this subspace the spectral space. If the mixture of Gaussian model is used to describe the data, it is recently proved that projecting to the spectral space can maintain the distance between the centers of Gaussian clouds and compress the radius of each cloud [109]. Effectively the Gaussians become more separated and more spherical after the decomposition. Therefore the clustering problem in the projected space would be easier

presumably. This is an interesting discovery because it means various algorithms can be used after a decomposition to improve the quality of a clustering, if the data is believed to come from a mixture of Gaussians.

In fact, a recent paper stacks the top k eigenvectors of the input matrix A together to form an intermediate matrix for the K-means algorithm to work on [61]. The rows of the intermediate matrix serve as the input points to a traditional K-means algorithm. Surprisingly, the hybrid algorithm yields better results than directly applying K-means to the 20 newsgroup data, evaluating by an adjusted Rand index. A detailed study on different combinations of existing techniques from spectral and combinatorial clustering algorithms show that mapping points into spectral space helps combinatorial techniques to find the perfect solution when the number of classes and clusters are both smaller than 5 [110].

In short, spectral clustering is an active field that has gained attention recently. There are algorithms in this field that show promise, although most experiments are small-scale, hand-crafted and lack systematic comparison with traditional favorites. We can do pure spectral clustering, which involves only the spectral properties of input, or we can devise hybrid algorithms, such as using k-means or other algorithms in the spectral space.

In this thesis, we focus on both pure and hybrid spectral methods. The first algorithm we consider optimizes a minimum-cut based objective function by using eigenvectors to partition a graph. The second algorithm projects input points to the spectral space and then applies distance-based clustering to the projected points. We hope the direct and

systematic comparison of these spectral methods with the iterative partitioning methods can help future spectral algorithm designers to focus their efforts accordingly based on our findings.

Chapter 3 Comparison Framework

In this chapter we describe the methodology we choose to compare clustering algorithms. We run the algorithms on labeled datasets so that the clusters produced by the algorithms can be compared with the labeled classes in the datasets. Using labeled classes avoids the tricky problem of defining the true clusters and therefore has wider spread of usage.

However even with the class labels we still have to determine the degree to which the clusters match the classes so that we can know which algorithms produce better quality clusters. In previous reported studies people have usually used a single evaluation index to calculate the match. We raise the question of whether a single index is enough. The rankings of algorithms could change if using different evaluation indices. We choose three indices from different backgrounds to evaluate and give a detailed analysis on the behaviors of the three indices on data.

Comparing algorithms could be rather ad hoc. There are plenty of datasets that these algorithms can run on. Chances are there are some datasets for each algorithm to excel. Therefore it does not suffice to run algorithms on certain datasets and conclude. It is

important to summarize the properties of different datasets and make predictions on when these algorithms would excel. To achieve this we need to fully test the algorithms on a wide spectrum of data. To supply the data we propose a data generative model, which extends the traditional generative model in the literature, and gives fuller control of the data characteristics. We describe the model in detail in this chapter.

Another source of our labeled data is realistic document collections we find on the Web. Most of them are articles from newspapers, with categories labeled by humans. They are of moderate sizes, with thousands of documents and tens of thousands of words per collection. We give more detailed description of the characteristics on these datasets in this chapter.

3.1 Synthetic Data

In this subsection we give rudimentary introduction to multivariate Gaussians. We describe the generative model to be used to create the synthetic datasets. We then explain the experimental setup.

3.1.1 Data Generative Model

To control the data characteristics for the comparison of the algorithms, we develop a data generation model that extends the mixture of Gaussians model. Unlike previous studies we control parameters of the model that correspond to data characteristics. We use the notion of separation to represent the level of difficulty to cluster so as to

thoroughly test the algorithm behaviors. We devise a method to place the component Gaussians of the mixture in the space to satisfy the separation requirement. The scale of our data sets resembles those in reality.

First we describe the mixture of Gaussians model. Then we move on to the details of generating a single cluster from a Gaussian distribution. We include a brief introduction to the multivariate normal distribution to clarify the notation and facilitate the discussion. Most of the introduction comes from the textbook by Duda and Hart [36].

3.1.1.1 Mixture of Gaussians

We choose the mixture of Gaussians model because of its simplicity and wide usage [35, 74, 105]. A mixture of Gaussians G_1, G_2, … G_k with non-negative mixing weights w_1, w_2, … w_k , where $\sum_i w_i = 1$, is the distribution in which a sample is produced by first picking a component Gaussian – the ith one is picked with probability w_i – and then producing a sample from that Gaussian.

"Data characteristic" is a general term, which may vary across fields and applications. We choose the characteristics that are intuitive and easy to control in our experiments, hoping to shed light on the relationship between objectives and certain data characteristics. The three data characteristics chosen are: (1) shape of the cluster; (2) volume of the cluster; and (3) size of the clusters.

The size of a cluster is how many points there are in the Gaussian cloud. It depends on the mixing weights of a mixture. We distinguish two situations: all clusters with the

same size, and each cluster with a different size. The mixing weights, when cluster sizes are not the same, are generated by randomly picking cluster sizes from 0 to the total number of points in a dataset and then normalizing the sizes by the total sum of all the generated sizes. When clusters are of the same size, the mixing weights are fixed, not generated randomly.

The details to control the other two factors, shape and volume, are in the subsection 3.1.1.3.

3.1.1.2 Multivariate Normal Distribution

The general multivariate normal density is

$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu})^t \Sigma^{-1}(\vec{x}-\vec{\mu})\right],$$

where $\vec{x}$ is a d-dimension column vector, $\vec{\mu}$ is the d-dimension mean vector, Σ is the $d \times d$ covariance matrix and $|\Sigma|$ is the determinant of the covariance matrix. For simplicity, we often write the density as $N(\vec{\mu}, \Sigma)$. The multivariate normal density is completely specified by $d + \frac{d(d+1)}{2}$ parameters, the elements of the mean vector μ and the independent elements of the covariance matrix Σ. Formally,

$$\vec{\mu} = E[\vec{x}]$$

and

$$\Sigma = E[(\vec{x}-\vec{\mu})(\vec{x}-\vec{\mu})^t],$$

where the expected value of a vector or a matrix is found by taking the expected values of its components.

The covariance matrix Σ is symmetric and positive semi-definite. We restrict our attention to strictly positive covariance matrix, i.e., $|\Sigma| > 0$. The diagonal element σ_{ii} is the variance of x_i, and the off-diagonal element σ_{ij} is the covariance of x_i and x_j.

If x_i and x_j are statistically independent, $\sigma_{ij} = 0$.

Samples drawn from a normal population tend to form a single cloud or cluster. The center of the cluster is determined by the mean vector, and the shape of the cluster is determined by the covariance matrix.

The quantity

$$r^2 = (\vec{x} - \vec{\mu})^t \Sigma^{-1} (\vec{x} - \vec{\mu})$$

is called the squared Mahalanobis distance from $\vec{x}$ to $\vec{\mu}$. Thus the contours of constant density are hyperellipsoids of constant Mahalanobis distance to $\vec{\mu}$. The principal axes of these hyperellipsoids are given by the eigenvectors of Σ, the eigenvalues determining the lengths of these axes.

It can be shown that the volume of the hyperellipsoid corresponding to a Mahalanobis distance r is given by

$$V = V_d |\Sigma|^{1/2} r^d ,$$

where V_d is the volume of a d-dimensional unit hypersphere:

$$V_d = \begin{cases} \dfrac{\pi^{d/2}}{\left(\dfrac{d}{2}\right)!}, & d \text{ even} \\ \dfrac{2^{d}\pi^{d-1/2}\left(\dfrac{d-1}{2}\right)!}{d!}, & d \text{ odd} \end{cases}$$

Thus, for a given dimensionality, the scatter of the samples varies directly with $|\Sigma|^{1/2}$.

3.1.1.3 Data Characteristics of a Single Gaussian

The shape of a Gaussian is determined by its covariance matrix [35]. If the covariance matrix is the identity matrix or a multiple of the identity matrix, the resulting Gaussian is spherical. Any positive definite matrix yields a non-spherical Gaussian. We obtain a random positive definite matrix by multiplying a full-rank random matrix with its transpose. A full-rank random matrix of any dimension can be generated by a standard Matlab routine, which generates according to a uniform distribution [101].

The volume of a Gaussian indicates the space the cloud occupies and is in direct proportion to the square root of the determinant of the covariance matrix [35]. However, the determinant of randomly generated covariance matrix is not stable. At times the Matlab routine we called to calculate the determinant returns the value of infinity, causing troubles in later computation. The reason could be that the generated matrices have very small values, therefore eliciting overflow problems while calculating the determinant.

We resort to using the largest eigenvalue of the covariance matrix to control the volume of the Gaussians. It happens the eigenvectors of the covariance matrix determine the orientation of the Gaussians in space. The corresponding eigenvalues controls the degree of scatter along these vectors. Therefore the volume of a Gaussian cloud responds to the magnitude of the largest eigenvalue of the covariance matrix. We find researchers also employ the same workaround in classification literature [9].

We distinguish between two scenarios: all Gaussians with the same volume, and each Gaussian with a different volume. When the volumes can vary, we randomly pick a number from 1 to 5, using the same random generator in Matlab, to be the largest eigenvalue of the covariance matrix for a Gaussian cloud. The choice of these numbers is rather arbitrary, as it describes roughly the relative degree of the scatter. When all Gaussian clouds have the same volume, we fix the largest eigenvalue to be 1, the unit volume.

3.1.1.4 Separation

The notion of separation comes from theoretical papers on learning mixture of Gaussians. A mixture of Gaussians is easiest to learn when the Gaussians do not overlap too much. Therefore the degree to which Gaussians are overlapping can indicate the level of difficulty for finding the true clusters. We take the following definition from Dasgupta [26].

Definition Two Gaussians $N(\mu_1;\sigma^2 I_n)$ and $N(\mu_2;\sigma^2 I_n)$ are considered c-separated if $\|\mu_1 - \mu_2\| \geq c\sigma\sqrt{n}$. More generally, Gaussians $N(\mu_1;\Sigma_1)$ and $N(\mu_2;\Sigma_2)$ in $\Re^n$ are c-separated if $\|\mu_1 - \mu_2\| \geq c\sqrt{n \ \max(\lambda_{\max}(\Sigma_1), \lambda_{\max}(\Sigma_2))}$, where $\lambda_{\max}(\Sigma)$ is shorthand for the largest eigenvalue of Σ. A mixture of Gaussians is c-separated if its component Gaussians are pair-wise c-separated.

A 2-separated mixture corresponds roughly to almost completely separated Gaussians, whereas a mixture that is 1- or 0.5-separated contains Gaussians which overlap significantly. The larger the separation, the easier to cluster.

In our experiments we generate both mean vectors and co-variance matrices for each Gaussian. The co-variance matrices will be generated according to the type of Gaussians we want, either spherical Gaussians or non-spherical Gaussians. The mean vectors, i.e., the center of Gaussians, will be generated and placed in $\Re^n$ at a given separation level. These data sets will serve as input to the clustering algorithms being compared. We record the clustering results so that we can compare the labeling by the algorithm with the original generated labels.

Given k and c, there are endless possibilities for a mixture being c-separated. For example, we could place the center of Gaussians on a straight line with distance between the adjacent centers satisfying the separation requirement. This would be easier for a clustering algorithm than some more compact placement of Gaussians.

Intuitively we seek a "spherical" placement of the component Gaussians. The simplest case would be an axis-aligned version where each Gaussian center is on an axis. Other

placements then can be achieved by rotation of the axis. The first center is always at the origin. Each of the following centers is placed at a distance D away from the origin on an axis. The ith mean vector has the form $\mu_i = [0 \ldots D \ldots 0]$, where D is at the ith position of the vector μ_i. The distance D satisfies the following

equation $D = \left\| \mu_i - \mu_0 \right\| = c \cdot \sqrt{\max(\lambda_{\max}(\Sigma_i), \lambda_{\max}(\Sigma_0)) \cdot d}$, where d is the dimension of a point. So each Gaussian is c-separated with the first Gaussian. It is easy to prove any pair of Gaussian is at least c-separated. Therefore the mixture is c-separated.

It would be interesting to explore the influences from different placement on the outcome of clustering algorithms. However we do not pursue that line of work for the time being.

3.1.2 Experimental Setup

Because we control three factors of a Gaussian cluster, the shape, volume and size of a cluster, we end up with eight cases to test. For each case, ten input files are generated. Evaluation indices are averaged over the ten files to reflect the trend. A round of files is a set of eight files, one for each of the eight cases. Since we are going to average evaluation index values over rounds, we keep all the random parameters for a round fixed, such as the mixing weights for different sized clusters and norms of covariance matrices for different volume clusters.

We investigate the quality of clustering at 5 degrees of separation. They are 0.25, 0.5, 0.75, 1.0 and 2.0. All files contain 1000 points that belong to 20 clusters. All points in a

dataset have the same dimension. This dimension can vary. However, experiments show that trends are similar for different dimensions, such as 20, 50 and 100 dimensions. Therefore we only show the results for 20-dimension points here.

To summarize, we have 5 (number of separations) x 1 (number of dimensions) x 10 (number of rounds) x 8 (number of cases) = 400 data input files.

3.1.3 Win-Lose Relationship

With the generated datasets, we are going to run algorithms on them and compare the quality of the resulting clusters. The evaluation indices we are going to use are introduced in later parts of this chapter. However, given evaluation index values for two algorithms on all the synthetic datasets, there is still a problem on how we determine which algorithm is better.

In general, we would like to aggregate the win-lose information for the algorithms from all datasets. Specifically, for two algorithms, we determine a winner for each dataset. We count the number of wins for an algorithm over all the datasets. The number is then divided by the total number of datasets and presented as the percentage of win for the particular algorithm.

Given two index values of two algorithms on a single dataset, determining who is the winner can be rather tricky. We go with the naive solution, which compares the two index values and if one is larger than the other, then the algorithm with the larger value is declared winner. If two values are the same, then it is a tie. There can be other more

complicated schemes to solve the problem. We explain two other options below. However, it is not clear there are any benefits coming from using more sophisticated solutions. Therefore, we stick with the naïve solution.

Consider the fact that for every case we investigate, ten datasets are generated. Therefore for each case, there is an average index value and a standard deviation, coming from the index values for each of the ten datasets for a given algorithm. Taken into account the standard deviation for the index values in a case, given a specific dataset, it is hard to determine how much difference between the two index values is needed to declare one algorithm wins and the other loses. For example, if the average index value for a case for both algorithms is 0.15 and the standard deviation is 0.04, one algorithm scores a 0.17 and the other scores a 0.14 on the same dataset, is the difference between the two values significant so that the first algorithm wins? Intuitively, if the difference between the two values is larger than the standard deviation, we could declare a winner. Otherwise, it would be a tie. However, the problem is that in our situation, we have two standard deviations for the two index values. It is not clear which standard deviation we should use.

The other solution tries to solve this problem by finding the overlapping range of index values for the two algorithms. If the two values fall into the overlapping region, we declare a tie. Otherwise, we declare the larger value to win. Given a data file, suppose Algorithm A has an average AveA and standard deviation SDA, and Algorithm B has an average AveB and standard deviation SDB, the overlapping region is defined as

$$\begin{cases} [Ave_B - SD_B,\ Ave_A + SD_A],\ if\ Ave_B + SD_B > Ave_A + SD_A\ and\ Ave_B - SD_B > Ave_A - SD_A \\ [Ave_A - SD_A,\ Ave_B + SD_B],\ if\ Ave_A + SD_A > Ave_B + SD_B\ and\ Ave_A - SD_A > Ave_B - SD_B \\ [Ave_B - SD_B,\ Ave_B + SD_B],\ if\ Ave_A + SD_A > Ave_B + SD_B\ and\ Ave_A - SD_A < Ave_B - SD_B \\ [Ave_A - SD_A,\ Ave_A + SD_A],\ if\ Ave_A + SD_A < Ave_B + SD_B\ and\ Ave_A - SD_A > Ave_B - SD_B \\ [],\ if\ Ave_A - SD_A > Ave_B + SD_B\ or\ Ave_A + SD_A < Ave_B - SD_B \end{cases}$$

We show the illustration for the definition in Figure 3.1. The intuition is that given a value from the overlapping region, it is hard to tell which algorithm produces that value. The overlapping region represents the set of index values that could come from both algorithms.

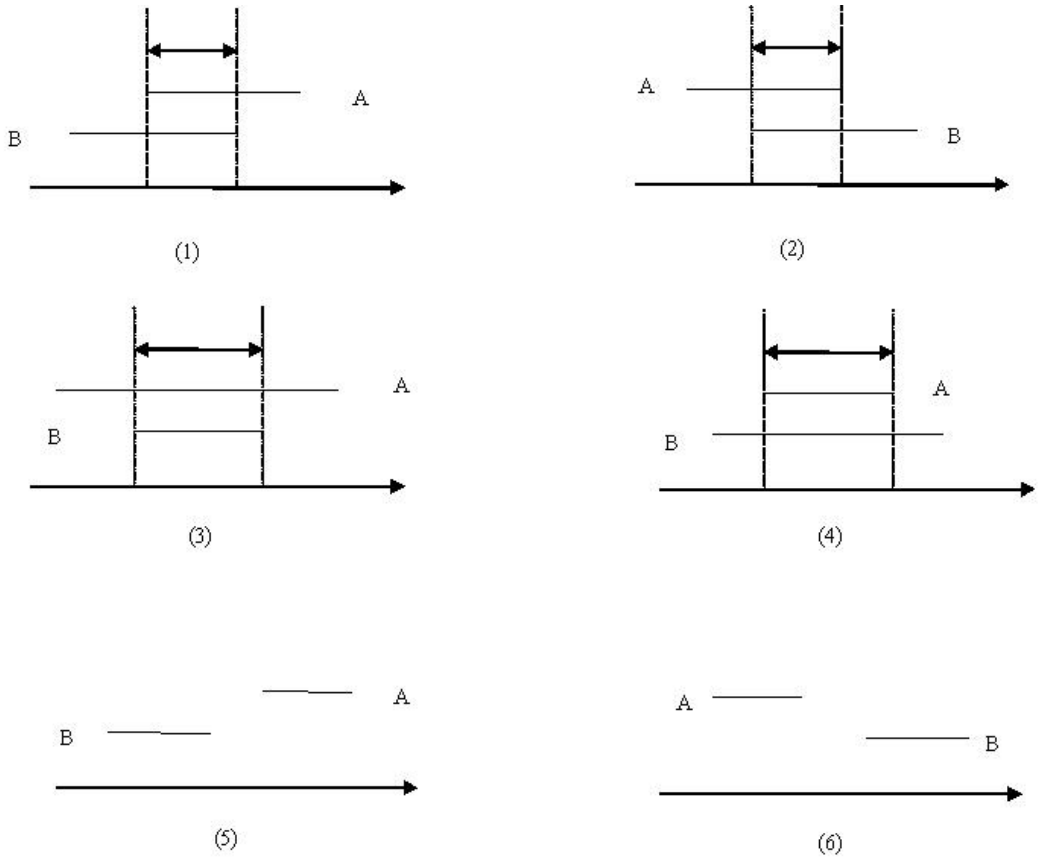


Figure 3.1 Intuitions for the Definition of Overlapping Region

The idea of using overlapping regions is not the same as declaring win only if difference between values exceeds one of the standard deviations. If one of the values falls into the overlapping region while the other is outside, we would declare a win even if two values could be very close. Another shortcoming for this idea is a lot of ties when applying to the synthetic data. Roughly there are at least 30% ties when we compare algorithms using the overlapping regions. It is not that ties are not good. In fact, the two algorithms could potentially produce similar quality of clusters. If the algorithms are true ties, we would expect any one algorithm wins and loses here and there. This would be different than the situation that one algorithm wins a small amount every single time. Even if the two algorithms are true ties, the information on when and how much it wins is still useful when we have to choose from the two.

3.2 Real Data

We download 6 document datasets from the Cluto Project [21]. These datasets are originally from the TREC collection [106]. Authors of Cluto have constructed and preprocessed these datasets. We describe their construction and preprocessing here for completeness. For all datasets, a stop-list is applied to remove common words, and all words are stemmed using Porter's suffix-stripping algorithm. Moreover, any term that occurs in fewer than two documents is eliminated.

The fbis dataset is from the Foreign Broadcast Information Service data of TREC-5, and the classes correspond to the categorization used in that collection. Datasets tr31 and

tr41 are derived from TREC-5, TREC-6, and TREC-7 collections. The classes of these datasets correspond to the documents that were judged relevant to particular queries. The datasets re0 and re1 are from Reuters-21578 text categorization test collection Distribution 1.0 [73]. For each dataset, authors of CLUTO selected documents that have a single label. Finally, the dataset wap is from the WebACE project [53]. Each document corresponds to a web page listed in the subject hierarchy of Yahoo! [2].

Table 3.1 Summary of Real Datasets

Data	Source	# of docs	# of terms	# of classes
Fbis	TREC	2463	12674	17
Tr31	TREC	927	10128	7
Tr41	TREC	878	7454	10
Re0	Reuters-21578	1504	2886	13
Re1	Reuters-21578	1657	3758	25
Wap	WebACE	1560	8460	20

We investigate the data characteristics of the real datasets in our study. For the purpose of obtaining characteristics that can be related to the generative model, we assume that a real dataset is a mixture of Gaussians, with each class corresponds to a single Gaussian distribution. Documents for each class are samples drawn from the underlying Gaussian distribution. Similar assumptions can be found in the literature of topic identification and tracking [5, 87, 93].

We calculate the separation between classes and class size distributions for these datasets. We test some of the higher dimension Gaussian properties to show anecdotal evidence that these classes conform to those properties and therefore support our choice of the Gaussian model.

3.2.1 Separation

For each pair of classes in a dataset, we calculate the centroid of both classes and define the class radius to be the distance from the centroid to the furthest point in the class. The separation between the two classes is then calculated as the ratio of the distance between the two class centroids to the maximum class radius of the two classes.

This is an approximation to the true separation because the class radius as calculated is an estimation on the true radius of the class, which is the largest eigenvalue of the covariance matrix of the underlying Gaussian distribution. The reasons are two-fold. On the one hand, there may be outliers for each class. On the other hand, given the high dimensionality of our dataset, we do not have enough data points for the class to take shape. The centroid of insufficient data may not be close to the center of the underlying distribution.

We show the minimum and maximum pair-wise class separations in Table 3.2. By our definition the separation of a clustering is the minimum separation of the pair-wise separation. However we feel it gives a better idea of the datasets by showing the other extreme of the maximum pair-wise separation. The median pair-wise separation for each dataset is also included in Table 3.2.

Table 3.2 Separations for Real Datasets

	Min Separation	Max Separation	Median Separation
Re0	0.2373	0.7689	0.5543
Re1	0.2783	0.7598	0.5370
Tr31	0.3246	0.8053	0.4574
Tr41	0.2871	0.7039	0.5312
Wap	0.2147	0.6524	0.4031
Fbis	0.2666	0.6901	0.4894

3.2.2 Class Sizes

In Table 3.3 we show the sizes of each class in a dataset. Note all datasets contain classes with different sizes.

Table 3.3 Class Sizes for Real Datasets

	Re0	Re1	Tr31	Tr41	Wap	Fbis
# of classes	13	25	7	10	20	17
# of docs	1504	1657	927	878	1560	2463
Class 0	608	371	352	243	341	506
Class 1	319	330	227	174	196	387
Class 2	219	137	151	162	168	358
Class 3	80	106	111	95	130	190
Class 4	60	99	63	83	97	139
Class 5	42	87	21	35	91	125
Class 6	39	60	2	33	91	121
Class 7	38	50		26	76	119
Class 8	37	48		18	65	94
Class 9	20	42		9	54	92
Class 10	16	37			44	65
Class 11	15	32			40	48
Class 12	11	31			37	46
Class 13		31			35	46
Class 14		27			33	46
Class 15		20			18	43
Class 16		20			15	38
Class 17		19			13	

Class 18		19			11	
Class 19		18			5	
Class 20		18				
Class 21		17				
Class 22		65				
Class 23		13				
Class 24		10				

Properties of High Dimension Gaussians

The expected squared distance from a point to the center, drawn from a Gaussian distribution N(μ,σI),is $\sigma^2 d$, where d is the dimension of a point [35]. According to the law of large numbers, with high probability, the distances of a large number of points to the center fall tightly around this expectation. This implies the mass of a high-dimension Gaussian concentrates within a thin shell around its radius. This is a property very easy to test. We draw a sphere with a certain radius around the center of a class and count how many points fall into this sphere.

In Table 3.4 we give the percentage of points within each class to be inside a sphere with a radius three quarters of the maximum radius of the class and with the same center.

Table 3.4 Percentage of Points within Sphere with Three Quarters of the Max Radius

	Re0	Re1	Tr31	Tr41	Wap	Fbis
Class 0	0	0	0	0	0	0
Class 1	0	0	0	0	0	0.0749
Class 2	0	0	0.0794	0	0	0
Class 3	0	0	0	0	0	0
Class 4	0.0333	0.0101	0	0	0	0
Class 5	0	0	0	0	0	0
Class 6	0.1538	0	0	0	0	0
Class 7	0.0789	0	0	0.0769	0	0.4873
Class 8	0.0810	0.4375		0.0555	0	0.0106

Class 9	0.45	0		0.2222	0	0
Class 10	0.0625	0			0	0
Class 11	0.5333	0.25			0.05	0
Class 12	0.0909	0			0	0
Class 13		0			0	0
Class 14		0			0	0.1739
Class 15		0.05			0	0
Class 16		0			0	0
Class 17		0			0	
Class 18		0			0	
Class 19		0			0	
Class 20		0				
Class 21		0.1176				
Class 22		0				
Class 23		0.0769				
Class 24		0				

In Table 3.5 we show the percentage of points for each class that falls into 90% of maximum class radius.

Table 3.5 Percentage of Points within Sphere with nine-tenths Maximum Radius

	Re0	Re1	Tr31	Tr41	Wap	Fbis
Class 0	0.1217	0.0053	0.0085	0	0	0.4347
Class 1	0.0062	0.0090	0.0792	0.1609	0	0.7519
Class 2	0.0319	0	0.3178	0.0555	0.0654	0.0083
Class 3	0.075	0	0.2612	0.1473	0.0538	0.4578
Class 4	0.7166	0.7575	0.3809	0.4819	0	0.7266
Class 5	0.3809	0.5862	0.3333	0.6	0	0.384
Class 6	0.4545	0	0	0.6969	0	0.6280
Class 7	0.7894	0		0.8076	0.1578	0.8655
Class 8	0.6216	0.875		0.7222	0.4615	0.6063
Class 9	0.75	0.1666		0.5555	0.3418	0.4565
Class 10	0.5625	0.3783			0.4153	0.4153
Class 11	0.9333	0.25			0.675	0.5208
Class 12	0.4545	0.4193			0.0810	0.2608
Class 13		0.4516			0	0.1086
Class 14		0			0.0606	0.7173
Class 15		0.45			0.0555	0.3023
Class 16		0.45			0	0.1578

Class 17		0.4210			0.3076	
Class 18		0			0.3636	
Class 19		0			0.2	
Class 20		0				
Class 21		0.7647				
Class 22		0				
Class 23		0.0769				
Class 24		0				

In Table 3.4 there are three classes, in three datasets respectively, at least 40% of whose data points fall into the test sphere. Meanwhile in Table 3.5 37 classes, out of 92 classes, have at least 40% of their data points falling into the test sphere. From Table 3.4 and Table 3.5 we can see the clouds that constitute the classes in real datasets are indeed almost hollow in the center, because even for a sphere with three-quarters of the maximum radius there are almost no points inside it. On the contrary, significant portions of the points are included, when the radius of the test sphere increases to ninety percent of the maximum radius.

3.3 Evaluation Indices

We focus on external evaluation indices in our work. In this section, we describe three evaluation indices from different background and analyze their behaviors on random cluster labels and algorithm rankings. Of the three evaluation indices we recommend the use of a single index to present the experimental results.

3.3.1 Contingency Tables

All external indices in the literature are derived from the contingency table. A contingency table H is a $n \times n$ matrix, with entry $h(c,k)$ indicating the number of objects belonging to cluster k with class label c. If the rows or columns of the contingency table are summed together, they are called the marginal tables, that is, $\left\{h(k) = \sum_c h(c,k)\right\}$ and $\left\{h(c) = \sum_k h(c,k)\right\}$. A further reduction is represented by the 2×2 contingency table $A = \left\{a_{ij} \mid i, j \in \{0,1\}\right\}$. The meanings of the components of this table are:

a_{00} -- number of pairs of objects in the same class and same cluster

a_{01} -- number of pairs of objects in the same class but not the same cluster

a_{10} -- number of pairs of objects in the same cluster but not the same class

a_{11} -- number of pairs of objects not in the same cluster and not in the same class

3.3.2 Hubert's Γ

Hubert's Γ is chosen in our study because it represents a family of evaluation indices in the traditional cluster analysis literature [58]. All indices of this family are based on the 2×2 contingency table. We give a brief description of the family in this subsection. The definitions for these indices are given in Table 3.6.

The Rand index is most popular in the traditional literature [34, 58]. It counts the number of pairs of objects that are placed correctly, which includes both a_{00} and a_{11}, and then normalizes the count by the total number of pairs. However this index does not scale in terms of the number of clusters because a_{11} increases as the number of clusters increases, causing the index to approach 1.

There are many indices that modify the Rand index, for example, the corrected Rand [58].

$$\text{Corrected Rand} = \frac{\text{Rand - E(Rand)}}{\text{Max(Rand)-E(Rand)}}$$

The corrected Rand normalizes the Rand index by subtracting the expected value of the Rand index under a particular null distribution, and then dividing by the maximum possible Rand index minus the expected value. The resulting index ranges from zero to one. However, it is hard to agree on which null distribution to use. For example, there is the random label hypothesis, which assumes all permutations of the labels are equally likely. Some people use the assumption of fixing the sum of the rows and columns in a contingency table and selecting the partition randomly, because it is simpler and easier to calculate than the more general random label hypothesis. Even if a null distribution is picked, the calculation of the expected value is error-prone. In fact the corrected Rand takes more than one form because people calculate their expectations differently.

Other external indices include the Fowlkes and Mallows, Jaccard and the Γ statistic [58]. They try to normalize the entry a_{00} differently, therefore do not have the problem of approaching 1. The Fowlkes and Mallows, the Γ statistic and the Rand index are linear

functions of each other. Please see Table 3.6 for details. Of them we choose the Γ statistic as the representative of the family because of its more intuitive mathematical interpretation.

Let $\{X(i,j)\}$ and $\{Y(i,j)\}$ be two n × n matrices. The raw Hubert's Γ statistic is the point correlation between the two matrices.

$$\Gamma_{raw} = \sum_i \sum_j X(i,j)Y(i,j)$$

If mx and my denote the means and sx and sy denote the standard deviation of the entries of matrices X and Y, the normalized Γ statistic is

$$\Gamma = \left\{(1/M)\sum_i \sum_j \left[X(i,j) - m_X\right]\left[Y(i,j) - m_y\right]\right\} / s_X s_y$$

where $M = n^2$ is the number of entries in the double sum and the moments are given by

$$m_X = (1/M)\sum_i \sum_j X(i,j), \; {s_X}^2 = (1/M)\sum_i \sum_j \left[X(i,j) - m_X\right]^2$$

m_y and s_y take similar form.

If both s_x and s_y are zero, the normalized Γ statistic is 1 because that indicates both matrices are constant. However, if one of the matrices is constant, the normalized Γ statistic is undefined.

The Γ statistic measures the degree of linear correspondence between the entries of X and Y. The normalized Γ is always between -1 and 1. Larger value of Γ suggests that the two matrices agree with each other more.

When applying to measure the fit between cluster labels and class labels, X and Y are defined as below. Note they are symmetric matrices now.

$$X(i,j) = \begin{cases} 1, & \text{if objects } i \text{ and } j \text{ have the same class label} \\ 0, & \text{otherwise} \end{cases}$$

$$Y(i,j) = \begin{cases} 1, & \text{if object } i \text{ and } j \text{ belong to the same cluster} \\ 0, & \text{otherwise} \end{cases}$$

We can see the raw Γ under this case is the number of pairs that belong to both the same class and the same cluster. In the four-entry contingency table A, it is the entry a00. In the formula to calculate Γ statistic, if we write the moments in another way and take into account the symmetry of the matrices,

$$m_x = m_1 / M,\ m_y = m_2 / M$$
$$m_1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X(i,j),\ m_2 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Y(i,j)$$

then

$$m_1 = a_{00} + a_{01},\ m_2 = a_{00} + a_{10}$$
$$M = n(n-1)/2 = a_{00} + a_{01} + a_{10} + a_{11}$$

These short hands are used in Table 3.6 to define the aforementioned external evaluation indices. Together with the definitions, we also show that the Rand, Fowlkes and Mallows and Γstatistic are linear functions of each other. Specifically, we define

$$Z = \sum_c \sum_k h(c,k)^2$$

Then the entries in the 2x2 contingency table can be expressed in terms of Z.

$$a_{00} = \sum_c \sum_k \binom{h(c,k)}{2} = (1/2)\sum_c \sum_k h(c,k)^2 - (n/2) = (1/2)Z - (n/2)$$

$$a_{01} = \sum_k \binom{h(k)}{2} - \sum_c \sum_k \binom{h(c,k)}{2} = (1/2)\sum_k h(k)^2 - (1/2)\sum_c \sum_k h(c,k)^2 = (1/2)\sum_k h(k)^2 - (1/2)Z$$

$$a_{10} = \sum_c \binom{h(c)}{2} - \sum_c \sum_k \binom{h(c,k)}{2} = (1/2)\sum_c h(c)^2 - (1/2)\sum_c \sum_k h(c,k)^2 = (1/2)\sum_c h(c)^2 - (1/2)Z$$

$$a_{11} = \binom{n}{2} - a_{00} - a_{01} - a_{10} = \frac{n(n+1)}{2} - \frac{1}{2}\left[\sum_c h(c)^2 + \sum_k h(k)^2\right]$$

Table 3.6 Definitions of Traditional External Evaluation Indices

Name	Formula
Rand	$\frac{a_{00}+a_{11}}{\binom{n}{2}} = 1 + \left[Z - (1/2)\left(\sum_c h(c)^2 + \sum_k h(k)^2\right)\right] \Big/ \binom{n}{2}$
Jaccard	$\frac{a_{00}}{a_{00}+a_{01}+a_{10}} = \left(Z-n\right) \Big/ \left(\sum_c h(c)^2 + \sum_k h(k)^2 - Z - n\right)$

Fowlkes and Mallows	$\frac{a_{00}}{\sqrt{m_1 \cdot m_2}} = (1/2)(Z-n) \Big/ \left[\sum_c \binom{h(c)}{2} \sum_k \binom{h(k)}{2} \right]^{1/2}$
Γstatistic	$\frac{Ma_{00} - m_1 \cdot m_2}{\sqrt{m_1 \cdot m_2 (M-m_1)(M-m_2)}} = \frac{n(n-1)(Z-n) - \left(\sum_k h(k)^2 - n \right) \left(\sum_c h(c)^2 - n \right)}{\sqrt{\left(\sum_k h(k)^2 - n \right) \left(\sum_c h(c)^2 - n \right) \left(n^2 - \sum_k h(k)^2 \right) \left(n^2 - \sum_c h(c)^2 \right)}}$

3.3.3 F-scores

The f-score measurement is more popular than the contingency-table based indices in recent classification and clustering literature [77, 89, 100, 119]. It is devised to combine two measures, precision and recall, into a single effectiveness measure when comparing the output of an algorithm to a priori structure. We choose it as a candidate because of its popularity.

F-score can be traced to the early days of information retrieval when the effectiveness of a retrieval system was measured by precision P and recall R [89]. The effectiveness measure E is

$$E = \frac{(1+\beta^2) \cdot PR}{\beta^2 P + R}$$

where β is a user-supplied parameter to indicate his/her preference between precision and recall. For the f-score, β=1.

In the situation we consider precision to be the 'purity' of a cluster and recall to be the 'encompassment' of a cluster. Specifically for any class c and any cluster k,

$$f(c,k) = \frac{2 \cdot \frac{h(c,k)}{h(k)} \cdot \frac{h(c,k)}{h(c)}}{\frac{h(c,k)}{h(k)} + \frac{h(c,k)}{h(c)}}$$

This version of f-score attaches equal importance to the 'purity' and 'encompassment' of a cluster. It is easy to imagine other versions where a user can specify which aspect of a cluster is more important. F-score of a class is defined to be

$$f(c) = \max_k f(c,k)$$

F-score for a clustering of k clusters is defined to be

$$f(solution) = \sum_c \frac{h(c)}{n} \cdot f(c)$$

where n is the total number of objects to be clustered.

3.3.4 Q_0

The third index we choose, Q_0, is a fairly new derivation by Byron Dom [32]. We have not seen any systematic application of the index yet. Basically it measures the conditional entropy between the cluster labels and the class labels. Put in another way, it determines how useful (in bits) the cluster labels are in encoding the class labels. From information theory, the expected per-symbol code length for the class labels given that

we know the associated cluster labels and their joint distribution $p(c,k)$ is given by the conditional entropy:

$$H(C \mid K) = -\sum_{c=1}^{|C|} \sum_{k=1}^{|K|} p(c,k) \log p(c \mid k)$$

The empirical conditional entropy estimates $H(C \mid K)$ using the contingency table H.

$$\tilde{H}(C \mid K) = -\sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{h(c,k)}{n} \log \frac{h(c,k)}{h(k)}$$

Conditional entropy has been used to measure the degree of association between variables and there are formulations using the idea to measure the distance between classes and clusters [7, 79, 99, 113]. However Dom argues in his paper that it is incomplete in the sense that to realize the code length given by the empirical conditional entropy the decoder must know H. Therefore we have to have a scheme for encoding H, and the code length for H should be added to the empirical conditional entropy.

The quality measure is the entire encoding cost per object as below.

$$Q(C,K) = \tilde{H}(C \mid K) + \frac{1}{n} \sum_{k=1}^{|K|} \log \binom{h(k) + |C| - 1}{|C| - 1}$$

The normalized form is calculated by the following equation.

$$Q_0(C,K) = \frac{\max_K \left[Q(C,K) \right] - Q(C,K)}{\max_K \left[Q(C,K) \right] - \min_K \left[Q(C,K) \right]}$$

The minimum value of Q(C, K) occurs when the clustering recovers exactly the original structure.

$$\min_K [Q(C,K)] = \frac{1}{n}\sum_{c=1}^{|C|} \log \binom{h(c)+|C|-1}{|C|-1}$$

However $\max_K [Q(C,K)]$ depends on details of the contingency table H. A tight upper bound is given below,

$$\max_K [Q(C,K)] = \tilde{H}(C) + \log|C|$$

where $\tilde{H}(C)$ is the marginal contingency table.

In summary, the measure Q_0 we calculate is as follows.

$$Q_0(C,K) = \frac{\tilde{H}(C) + \log|C| - Q(C,K)}{\tilde{H}(C) + \log|C| - \frac{1}{n}\sum_{c=1}^{|C|} \log \binom{h(c)+|C|-1}{|C|-1}}$$

3.3.5 Comparison of Evaluation Indices

In this subsection we would like to investigate the properties of these evaluation indices, such as what their expected value are given random cluster labels. We would also like to know if these indices give different rankings of algorithms. If they give similar rankings then using one of them to evaluate is good enough. Otherwise if they give drastically different rankings, it is evident that using a single index to evaluate is not appropriate. In such situations, it is better to have multiple indices to vote for the best algorithm.

3.3.5.1 Index Values under Random Cluster Labels

Given fixed class labels and the number of clusters desired, we generate random cluster labels for each point. We want to know the expected value for each measure under these random cluster labels. A clustering algorithm should result in evaluation values larger than those of the random cluster labels to do a good job.

The algorithm to calculate expected index values on random clusterings are as follows.

Fix class labels for each object.

Random permute the n objects.

Generate k-1 random numbers to chop the n objects into k clusters.

Calculate the evaluation index on the clustering.

We tried on two families of class sizes. In one family all classes have equal sizes. In the other family each class has arbitrary sizes.

We generate ten sets of class labels for each family. For each set of class labels, one hundred sets of random cluster labels are generated and index values are calculated. Therefore we collect one thousand index values for each measure for each family. The statistics of these are listed in Table 3.7.

Table 3.7 Expected Values of Indices for Random Clusters

	Same size class	Different size class
F-score	0.1158±0.0038	0.1213±0.0042
Hubert's Γ	0.0008±0.0013	0.0004±0.0014
Q_0	0.4766±0.0040	0.4857±0.0043

Note the Γmeasure shows signs of converging to zero. The reason for its large standard deviation is that its value range for random cluster labels is drastic. While many values are around 0.001, others may go as low as 0.0001, therefore making the standard deviation very large. The Q_0 measure roughly converges to 0.5 for the random cluster labels. While being an intuitive number for random cases, the value limits the usable range of the measure. In fact, the measure does not have as large a range of values to distinguish between good and better clustering.

3.3.5.2 Disagreement on Algorithm Ranking

One interesting topic is to see if the algorithm rankings induced by these measures differ. We study the difference in ranking for two of our algorithms: k-means and min-max cut with greedy. For the time being we only care for the ranking between algorithms and therefore spare the details of these algorithms, which appear in the following chapter.

For each generated dataset, we count the ranking of the two algorithms by all three measures: f-score, Hubert's Γ and Q_0. If the ranking is the same then we say the three measures agree. If the ranking is not the same, the measures disagree. One of the measures is the cause for this disagreement. We break down the disagreement by separation of data sets in Table 3.8.

Table 3.8 Disagreement in Algorithm Rankings (Two Algorithms, Three Indices)

Separation	0.25	0.5	0.75	1.0	2.0	3.0	total
Total number of data files	80	80	80	80	80	80	480
Total disagreement	29%	9%	7%	4%	8%	25%	14%
F disagree	16%	5%	6%	1%	4%	24%	9%
Γ disagree	4%	1%	1%	3%	0%	0%	2%
Q_0 disagree	9%	3%	0%	0%	4%	1%	3%

F-score is the measure that causes disagreement most. It disagrees with the other two measures on at least 25% of the input files when the separation of the file is either very small (0.25) or very large (3.0). Note the clustering is of poor quality when the separation is 0.25 and the resulting clusters are of relatively high quality when the separation is high. The actual differences of cluster quality for theses cases are relatively small and therefore cause the ranking of them to be more susceptible to variations.

3.3.5.3 Distribution of Measure Values

Intuitively we want a measure that responds to the change of quality of clusters linearly, given a definition of cluster quality. In Figure 3.2, we give different curves to illustrate our point. In these figures, the x-axis represents the quality of clusters, with zero indicating the worst quality and one the best quality. The y-axis denotes the value of a measure, ranging from zero to one. In the figure on the left, the measure value makes a steep jump when the cluster quality is close to perfect. Therefore only nearly perfect clusters would be considered good clusters while some clusters with middle range quality would end up indistinguishable from the ones with poor quality. In the middle

figure, a similar situation happens, except that this time the clusters with middle range quality are mingled with clusters with good quality.

Figure 3.2 Meausre Distribution Illustration

Ideally we like to have a measure with a value distribution similar to the one on the right in Figure 3.2. The distribution has the property of responsiveness. The measure responds to the change of cluster quality. Clusters with different degree of quality have different measure values. It also has the property of stability. A small change in the cluster quality results in a small change in the measure value.

Given the three external indices we study, we are able to plot in three dimensions the distribution of measure values for two of them: f-score and Hubert's Γ. The Q_0 measure has too complicate a form to plot in three dimensions.

We plot the f-score distribution using the precision P and the recall R as the x and y axis. The z-axis represents the f-score value. Using precision and recall to rewrite the formula of f-score, we get

$$f = \frac{2P \cdot R}{P + R}$$

Note both precision and recall values range from zero to one. So does the f-score. The plot is in Figure 3.3. We observe the distribution of f-score in the space of precision and recall is rather smooth and satisfies the properties of both responsiveness and stability.

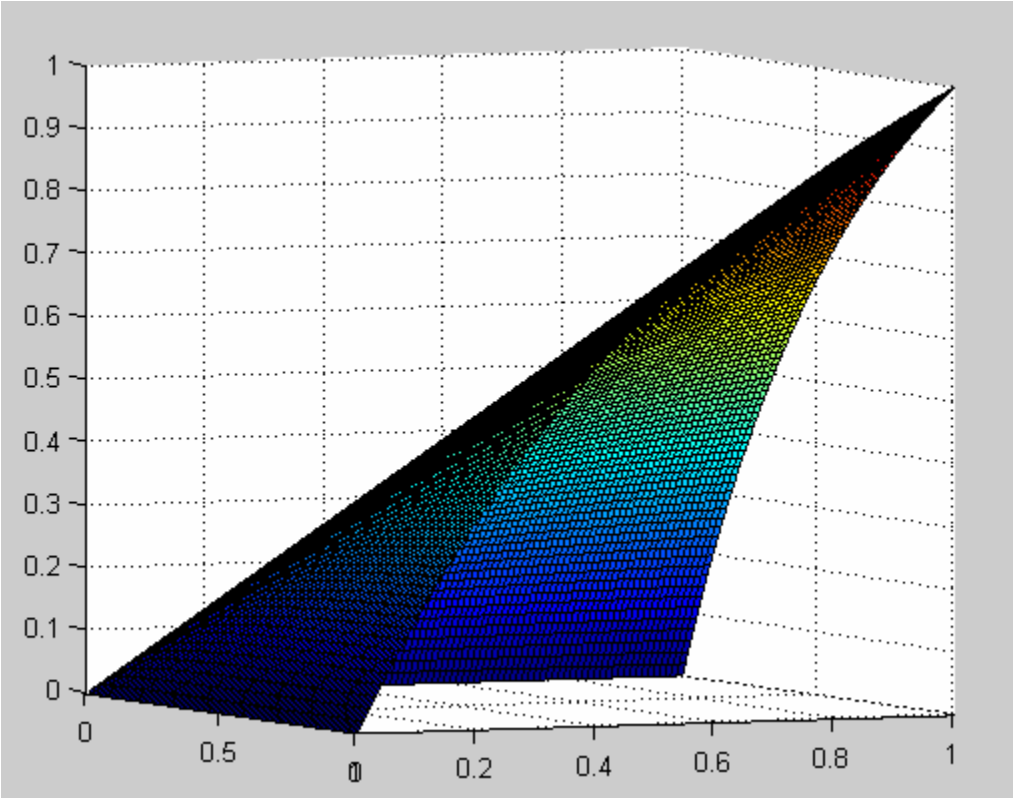


Figure 3.3 Distribution of f-score values

The measure Hubert's Γ is based on the four entries of a 2x2 contingency table. To plot it out in three dimension, we have to fix some entries so that to observe the distribution.

The formula of Hubert's Γ is

$$\frac{Ma_{00} - m_1 \cdot m_2}{\sqrt{m_1 \cdot m_2 (M - m_1)(M - m_2)}}$$

$$m_1 = a_{00} + a_{01},\ m_2 = a_{00} + a_{10}$$

In particular, we fix the total number of pairs (M) and let a01 and a10 be identical. Therefore a00, the number of pairs that are in the same class and same cluster, and a01, the number of pairs that are in the same class but not the same cluster, are the two free variables. We think this scheme allows the degree of matching between classes and clusters to change and therefore depicts a reasonable snapshot of the distribution of Hubert's Γ.

Fix M at 200, we have the plot in Figure 3.4. The black flat surface to the right is the zeros since not all parts of the grid is reachable by the 2x2 contingency table.

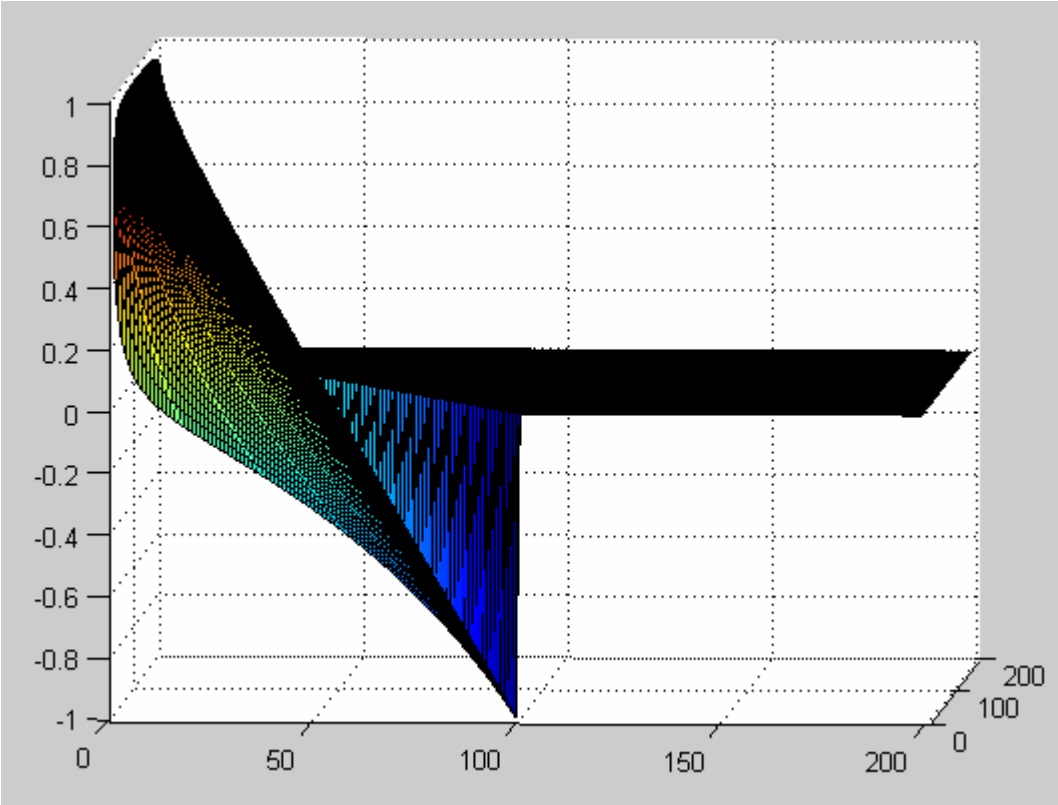


Figure 3.4 Distribution of Hubert's Γ, M=200

In Figure 3.5, we show the same plot but in a different angle. In summary, the distribution of Hubert's Γ usually satisfies the properties of responsiveness and stability.

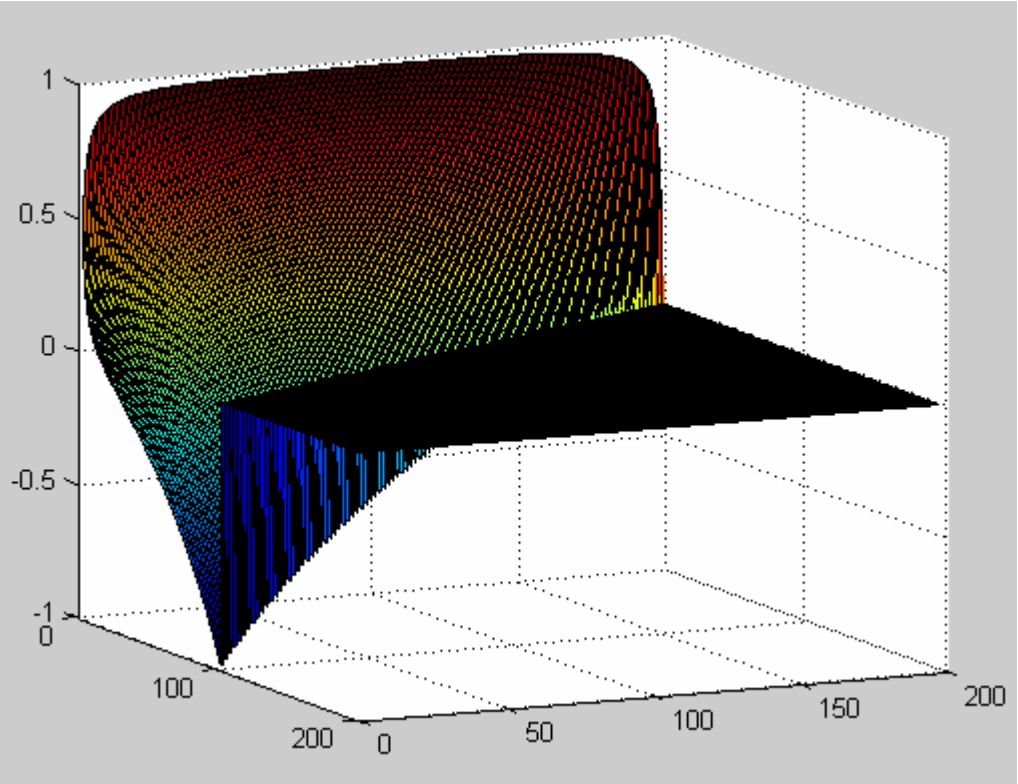


Figure 3.5 Distribution of Hubert's Γ, M=200

3.3.5.4 Conclusion

From the above analysis of the three evaluation indices, we observe the disagreement on algorithm rankings happens for very small and very large separations most of the time. In our experiments we focus on the comparison among algorithms and analyze algorithmic behaviors for a selected range of separation values. We start from very small separation, such as 0.25, to show the starting point of the algorithms on very difficult datasets. The main analysis focuses on the middle range of separation values, such as 0.5, 0.75 and 1.0. Therefore we feel it appropriate to use a single index to evaluate the algorithms under investigation. Of the three evaluation indices we choose

to present all our experimental results in f-score. The reasons are two fold. On the one hand the other two measures have some peculiar properties that may affect the evaluation. For example, the Γ measure is converging to zero for random cases with large variations, suggesting large dynamic range for harder cases. Both the Γ and Q_0 measure have a limited usable range, which is roughly half of the full range, because under random cases they converge to the middle point of the full range. On the other hand the f-score measure is the most popular choice among the current practitioners. Moreover, the distribution of the f-score measure satisfies our intuition of the properties of a good measure: responsiveness and stability. Therefore in this thesis we present our experimental study of the comparison of clustering algorithms using a single evaluation index: the f-score.

Chapter 4 Objective Functions and Optimization Approaches

In this chapter we investigate the behaviors of objective functions under different optimization approaches. We describe the objective functions and optimization approaches in our study. Two objectives, sum-of-squares and min-max cut, are chosen because of their claimed behavioral discrepancy in response to cluster shapes. Greedy is the baseline search strategy chosen because of its popularity. Kernighan-Lin is studied because it searches more exhaustively than greedy and runs in reasonable time. These objective functions and optimization approaches combine into four different algorithms. We run these algorithms on both synthetic and real data sets to compare their performance in terms of both efficiency and effectiveness.

We show that the objective sum-of-squares performs well under greedy, and Kernighan-Lin can improve the quality of clusters produced by sum-of-squares significantly. However the quality of clusters produced by min-max cut does not improve under Kernighan-Lin comparing with under greedy. As a result the winning margin of min-max cut over sum-of-squares in the different-sized clusters disappears when upgrading from greedy to Kernighan-Lin. We speculate the reason is that greedy can do such a

good job of optimizing the objective of min-max cut that there is not much room for Kernighan-Lin to improve.

We conclude that the algorithm with the best quality of clusters out of the four is sum-of-squares with Kernighan-Lin, which is also the longest-running one. In contrast, the algorithm of k-means can produce clusters of 95% of the best quality achievable among the four algorithms in running time an order of magnitude faster comparing with the algorithm with the best quality. The findings are contrary to the belief that k-means sacrifices quality for speed.

4.1 K-means

There are two types of clustering algorithms: hierarchical and partitional [58]. The hierarchical methods build a tree of nested partitions either top-down or bottom-up. Partitional clustering algorithms seek a flat partition of data to optimize a certain criterion, which we call an objective function. People have long argued in the literature that partitional algorithms are sacrificing quality of clusters compared with the hierarchical algorithms [58]. However, hierarchical methods cannot be widely applied to large data sets because of their quadratic time complexity. Such data sets are ubiquitous since the Web age.

The partitional algorithm we focus on is the k-means algorithm. K-means is arguably the most popular partitional algorithm because of its linear time complexity and ease of implementation [78]. New algorithms have been proposed for finding high-quality flat

clusterings in linear time [4, 15, 49, 52, 55, 82, 92, 98, 116], but they are more difficult to implement and have not become widely used. Therefore we investigate methods to improve the quality of clusters produced by the basic k-means algorithm.

4.2 Objective Functions

K-means uses the objective of minimizing the sum of squared distance to centers, which is referred to as sum-of-squares in the literature. Sum-of-squares has long suffered criticism that it favors spherical clusters [51, 58]. There are other objective functions in the literature, such as max-diameter and average pair-wise distance, which more or less draw the same criticism [58]. Recently many clustering algorithms using minimum-cut based objective functions have been proposed; they show moderate sign of success in terms of quality [30, 64, 95, 96]. The minimum-cut based objective does not show any preference in terms of the shape of a cluster, and therefore we choose it to compare with the sum-of-squares objective function. The work of Zhao and Karypis supports our choice of sum-of-squares and min-max cut as competitors [119]. In their study they show that sum-of-squares wins by small margins over min-max cut when used as a subroutine in a bisecting hierarchical technique.

4.2.1 Sum-of-Squares

The sum-of-squares, which is used as the objective function for both K-means and Ward's method [37], is chosen because algorithms with this objective have been

reported in the literature to yield the best quality clusters [12, 70, 80, 119]. Given a set C of data points, represented by d-dimensional vectors $\vec{p}_i,\ i \in [0,n)$, the sum of square criterion is defined as

$$\sum_{i=0}^{n-1} dist(\vec{p}_i, \vec{c})^2, \quad \vec{c} = \frac{\sum_{i=0}^{n-1} \vec{p}_i}{n}.$$

Distance between two vectors is the Euclidean distance. Note that the center is not necessarily among the given input points. The objective aggregates the distance from each point to the cluster center.

When there are k clusters, the objective to minimize is

$$\sum_{j=1}^{k} \sum_{i \in C_j} dist(\vec{p}_i, \vec{c}_j), \quad \vec{c}_j = \frac{\sum_{i \in C_j} \vec{p}_i}{|C_j|}.$$

We seek a partition of the input points such that the sum of the distances between every point and its cluster center is minimized.

4.2.2 Min-max Cut

The second objective chosen is from the family of the minimum-cut based objectives because they are believed not to favor a specific shape. The family includes normalized cut [96], conductance [63] and min-max cut [30]. In our preliminary experiments both normalized cut and conductance perform relatively worse than min-max cut in terms of the quality of the resulting clusters. We present experimental evidence in the next

section. Based on the empirical results, we choose the min-max cut as the best representative of this family.

To define the objectives of the cut-based family, we treat the input data points as a graph G=(V, E), where V is the set of all input points $\vec{p}_i$, and E is the set of edges connecting each pair of points. Each edge carries a weight which indicates the similarity between the two points. The closer in terms of Euclidean distance, the more similar they are. Weight $w(a,b)$ is calculated as $1-\frac{dist(p_a,p_b)}{\max dist(p_i,p_j)}$. A cut divides the node set V into two subsets: A and B.

The min-max cut is defined as follows.

$$\text{min-max cut (A, B)} = \frac{\sum_{i\in A, j\in B} w(i,j)}{\sum_{i,j\in A} w(i,j)} + \frac{\sum_{i\in A, j\in B} w(i,j)}{\sum_{i,j\in B} w(i,j)}$$

In the case when there are more than 2 clusters, the min-max cut objective we minimize is

$$\text{min-max cut } (\{C_t\text{: t=1...k}\}) = \sum_t \frac{\sum_{i\in C_t, j\in V-C_t} w(i,j)}{\sum_{i,j\in C_t} w(i,j)}$$

The normalized cut is similar to the min-max cut, except in the denominator, the sum of weights from nodes inside the cluster to the outside world substitute the sum of weights for pairs of nodes inside a single cluster.

$$\text{normalized cut (A, B)} = \frac{\sum_{i \in A, j \in B} w(i, j)}{\sum_{i \in A, j \in V} w(i, j)} + \frac{\sum_{i \in A, j \in B} w(i, j)}{\sum_{i \in B, j \in V} w(i, j)}$$

Instead of a two-term definition, the conductance uses the minimum of the two denominator terms in the definition of normalized cut.

$$\text{conductance (A, B)} = \frac{\sum_{i \in A, j \in B} w(i, j)}{\min(\sum_{i \in A, j \in V} w(i, j), \sum_{i \in B, j \in V} w(i, j))}$$

The definitions of both conductance and normalized cut when the number of clusters exceeds two follow the same format of the definition for more-than-two-cluster min-max cut.

4.2.2.1 Comparisons between Cut-based Objectives

In this section we present the experimental evidence that min-max cut is the best objective of the cut-based objective family. We use two optimization approaches to optimize the two objectives: greedy and Kernighan-Lin. The details of these optimization approaches can be found in the next section. For the time being we would like to know which objective, normalized cut or min-max cut, gives the best quality of clusters given an optimization approach.

In Table 4.1 and Table 4.2, we present the average f-scores for the two algorithms: min-max cut with greedy and normalized cut with greedy, using the synthetic datasets. We observe the algorithm min-max cut with greedy wins over normalized cut with greedy. In Table 4.3 and Table 4.4, the comparison is between the two algorithms: min-max cut

with Kernighan-Lin and normalized cut with Kernighan-Lin. Although the Kernighan-Lin approach helps normalized cut to improve the quality of clusters in the non-spherical category, the algorithm min-max cut with Kernighan-Lin still wins over normalized cut with Kernighan-Lin most of the time. Therefore we choose min-max cut as the representative of the cut-based objective family.

Table 4.1 Average F-scores for Min-max Cut with Grd and Normalized Cut with Grd

Greedy				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Mcut	0.15±0.01	0.39±0.04	0.86±0.01	0.97±0.00	0.99±0.00
			ncut	0.10±0.00	0.10±0.00	0.10±0.00	0.10±0.00	0.99±0.01
		Diff Size	Mcut	0.16±0.01	0.42±0.03	0.69±0.05	0.73±0.05	0.79±0.04
			ncut	0.12±0.01	0.12±0.01	0.12±0.01	0.13±0.01	1.00±0.00
	Diff Volume	Same Size	Mcut	0.22±0.05	0.63±0.12	0.91±0.02	0.98±0.01	0.99±0.00
			ncut	0.10±0.00	0.10±0.00	0.10±0.00	0.10±0.00	0.54±0.47
		Diff Size	Mcut	0.24±0.05	0.57±0.07	0.72±0.05	0.74±0.05	0.78±0.05
			ncut	0.12±0.01	0.12±0.01	0.12±0.01	0.12±0.01	0.57±0.46
Non-sphere	Same Volume	Same Size	Mcut	0.73±0.04	0.99±0.00	0.99±0.01	0.96±0.02	0.91±0.03
			ncut	0.10±0.00	0.10±0.00	0.98±0.02	0.97±0.03	0.96±0.03
		Diff Size	Mcut	0.68±0.04	0.76±0.05	0.78±0.04	0.79±0.04	0.79±0.06
			ncut	0.12±0.01	0.12±0.01	0.81±0.36	1.00±0.01	0.99±0.01
	Diff Volume	Same Size	Mcut	0.84±0.06	0.99±0.00	0.98±0.02	0.91±0.03	0.88±0.04
			ncut	0.10±0.00	0.10±0.00	0.53±0.46	0.94±0.09	0.95±0.03
		Diff Size	Mcut	0.70±0.05	0.76±0.04	0.79±0.04	0.78±0.05	0.81±0.06
			ncut	0.12±0.01	0.12±0.01	0.78±0.36	0.92±0.23	0.99±0.02

Table 4.2 Percentage of Win for Min-max Cut with Grd and Normalized Cut with Grd

	Min-max Cut	Normalized Cut
Spherical	87%	13%
Non-spherical	52%	48%
Same volume	66%	34%
Different volume	72%	28%
Same size	73%	27%
Different size	66%	34%

From Table 4.1, we observe the f-scores for normalized cut exhibit two trends. First, they are extremely low even when the separation increases to 1.0 for the spherical datasets. Secondly, when the f-scores do increase, they have large standard deviation, like those numbers for non-spherical datasets with separation 0.75. The reason is that the normalized cut objective value does not match with class structure. Specifically, the final objective values for normalized cut keep dropping when the separation increases. However, the lower objective values do not necessarily lead to higher f-scores. In fact, a small difference in objective values can correspond to drastically different f-scores. That accounts for the large standard deviation in higher separation cases because even when the ending objective values for ten rounds are similar, they result in f-scores at two extremes. We do not consider this is a feature of the normalized cut. Therefore, even as it can outperform min-max cut for separation 2.0 and above cases, the min-max cut has a consistent performance, especially for the middle range of separations, which are the most encountered situations in real-world datasets.

Table 4.3 Average F-scores for Min-max Cut and Normalized Cut with Kernighan-Lin

KL				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Mcut	0.15±0.01	0.40±0.03	0.85±0.01	0.97±0.00	0.99±0.00
			ncut	0.10±0.00	0.10±0.00	0.10±0.00	0.10±0.00	1.00±0.00
		Diff Size	Mcut	0.16±0.01	0.42±0.03	0.69±0.05	0.74±0.05	0.79±0.05
			ncut	0.12±0.01	0.12±0.01	0.12±0.01	0.13±0.01	1.00±0.00
	Diff Volume	Same Size	Mcut	0.20±0.04	0.60±0.13	0.91±0.01	0.98±0.00	0.99±0.01
			ncut	0.10±0.00	0.10±0.00	0.10±0.00	0.10±0.00	0.64±0.47
		Diff Size	Mcut	0.22±0.04	0.55±0.07	0.72±0.05	0.74±0.05	0.79±0.05
			ncut	0.12±0.01	0.12±0.01	0.12±0.01	0.13±0.01	0.82±0.37
Non-sphere	Same Volume	Same Size	Mcut	0.73±0.04	0.99±0.00	0.99±0.01	0.93±0.04	0.86±0.05
			ncut	0.10±0.00	0.10±0.00	1.00±0.00	1.00±0.00	1.00±0.00
		Diff Size	Mcut	0.68±0.04	0.75±0.05	0.77±0.04	0.81±0.04	0.82±0.07
			ncut	0.13±0.01	0.29±0.35	0.99±0.02	1.00±0.00	1.00±0.00
	Diff Volume	Same Size	Mcut	0.83±0.06	0.99±0.00	0.97±0.03	0.93±0.03	0.89±0.05
			ncut	0.10±0.00	0.19±0.29	0.82±0.38	1.00±0.01	1.00±0.00
		Diff Size	Mcut	0.69±0.05	0.76±0.05	0.79±0.03	0.80±0.03	0.80±0.06
			ncut	0.12±0.01	0.38±0.41	0.88±0.28	0.99±0.01	1.00±0.00

Table 4.4 Percentage of Win for Min-max Cut and Normalized Cut with Kernighan-Lin

	Min-max Cut	Normalized Cut
Spherical	83%	17%
Non-spherical	39%	61%
Same volume	59%	41%
Different volume	63%	37%
Same size	63%	37%
Different size	60%	40%

4.3 Optimization Approaches

Once an objective function is defined, we can search the solution space exhaustively for the best possible objective value. Unfortunately this is not practical even for very small

input size. Many search heuristics appear to the rescue. These heuristics differ both in the exhaustiveness of their searches and their search trajectories in the solution space.

K-means algorithm uses a greedy optimization procedure. It searches by always moving from a candidate solution to a better candidate solution, and therefore tends to get stuck at local optima. The other extreme in heuristics is simulated annealing, which accepts random worse moves hoping to get out of local optima [66]. However simulated annealing is slow and requires significant parameter tuning. Kernighan-Lin is somewhere in the middle of the spectrum defined in terms of both the quality of the local optima and time complexity. The approach has been shown to out-perform simulated annealing in partitioning graphs with some structure [59]. Therefore we compare Kernighan-Lin to the greedy approach and reveal how much quality can be gained by the substitution.

Both greedy and Kernighan-Lin approaches have two stages: initial configuration and refinement. The initial assignment stage produces a starting configuration of k clusters for the refinement stage to work on. In our implementation the two approaches share the same initial configuration stage. The refinement stages of both approaches consider moving an object from one cluster to another and determining whether this is a "better move", i.e., the objective value is better, or a "worse move". The two approaches differ in whether to go on searching when encountering worse moves. The greedy approach only accepts better moves. If there are no better moves, the procedure terminates. Kernighan-Lin is a procedure that accepts some worse moves when refining clusters, in the hope that eventually the objective function is improved.

We supply the algorithms with the desired number of clusters, which in our case is the number of classes given by the classified datasets. Therefore we are studying the algorithmic behavior when the algorithm produces the same number of clusters as the classes. The question of determining how many clusters in a data set is important and interesting. However it is beyond the scope of this thesis.

4.3.1 Greedy

The greedy approach starts with an initial configuration stage [119]. There are different ways to obtain an initial configuration. We choose a simple version, which first picks k seed points uniformly randomly from the input data set. Then the remaining points are assigned to the seed point with the closest distance. The refinement stage consists of several passes. Each pass considers each point in the data set in random order and determines if moving it to any other cluster will improve the objective. If an improvement exists the point is moved to the cluster that will result in the largest improvement in the objective function. Cluster centers are updated after each move. The refinement terminates as soon as no point is moved. The algorithm is greedy in the sense that it only moves those points that will improve the objective.

We include the pseudo-code of the greedy approach as follows.

1. Random pick k points from input. Call them seeds.
2. Assign every point to its closest seed.
3. Calculate the centers of all clusters.
4. Randomly iterate through all input points. For each point p_k,

4.1 Calculate its distance to every cluster center.

4.2 Record the closest center.

4.3 If p_k is not in the cluster whose center it is closest to, move the point there.

4.4 Recalculate the centers of all clusters.

5. Repeat Step 4 until no point needs moving.

4.3.2 Kernighan-Lin

The Kernighan-Lin approach shares the same initial configuration stage with the greedy approach. They differ in the refinement strategy. The refinement stage of Kernighan-Lin also consists of several passes [65]. In each pass, every point is moved exactly once, to the cluster that achieves the best possible change of the objective value. For each pass, the algorithm keeps a record of the so-far best objective and its configuration, which is returned at the end of each pass. The algorithm also keeps track of the current objective and configuration. The best objective is not necessarily the same as the current objective because a point is moved to the destination cluster that results in the best possible change, even if the change is worsening the current objective value. In other words the refinement accepts certain worse moves. If the temporary dip does lead to objectives better than the best objective encountered so far, the records are updated. Otherwise the best objective remains the same. The refinement ends when a pass finishes with no better objective than the last pass.

The Kernighan-Lin approach explores more solution space than the greedy approach because it sometimes searches down-hill in the hope to climb higher than the previous local peak. In contrast the greedy approach is a strict hill-climbing procedure and hence its tendency to stuck at local optima. Meanwhile the Kernighan-Lin approach exhibits a certain degree of flexibility of getting out of local optima.

The pseudo-code of the Kernighan-Lin approach is as follows.

1. Randomly pick k points from input. Call them seeds.

2. Assign every point to its closest seed.

3. Calculate the objective value for the current clusters and assign it to cur_best_value. Record the configuration in best_config and cur_config.

4. Repeat until the change between the best objectives from two consecutive executions of this outer loop is smaller than a threshold (the threshold is currently 0.0001).

    (1) Label each point free.

    (2) While there are free points

        (i) Calculate the new objective values for each free point to move to cluster j, $j \in [1..k]$ from cur_config.

        (ii) Choose the free point with best new objective values to move. Store the resulting objective value in cur_candidate_value. Record the move in cur_config. Tag the point not free.

        (iii) If cur_candidate_value is better than cur_best_value, update cur_best_value with cur_candidate_value and best_config with cur_config.

5. The result clusters are recorded in best_config. The resulting objective value is stored in cur_best_value.

Note the version of Kernighan-Lin approach we use in our experiments updates the current configuration after checking all the free points at the time being and choosing the point with the most gain. Therefore every update takes O(nk) calculations of the objective functions, where n is the number of points and k is the number of clusters. There is an alternate version that calculates O(k) times of objective functions for Step 5. For every update of current configuration, the algorithm checks each free point and determines which cluster it moves to if a better objective is available. The latter version does not search the solution space as exhaustively as the version we use, but it runs in shorter time and might be a choice if time is a concern for some applications.

4.3.3 Random Starts

The initial configuration stage of both optimization approaches asks for random seeds to start a clustering. Sometimes seeds are good: when they are close to the centers of underlying classes. Sometimes they are bad because they cluster together and subsequent movements cannot help much. This invites questions such as whether the poor quality of an algorithm is due to bad seeds, instead of the objective function or the optimization approach. To compensate for the randomness of seeds, we repeat the same algorithm on the same dataset ten times. Each time a different set of random seeds is chosen and the objective function value of the final clustering is recorded. Of the ten runs, the one run with the best ending objective values is picked to be the final output.

This is a standard practice reported in the literature. We hope the repetition can do justice to the algorithms.

The number ten is used in the literature without much justification [119]. Since we are comparing our findings with those in the literature, we use ten random starts in our main experiments to be more directly comparable to other reported results. However we are left with the question of how many runs are necessary so that we can have confidence that the resulting objective value is close to the optimal objective value of clusterings from all possible seeds.

In this section we investigate the appropriate number of random starts as to obtain fair quality of clusters. If the optimal value for ten random starts is close enough to the optimal value of all possible starts, we are confident in using the result from ten runs to represent the behavior of an algorithm. Otherwise it would be a good idea to try more runs.

To answer this question, we have the algorithms running a hundred runs on the same dataset and record the objective values achieved for these runs. The range between minimum and maximum objective value is then divided equally into ten intervals and a histogram is calculated. The first count details how many objective values falls into the range whose width is ten percent of the total range from minimum to maximum, starting from minimum value of the hundred runs. It gives an idea of how often the objectives ends up within close proximity to the minimum of a hundred runs. If the first count is 10 or more, that means the probability of getting an ending objective value close to the

minimum is roughly 0.1. Therefore we have a fair chance of getting a good enough value by repeating ten different random starts.

In Figure 4.1 we show an aggregated histogram of ending objective values for the 400 synthetic data files, using the algorithm sum-of-squares with greedy. For each data file a histogram of ending objective values for the hundred runs is calculated. We average the counts in the individual histograms respectively. The idea is to treat each range between minimum and maximum objective values of a hundred runs on a single data file as a unit range. Therefore the counts can be added together. In Figure 4.1, the x-axis represents this unit range, where zero means the minimum objective achieved during a hundred runs and one indicates the maximum in the same hundred runs. An interval of this unit range represents a certain percentage of total distance away from the minimum value. The y-axis shows the number of ending objectives that falls into the corresponding range of values. Note all data points in Figure 4.1 should add up to 100.

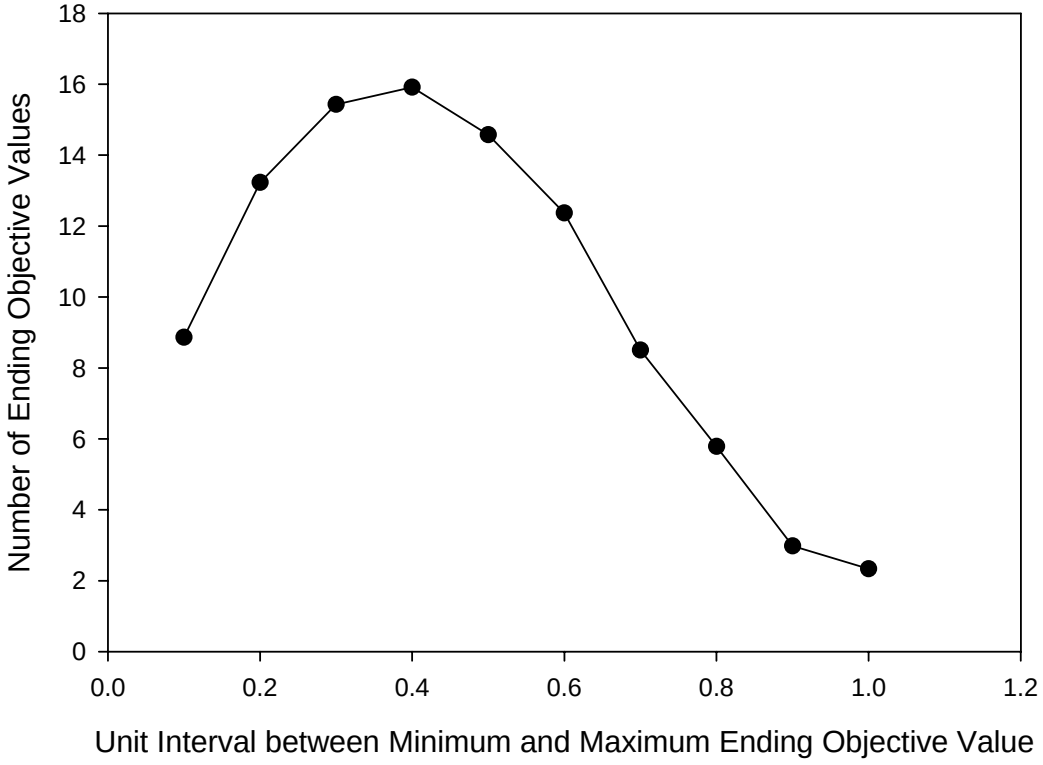


Figure 4.1 Average Histogram of Objective Values for A Hundred Runs – SOS w/ Grd

In Figure 4.1 the first count represents the number of objective values that fall within ten percent of the minimum value for a hundred runs. We observe the first count is about 9. Unfortunately the standard deviation is as high as 7.0013. To know if significant outliers cause the large variances, we check into the individual 400 first counts. Of the 400 first counts, 135 values are below half the average, that is, roughly 4.5, while 85 are larger than the average by at least a half, that is, roughly 13.5.

In Figure 4.2 we show the distribution for the 400 first counts. At one extreme of the 400 synthetic data files, out of a hundred random starts, 47 runs end up within ten percent of the minimum objective. At the other extreme, only 1 run out of 100 random

starts produces an ending objective within the same neighborhood of the minimum. The x-axis represents the first counts, that is, the number of ending objectives to fall within ten percent of the minimum ending objective of a hundred runs, for the algorithm sum-of-squares with greedy. The range from the minimum count to the maximum count is divided equally into ten intervals. The y-axis gives the number of first counts that falls into that interval. This is a histogram of the individual first counts that constitute the first data point in Figure 4.1. Note the sum of the data points in Figure 4.2 is 400.

We show the distribution to investigate how the number of ending objectives that fall within ten percent of the minimum, distributes around its average. Understanding the distribution can help answer the question how many random starts are needed to have a fair chance to obtain a good enough ending objective for a certain algorithm.

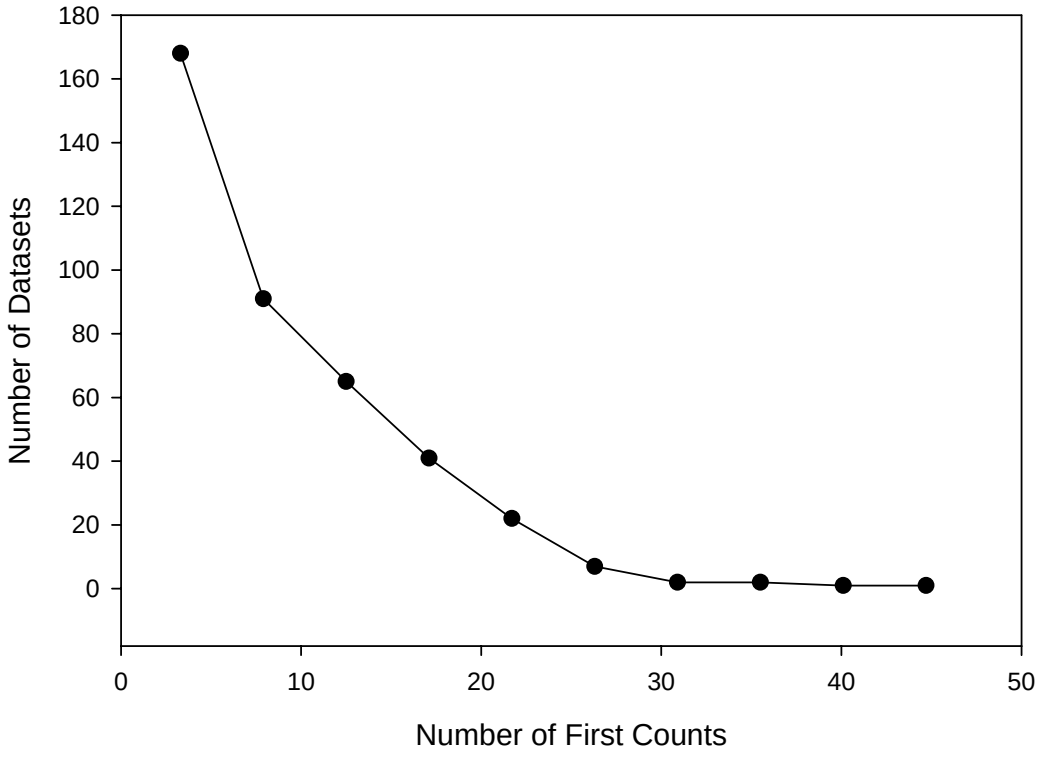


Figure 4.2 Histogram of First Counts – SOS with Greedy

The average number of ending objectives to fall into ten percent within the minimum for 400 synthetic data files is 8.86, which is the second bucket in Figure 4.2. The first and second data point in Figure 4.2 represents the files whose number of ending objectives that are within ten percent of the minimum is less than 10, out of a hundred runs. Approximately 300 files are behind these two points.

This leads us to believe that ten sets of random seeds are not enough for us to have confidence in the closeness of the minimum of ten runs to the minimum of a hundred runs for the algorithm sum-of-squares with greedy. In fact for roughly 170 files, less than five ending objectives, out of a hundred values, end up within ten percent of the

optimum. Chances are the best objective of ten runs is more than ten percent away from the minimum of a hundred runs. If twenty runs are employed, at least for half the datasets, there is a fair chance to obtain a close enough representative.

One drawback of the above analysis is by stretching every range between minimum and maximum values to a unit range we are penalizing the data files whose ranges between minimum and maximum ending objective values are small. On the other hand, it is very hard to introduce absolute threshold in terms of an objective value, because the objective of sum-of-squares on these data files has a wide range.

We have presented our analysis for the algorithm of sum-of-squares with greedy. However for the other three iterative algorithms, similar analyses draw the same conclusion. For example, for the algorithm sum-of-squares with Kernighan-Lin, we present the aggregated histogram in Figure 4.3 and the distribution of the first counts in Figure 4.4.

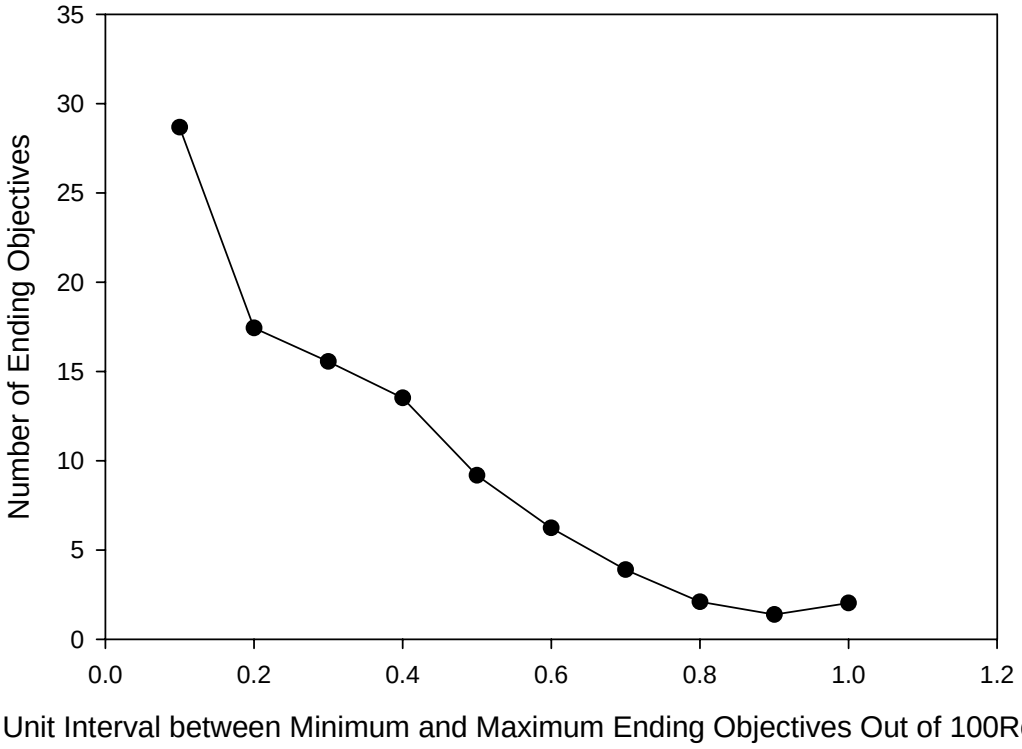


Figure 4.3 Average Histogram of Ending Objectives for A Hundred Runs—SOS w/ KL

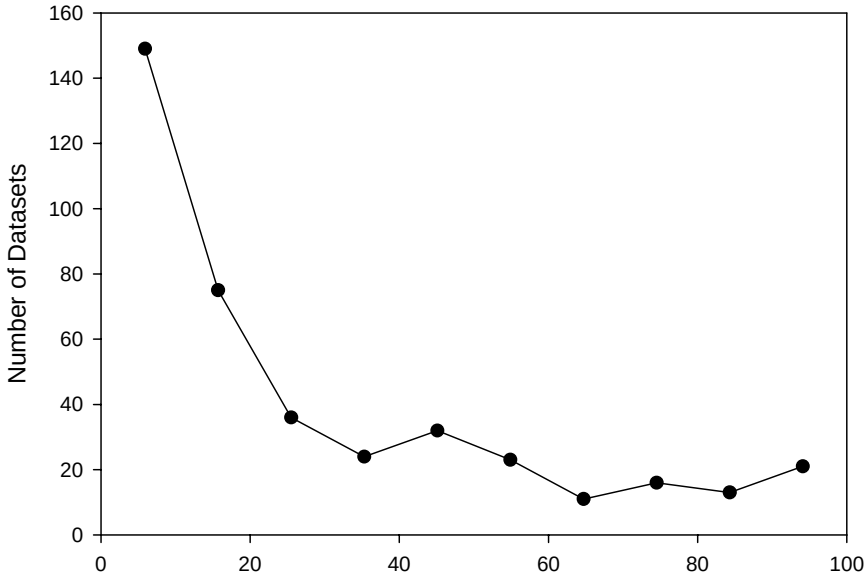


Figure 4.4 Distribution of First Counts – SOS with Kernighan-Lin

From Figure 4.4 we observe on 150 synthetic data files, less than 10 ending objectives, out of a hundred runs, fall into ten percent of the minimum value. There are similar plots for the two algorithms using the min-max cut objective. They resemble the plots in Figure 4.3 and 4.4 and therefore we spare the redundancy of showing them.

Note we have made two assumptions in the above reasoning. First we assume the optimal objective value can be achieved by many random trials. In this case we simulate the minimum objective value from all possible random starts by the minimum of a hundred runs. The second assumption is that the objective values achieved from separate random seeds are independent of each other. That is to say the appearance of a value does not affect the appearance of another value. Therefore if ten times out of a

hundred a value is close to the optimum of a hundred runs, once out of ten a value is close to the optimum of a hundred runs.

We realize the time constraint facing most users of clustering algorithms may prohibit running many runs may not be feasible. At least in this case people should be aware that the clusterings obtained from these iterative algorithms may be not close to the best achievable.

4.4 Two Objectives Compared under Same Optimization

In this section we detail the comparison of the cluster quality between the two objectives by using the same optimization approach. By doing this we examine whether the relative performance using different objectives changes when the optimization approaches change. In all the tables in this chapter, each entry contains two numbers. The one before the ± sign is the average while the number after the sign is the standard deviation. The average is taken over ten input files for the same case. Please refer to Chapter 3 for the explanation on the details of the construction of the datasets. For the four iterative clustering algorithms under investigation, for each input file, ten sets of random starts are generated. The output from an input file is the run with the best objective value out of the ten random starts.

4.4.1 Sum-of-Squares vs. Min-max Cut, Greedy

In Table 4.5 we show the average f-scores for the algorithms sum-of-squares with greedy and min-max cut with greedy.

Table 4.5 Average F-scores for Sum-of-Squares and Min-max Cut under Greedy

Greedy				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Sos	0.15±0.00	0.36±0.04	0.83±0.01	0.93±0.04	0.94±0.04
			Mcut	0.15±0.01	0.39±0.04	0.86±0.01	0.97±0.00	0.99±0.00
		Diff Size	Sos	0.15±0.01	0.41±0.03	0.79±0.03	0.90±0.04	0.92±0.02
			Mcut	0.16±0.01	0.42±0.03	0.69±0.05	0.73±0.05	0.79±0.04
	Diff Volume	Same Size	Sos	0.20±0.03	0.57±0.13	0.86±0.04	0.93±0.04	0.93±0.04
			Mcut	0.22±0.05	0.63±0.12	0.91±0.02	0.98±0.01	0.99±0.00
		Diff Size	Sos	0.24±0.05	0.59±0.11	0.82±0.06	0.91±0.04	0.92±0.05
			Mcut	0.24±0.05	0.57±0.07	0.72±0.05	0.74±0.05	0.78±0.05
Non-sphere	Same Volume	Same Size	Sos	0.70±0.05	0.94±0.04	0.91±0.07	0.90±0.04	0.89±0.05
			Mcut	0.73±0.04	0.99±0.00	0.99±0.01	0.96±0.02	0.91±0.03
		Diff Size	Sos	0.67±0.05	0.88±0.05	0.87±0.06	0.89±0.05	0.87±0.06
			Mcut	0.68±0.04	0.76±0.05	0.78±0.04	0.79±0.04	0.79±0.06
	Diff Volume	Same Size	Sos	0.75±0.09	0.92±0.06	0.88±0.09	0.92±0.04	0.88±0.06
			Mcut	0.84±0.06	0.99±0.00	0.98±0.02	0.91±0.03	0.88±0.04
		Diff Size	Sos	0.73±0.09	0.87±0.06	0.89±0.05	0.88±0.04	0.88±0.06
			Mcut	0.70±0.05	0.76±0.04	0.79±0.04	0.78±0.05	0.81±0.06

We summarize the win-lose findings in Table 4.6.

Table 4.6 Percentage of Win for Greedy

	Sum-of-squares	Min-max cut
Spherical	43%	57%
Non-spherical	59%	41%
Same volume	49%	51%
Different volume	53%	47%
Same size	19%	81%
Different size	83%	17%

From the table we observe that under the greedy approach, sum-of-squares and min-max cut produces clusters with comparable quality in four out of six categories. If

classes have same sizes, min-max cut wins overwhelmingly. However if classes have different sizes, sum-of-squares wins by a large margin.

4.4.2 Sum-of-Squares vs. Min-max Cut, Kernighan-Lin

In Table 4.7 we show the f-scores for the two algorithms sum-of-squares with Kernighan-Lin and min-max cut with Kernighan-Lin.

Table 4.7 Average F-scores for Sum-of-Squares and Min-max Cut under Kernighan-Lin

KL				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	sos	0.15±0.01	0.45±0.05	0.84±0.01	0.96±0.01	0.98±0.03
			mcut	0.15±0.01	0.40±0.03	0.85±0.01	0.97±0.00	0.99±0.00
		Diff Size	sos	0.16±0.01	0.46±0.03	0.81±0.03	0.91±0.03	0.91±0.04
			mcut	0.16±0.01	0.42±0.03	0.69±0.05	0.74±0.05	0.79±0.05
	Diff Volume	Same Size	Sos	0.22±0.04	0.64±0.11	0.86±0.03	0.96±0.04	0.95±0.05
			Mcut	0.20±0.04	0.60±0.13	0.91±0.01	0.98±0.00	0.99±0.01
		Diff Size	Sos	0.25±0.05	0.61±0.10	0.84±0.05	0.92±0.05	0.92±0.04
			mcut	0.22±0.04	0.55±0.07	0.72±0.05	0.74±0.05	0.79±0.05
Non-sphere	Same Volume	Same Size	sos	0.72±0.05	1.00±0.01	1.00±0.00	1.00±0.00	0.98±0.03
			mcut	0.73±0.04	0.99±0.00	0.99±0.01	0.93±0.04	0.86±0.05
		Diff Size	sos	0.69±0.07	0.92±0.04	0.96±0.02	0.94±0.03	0.91±0.03
			mcut	0.68±0.04	0.75±0.05	0.77±0.04	0.81±0.04	0.82±0.07
	Diff Volume	Same Size	sos	0.79±0.10	0.99±0.02	0.96±0.07	0.95±0.08	0.93±0.06
			mcut	0.83±0.06	0.99±0.00	0.97±0.03	0.93±0.03	0.89±0.05
		Diff Size	sos	0.74±0.10	0.93±0.06	0.94±0.03	0.92±0.03	0.91±0.04
			mcut	0.69±0.05	0.76±0.05	0.79±0.03	0.80±0.03	0.80±0.06

In Table 4.8 we summarize the win-lose comparison between the two algorithms.

Table 4.8 Percentage of Win for Kernighan-Lin

	SOS	MCUT	Tie
Spherical	52%	36%	12%
Non-spherical	40%	41%	19%
Same volume	43%	40%	17%
Diff volume	50%	38%	12%
Same size	39%	31%	30%
Diff size	54%	46%	0%

From these two tables we find under the Kernighan-Lin approach, sum-of-squares wins over min-max cut, sometimes by large margin, sometimes by smaller margins, in all except one category, which is the non-spherical case. The performance difference between the two objectives in the size categories disappears. Instead, sum-of-squares wins in both the same-size and different-size cases. Also there are many ties in contrast to under the greedy approach, where there are no ties at all.

4.5 Two Objectives Compared under Different Optimization

In this subsection we compare the performance of a single objective under two optimizations. We hope this can help understand how each objective responds to the change of optimization approach.

4.5.1 Sum-of-Squares, Greedy vs. KL

In Table 4.9 we describe the f-scores of the two algorithms, sum-of-squares with Greedy and sum-of-squares with KL, on the synthetic datasets.

Table 4.9 Average F-scores for Sum-of-Squares under Greedy and Kernighan-Lin

Sos				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Grd	0.15±0.00	0.36±0.04	0.83±0.01	0.93±0.04	0.94±0.04
			KL	0.15±0.01	0.45±0.05	0.84±0.01	0.96±0.01	0.98±0.03
		Diff Size	Grd	0.15±0.01	0.41±0.03	0.79±0.03	0.90±0.04	0.92±0.02
			KL	0.16±0.01	0.46±0.03	0.81±0.03	0.91±0.03	0.91±0.04
	Diff Volume	Same Size	Grd	0.20±0.03	0.57±0.13	0.86±0.04	0.93±0.04	0.93±0.04
			KL	0.22±0.04	0.64±0.11	0.86±0.03	0.96±0.04	0.95±0.05
		Diff Size	Grd	0.24±0.05	0.59±0.11	0.82±0.06	0.91±0.04	0.92±0.05
			KL	0.25±0.05	0.61±0.10	0.84±0.05	0.92±0.05	0.92±0.04
Non-sphere	Same Volume	Same Size	Grd	0.70±0.05	0.94±0.04	0.91±0.07	0.90±0.04	0.89±0.05
			KL	0.72±0.05	1.00±0.01	1.00±0.00	1.00±0.00	0.98±0.03
		Diff Size	Grd	0.67±0.05	0.88±0.05	0.87±0.06	0.89±0.05	0.87±0.06
			KL	0.69±0.07	0.92±0.04	0.96±0.02	0.94±0.03	0.91±0.03
	Diff Volume	Same Size	Grd	0.75±0.09	0.92±0.06	0.88±0.09	0.92±0.04	0.88±0.06
			KL	0.79±0.10	0.99±0.02	0.96±0.07	0.95±0.08	0.93±0.06
		Diff Size	Grd	0.73±0.09	0.87±0.06	0.89±0.05	0.88±0.04	0.88±0.06
			KL	0.74±0.10	0.93±0.06	0.94±0.03	0.92±0.03	0.91±0.04

In Table 4.10 we give the win-lose comparison between the two algorithms.

Table 4.10 Percentage of Win for SOS w/ Grd and SOS w/ KL

	SOS with Greedy	SOS with KL	Tie
Spherical	26%	73%	1%
Non-spherical	14%	83%	3%
Same Volume	18%	80%	2%
Different Volume	22%	75%	3%
Same Size	16%	80%	4%
Different Size	24%	76%	0%

The objective sum-of-squares responds well to the change to a more exhaustive search strategy from the greedy approach to the Kernighan-Lin approach. The quality of

clusters benefits from the change. The algorithm of sum-of-squares with Kernighan-Lin consistently wins over the algorithm of sum-of-squares with greedy by a large margin.

4.5.2 Min-max Cut, Greedy vs KL

In Table 4.11 we detail the performance of the two algorithms, min-max cut with greedy and min-max cut with Kernighan-Lin.

Table 4.11 Average F-scores for Min-max Cut under Greedy and KL

Mcut				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Grd	0.15±0.01	0.39±0.04	0.86±0.01	0.97±0.00	0.99±0.00
			KL	0.15±0.01	0.40±0.03	0.85±0.01	0.97±0.00	0.99±0.00
		Diff Size	Grd	0.16±0.01	0.42±0.03	0.69±0.05	0.73±0.05	0.79±0.04
			KL	0.16±0.01	0.42±0.03	0.69±0.05	0.74±0.05	0.79±0.05
	Diff Volume	Same Size	Grd	0.22±0.05	0.63±0.12	0.91±0.02	0.98±0.01	0.99±0.00
			KL	0.20±0.04	0.60±0.13	0.91±0.01	0.98±0.00	0.99±0.01
		Diff Size	Grd	0.24±0.05	0.57±0.07	0.72±0.05	0.74±0.05	0.78±0.05
			KL	0.22±0.04	0.55±0.07	0.72±0.05	0.74±0.05	0.79±0.05
Non-sphere	Same Volume	Same Size	Grd	0.73±0.04	0.99±0.00	0.99±0.01	0.96±0.02	0.91±0.03
			KL	0.73±0.04	0.99±0.00	0.99±0.01	0.93±0.04	0.86±0.05
		Diff Size	Grd	0.68±0.04	0.76±0.05	0.78±0.04	0.79±0.04	0.79±0.06
			KL	0.68±0.04	0.75±0.05	0.77±0.04	0.81±0.04	0.82±0.07
	Diff Volume	Same Size	Grd	0.84±0.06	0.99±0.00	0.98±0.02	0.91±0.03	0.88±0.04
			KL	0.83±0.06	0.99±0.00	0.97±0.03	0.93±0.03	0.89±0.05
		Diff Size	Grd	0.70±0.05	0.76±0.04	0.79±0.04	0.78±0.05	0.81±0.06
			KL	0.69±0.05	0.76±0.05	0.79±0.03	0.80±0.03	0.80±0.06

In Table 4.12 we present the win-lose comparison between the two algorithms.

Table 4.12 Percentage of Win for MCUT w/ Grd and MCUT w/ KL

	Mcut with Greedy	Mcut with Kernighan-Lin	Tie
Spherical	52%	36%	12%
Non-spherical	40%	41%	19%
Same Volume	43%	40%	17%
Different Volume	50%	38%	12%

Same Size	39%	31%	30%
Different Size	54%	47%	0%

When applying the Kernighan-Lin approach to the objective min-max cut, no big improvement in cluster quality happens. In fact, not only there are many ties, in five out of six categories, the algorithm min-max cut with greedy wins over the supposedly better algorithm min-max cut with Kernighan-Lin. In contrast to the big improvement happening to the sum-of-squares objective, changing to a more exhaustive search does not help produce better local optima at all. We consider the reasons behind this behavior in section 4.7.

4.6 Four Algorithms Comparison

So far we have presented pair-wise comparisons between these algorithms, focusing on the effects of the objective functions or the optimization approaches. To give a clearer picture of the four algorithms we detail the performance of the four algorithms head-to-head in this subsection.

4.6.1 Sum-of-Squares, KL vs Min-max Cut, Greedy

From the aforementioned sections we conclude that in terms of performance sum-of-squares with Kernighan-Lin wins over sum-of-squares with greedy. We also conclude that the performance of the two algorithms with min-max cut objective is roughly the same. Min-max cut with greedy is slightly better in terms of the winning percentage.

Therefore to know the winner of the four algorithm in terms of quality, we present the comparison between sum-of-squares with Kernighan-Lin and min-max cut with greedy.

In Table 4.13 we give the average f-scores for the two algorithms on the synthetic datasets.

Table 4.13 Average F-scores for SOS w/ KL and MCUT with Grd

				0.25	0.5	0.75	1.0	2.0
Sphere	Same Vol	Same Size	Sos/kl	0.15±0.01	0.45±0.05	0.84±0.01	0.96±0.01	0.98±0.03
			Mcut/grd	0.15±0.01	0.39±0.04	0.86±0.01	0.97±0.00	0.99±0.00
		Diff Size	Sos/kl	0.16±0.01	0.46±0.03	0.81±0.03	0.91±0.03	0.91±0.04
			Mcut/grd	0.16±0.01	0.42±0.03	0.69±0.05	0.73±0.05	0.79±0.04
	Diff Vol	Same Size	Sos/kl	0.22±0.04	0.64±0.11	0.86±0.03	0.96±0.04	0.95±0.05
			Mcut/grd	0.22±0.05	0.63±0.12	0.91±0.02	0.98±0.01	0.99±0.00
		Diff Size	Sos/kl	0.25±0.05	0.61±0.10	0.84±0.05	0.92±0.05	0.92±0.04
			Mcut/grd	0.24±0.05	0.57±0.07	0.72±0.05	0.74±0.05	0.78±0.05
Non-sphere	Same Vol	Same Size	Sos/kl	0.72±0.05	1.00±0.01	1.00±0.00	1.00±0.00	0.98±0.03
			Mcut/grd	0.73±0.04	0.99±0.00	0.99±0.01	0.96±0.02	0.91±0.03
		Diff Size	Sos/kl	0.69±0.07	0.92±0.04	0.96±0.02	0.94±0.03	0.91±0.03
			Mcut/grd	0.68±0.04	0.76±0.05	0.78±0.04	0.79±0.04	0.79±0.06
	Diff Vol	Same Size	Sos/kl	0.79±0.10	0.99±0.02	0.96±0.07	0.95±0.08	0.93±0.06
			Mcut/grd	0.84±0.06	0.99±0.00	0.98±0.02	0.91±0.03	0.88±0.04
		Diff Size	Sos/kl	0.74±0.10	0.93±0.06	0.94±0.03	0.92±0.03	0.91±0.04
			Mcut/grd	0.70±0.05	0.76±0.04	0.79±0.04	0.78±0.05	0.81±0.06

In Table 4.14 we list the win-lose comparison between the two algorithms.

Table 4.14 Percentage of Win for SOS w/ KL and Mcut w/ Grd

	SOS with KL	Mcut with Greedy
Spherical	66%	34%
Non-spherical	85%	15%
Same Volume	80%	20%
Different Volume	72%	28%
Same Size	59%	41%
Different Size	93%	7%

From the two tables we conclude the sum-of-squares with Kernighan-Lin beats min-max cut with greedy consistently in all categories. The algorithm with the best cluster quality out of the four candidates is the sum-of-squares with Kernighan-Lin. We had expected the min-max cut with greedy would win over sum-of-squares with greedy because the cut objective is claimed not to favor any cluster shapes and half our synthetic data is non-spherical. The second expectation we had was a more exhaustive search would yield better quality and therefore the min-max cut with Kernighan-Lin should come out as the winner. Both expectations are wrong.

4.6.2 Four algorithms on real data

Besides synthetic datasets, we also apply the four algorithms to the real datasets downloaded from the Web. Please refer to Chapter 3 for details of these datasets. The f-scores for these algorithms on real datasets are in Table 4.15. In Table 4.16 we list the difference between f-scores for two algorithms, whose names are shown at the first row of the table.

Table 4.15 F-scores for Four Algorithms on Real Datasets

	SOS with Greedy	Mcut with Greedy	SOS with KL	Mcut with KL
Re0	0.496	0.397	0.454	0.428
Re1	0.483	0.436	0.491	0.442
Tr31	0.697	0.593	0.694	0.568
Tr41	0.644	0.687	0.730	0.692
Wap	0.455	0.501	0.519	0.519
Fbis	0.549	0.522	0.608	0.500

Table 4.16 Algorithm Improvements

	Mcut/grd-sos/grd	Mcut/kl-sos/kl	Sos/kl – sos/grd	Mcut/kl – mcut/grd
Re0	-0.099	-0.026	-0.042	0.031
Re1	-0.047	-0.049	0.008	0.006
Tr31	-0.104	-0.126	-0.003	-0.025
Tr41	0.043	-0.038	0.086	0.005
Wap	0.046	0	0.064	0.018
Fbis	-0.027	-0.108	0.059	-0.022

For the algorithms using the greedy approach, on four out of six datasets, sum-of-squares produces better f-scores than min-max cut. On the remaining two datasets, tr41 and wap, min-max cut yields clusters with better quality. When applying the Kernighan-Lin algorithms, sum-of-squares consistently outperforms min-max cut, except that on one dataset, they score the same.

Comparing optimization approaches, the Kernighan-Lin approach can improve sum-of-squares over greedy on four datasets, while on the other two, re0 and tr31, greedy with sum-of-squares has better f-scores. For the objective min-max cut, again, on four datasets, the Kernighan-Lin approach can improve on greedy, while on datasets tr31 and fbis, greedy wins over Kernighan-Lin.

We consider such findings confirm the following three observations on the synthetic datasets. First, the sum-of-squares with greedy algorithm outperforms min-max cut with greedy, because the datasets consist of classes with different sizes. Secondly, sum-of-squares with Kernighan-Lin improves comparing to sum-of-squares with greedy, because Kernighan-Lin helps sum-of-squares. Thirdly, sum-of-squares with Kernighan-

Lin wins over min-max cut Kernighan-Lin, making sum-of-squares with Kernighan-Lin the best performing algorithm out of the four, in terms of cluster quality.

The difference between algorithmic behavior on real and synthetic datasets happens when min-max cut with Kernighan-Lin improves on min-max cut with greedy on four datasets. Previously, we observe the Kernighan-Lin approach cannot improve min-max cut much. However, on real datasets, the f-scores for min-max cut with Kernighan-Lin are better than min-max cut with greedy.

4.7 Observations

4.7.1 KL cannot improve min-max cut much

We observe from Table 4.11 and Table 4.12 that Kernighan-Lin does not improve the objective min-max cut, making its winning margin over sum-of-squares in same-sized classes under greedy disappear. Meanwhile the same Kernighan-Lin approach helps sum-of-squares improve cluster quality. We propose the following possible reasons:

The initial assignment stage used by both optimization approaches is favoring the greedy approach because it creates the initial clusters by assigning input points to the closest cluster center. We call this scheme the seeded assignment scheme.

The greedy optimization is closing in on the optimum value of the min-max cut objective of the graph and therefore the better optimization approach cannot further improve it.

We implement a random initial assignment for both approaches. We randomly shuffle the order of the input points and chop up the sequence into initial clusters by using randomly generated indices as the cluster boundaries. We do not see a difference in the behaviors of min-max cut objectives for either greedy or Kernighan-Lin. We present the f-scores for the two algorithms, min-max cut with greedy and min-max cut with Kernighan-Lin, using the random initial assignment, in Table 4.17.

Table 4.17 F-scores for Min-max cut Algorithms Using Random Initial Assignment

Mcut				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Grd	0.15±0.01	0.40±0.03	0.86±0.01	0.97±0.00	0.99±0.00
			KL	0.15±0.01	0.37±0.04	0.85±0.01	0.97±0.00	0.99±0.00
		Diff Size	Grd	0.16±0.01	0.42±0.03	0.70±0.04	0.74±0.05	0.78±0.04
			KL	0.16±0.01	0.43±0.03	0.70±0.05	0.73±0.05	0.78±0.04
	Diff Volume	Same Size	Grd	0.21±0.04	0.63±0.12	0.91±0.01	0.98±0.00	0.99±0.00
			KL	0.21±0.04	0.62±0.14	0.91±0.02	0.98±0.00	0.99±0.00
		Diff Size	Grd	0.23±0.04	0.57±0.06	0.72±0.05	0.74±0.05	0.78±0.04
			KL	0.23±0.04	0.56±0.07	0.72±0.05	0.75±0.05	0.78±0.05
Non-sphere	Same Volume	Same Size	Grd	0.70±0.03	0.99±0.00	0.99±0.00	0.99±0.00	0.99±0.00
			KL	0.72±0.03	0.99±0.00	0.99±0.00	0.99±0.00	0.99±0.00
		Diff Size	Grd	0.67±0.05	0.76±0.05	0.77±0.04	0.78±0.03	0.80±0.04
			KL	0.67±0.05	0.75±0.05	0.78±0.04	0.79±0.03	0.83±0.03
	Diff Volume	Same Size	Grd	0.83±0.05	0.99±0.00	1.00±0.00	1.00±0.00	0.99±0.02
			KL	0.81±0.06	0.99±0.00	1.00±0.00	1.00±0.00	0.99±0.02
		Diff Size	Grd	0.69±0.05	0.76±0.04	0.78±0.04	0.79±0.04	0.80±0.05
			KL	0.70±0.05	0.76±0.05	0.78±0.04	0.78±0.04	0.81±0.06

We can see in Table 4.17 that Kernighan-Lin makes little improvement on the cut objective comparing with the greedy approach under this random initial assignment. Therefore we conclude the initial assignment is not the reason that Kernighan-Lin cannot improve min-max cut.

It is not practical to calculate the minimum min-max cut of a given graph because the calculation would be exponential in terms of the number of nodes in a graph. Note this optimum value is different from the optimum f-score. An optimum objective value does not necessarily lead to optimum f-scores. If the optimal min-max cut value is unknown, we cannot confirm the second speculation.

Instead we resort to check the improvement between the ending objectives and the starting objectives. If min-max cut values are improved more significantly than the sum-of-squares for the greedy approach, it offers supporting evidence that the cut objective is reaching its optimum even using a naïve optimization such as greedy.

To measure this improvement, the two algorithms have to start from a common initial assignment. We did not ensure this in previous experiments. However for this experiment we rerun the algorithms and make sure the two approaches start from the same initial clusters and therefore the same starting objective function values. The initial clusters are produced using the seeded assignment scheme.

We calculate the improvement as a ratio of the gain of the objective values to the starting objectives. This time we define a category as a combination of shape, volume, size and separation. There are 40 categories in total. Note the category here has a different definition than the category defined in previous parts of this chapter. We add all the improvement for a given category and average them. Each algorithm runs 10 times on each data file to compensate for the randomness of initial assignment. It is the range of an optimization to search that we are investigating. Therefore we add the

percentage gain of each of these runs, instead of adding only the gain from the run with best objective.

In Figure 4.5 we show the percentage gain for each objective and each optimization approach.

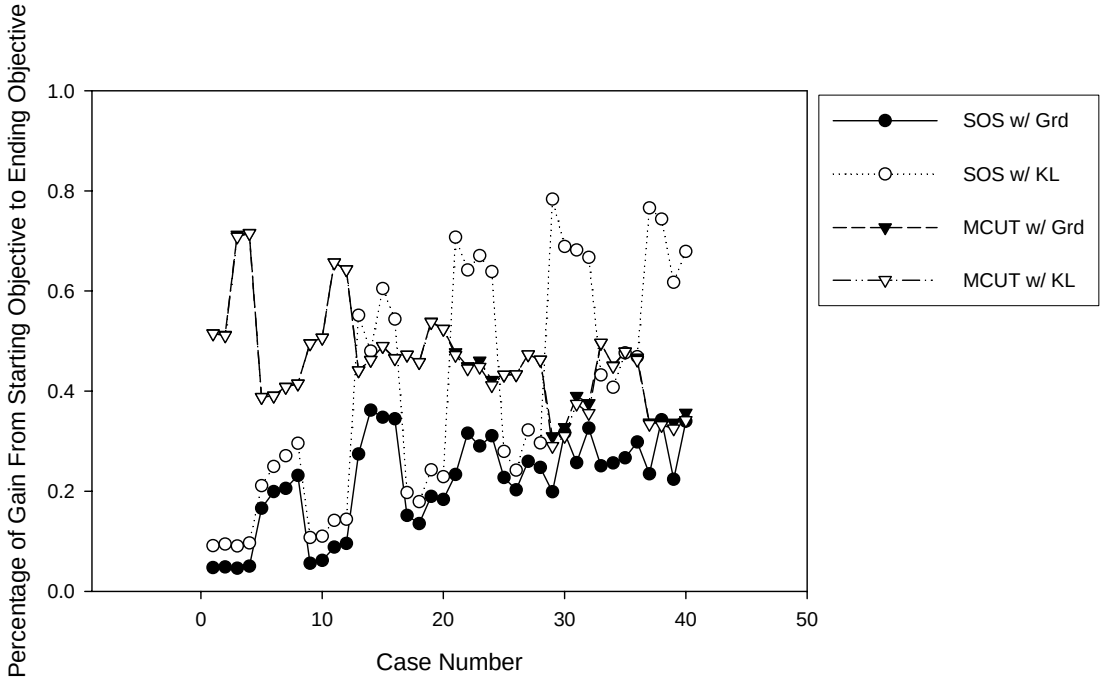


Figure 4.5 Percentage Gain for Each Objective and Optimization

We observe that for the mcut objective, the greedy approach improves the objective significantly more than the sum-of-squares objective. Although Kernighan-Lin continues to improve the sum-of-squares objective, it does little to change the cut objective for the better.

In Figure 4.6 we show the difference between the improvement percentages for the two approaches for both objectives. The figure reinforces of the idea that Kernighan-Lin is

not improving the min-max cut. Only this time the conclusion is coming from experiments when both approaches start from the same initial assignment.

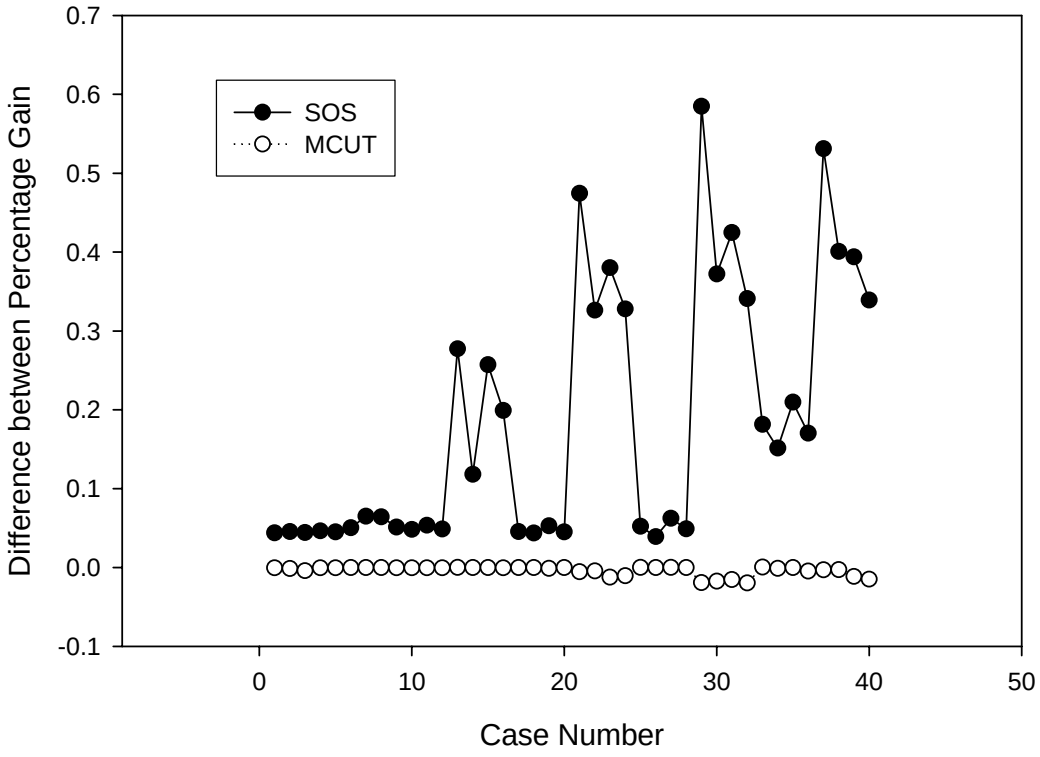


Figure 4.6 Added Percentage Gain for Grd and KL Using Common Starts

4.7.2 Efficiency

We did not implement the algorithms for efficiency; therefore the running times we have observed give only a rough comparison. The experiments have been run on a Pentium IV 1.4Ghz with 1Gb memory. The algorithms are implemented in Java. We report the running time that each algorithm uses to produce the results for all 400 files. Sum-of-squares with greedy finishes in approximately two minutes. Min-max cut with

either greedy or Kernighan-Lin stops around 15 minutes. Sum-of-squares with Kernighan-Lin finishes in roughly two hours. The Kernighan-Lin approach adds extra cost to each iteration compared with greedy. However it is the speed of convergence for each pass that dominates the total execution time. It takes longer for sum-of-squares to converge under Kernighan-Lin compared with min-max cut, making sum-of-squares with Kernighan-Lin the longest running algorithm of the four.

4.7.3 Conclusions on comparisons of algorithms

We draw a dominance graph in terms of quality for the four algorithms based on the comparisons. An up arrow means the algorithm at the arrowhead dominates the one at the end of the arrow. The double curves indicate the two algorithms have roughly comparable quality of clusters.

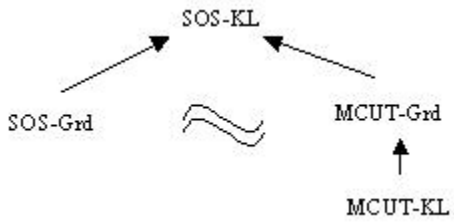


Figure 4.7 Dominance Graph For Four Algorithms

The winner of the four algorithms is sum-of-squares with Kernighan-Lin in terms of quality. The algorithm sum-of-squares with greedy ties with the min-max cut with greedy because they roughly have similar quality of clusters in both shape and volume categories. Moreover, sum-of-squares with greedy wins in the different sized category and min-max cut with greedy wins in the same-sized category. Given the fact that most

real-world datasets have different sized classes, we consider the algorithm sum-of-squares with greedy the runner up of the four. Note the algorithm sum-of-squares with greedy is actually a version of the clustering algorithm K-means.

To understand the relative quality gain for each algorithm, we aggregate the relative quality of the four algorithms on each synthetic dataset and calculate their average and standard deviations in Table 4.18. The relative quality of an algorithm is defined to be the ratio of the f-score for this algorithm to the maximum f-score of the four algorithms on the same dataset.

Table 4.18 Average Relative Quality and Standard Deviation of Four Candidate Algorithms

	Sos-grd	Sos-kl	Mcut-grd	Mcut-kl
Ave/standard deviation	0.934±0.060	0.983±0.034	0.918±0.077	0.912±0.076

Numbers in Table 4.18 roughly match the dominance graph drawn before based on the comparisons in last section. The only difference is between sum-of-squares with greedy and min-max cut with greedy. The relative quality of the later is smaller than the previous. However, we declare they roughly equal in the dominance graph, considering the fact that the min-max cut with greedy and the sum-of-squares with greedy win overwhelmingly in the same-sized cases and different-sized cases respectively.

K-means is a popular clustering algorithm due to its linear time complexity and easy implementation. However people have long believed that it sacrifices quality for speed. In our study, we vary the components of the k-means algorithm by substituting another

objective and another optimization approach. It turns out k-means ends up as the runner up in terms of cluster quality out of the four algorithms. Additionally, k-means is the fastest running one out of the four. It produces 95% of the quality from the best algorithm while runs an order of magnitude faster. We conclude that k-means is a rather good algorithm, both in its fast running times and in the quality of resulting clusters. It produces clusters with at least comparable quality to the min-max cut alternative, but with shorter running time.

When higher cluster quality is desired, we can resort to the algorithm sum-of-squares with Kernighan-Lin. With significant longer running time, the algorithm can find clusters of better quality than the K-means.

If we have the knowledge on the structure of the given datasets, such as they contain classes of the same size, the algorithm min-max cut with greedy can be applied to obtain better clusters than the traditional K-means. Other than this scenario, the min-max cut objective does not show any evidence to justify paying the cost of its longer running time.

Chapter 5 Objective Functions and Data Characteristics

In this chapter we investigate the behaviors of objective functions in response to data characteristics. We focus on the two objective functions, sum-of-squares and min-max cut, introduced in the last chapter. In previous literature the sum-of-squares has had a reputation for favoring spherical clusters while the min-max cut has been thought to be shape independent [64, 116]. The sum-of-squares has also been criticized for splitting the large clusters in situations where there are different sized clusters [50, 51, 64]. For datasets with non-spherical clusters with different sizes, one would have expected the min-max cut to excel. In fact in our study under these conditions the sum-of-squares performs better than the min-max cut.

In fact we discover a distinguished feature of min-max cut: it works well with same size clusters but miserably with different sized clusters. The objective, under both optimization approaches, produces consistently lower f-scores for datasets with different sized classes than those for datasets with same sized classes. We give detailed analysis of this behavior. Another interesting phenomenon we observe is that even for

datasets with same sized clusters, min-max cut with greedy wins by the least percentage in the category of non-spherical clusters, comparing to sum-of-squares with greedy.

"Data characteristics" is a general term, which has varied definitions across application domains. In our study we choose to define them in the most intuitive and controllable way, hoping to shed light on relationships between objective functions and data characteristics. By studying the relationships we hope to identify certain features of the objective functions such that these findings can be used to predict the winning algorithm for previously unseen datasets.

5.1 Data Characteristics Recap

We have described the data generative model in Chapter 3. For the ease of discussion we summarize the key points here. In the mixture of Gaussian model, we control three factors of a Gaussian: its shape, volume and size.

The shape of a Gaussian can be spherical or non-spherical. It is determined by the covariance matrix of the model. If identity matrix is used the Gaussian cloud is spherical. If a positive definite matrix is supplied, the resulting Gaussian is non-spherical.

The volume of a Gaussian is controlled by the largest eigenvalue of the covariance matrix. In our study the volume comes from a fixed range of positive integers.

The size of a Gaussian is the number of points in the cloud while the volume of the Gaussian is the space it takes up. The size of each cluster for our collection of clusters

containing 1000 points is controlled by the mixing weights of the mixture model. If clusters have the same size, the mixing weights are fixed. When clusters are of different sizes, the mixing weights are generated uniformly randomly for different sized clusters.

We have presented the experimental results in Chapter 4. However for the sake of easy reading, we replicate the necessary tables here to illustrate the behaviors on which we are focusing.

Table 4.1 F-scores for Sum-of-Squares with Greedy and Min-max Cut with Greedy

Greedy				0.25	0.5	0.75	1.0	2.0
Sphere	Same Volume	Same Size	Sos	0.15±0.00	0.36±0.04	0.83±0.01	0.93±0.04	0.94±0.04
			Mcut	0.15±0.01	0.39±0.04	0.86±0.01	0.97±0.00	0.99±0.00
		Diff Size	Sos	0.15±0.01	0.41±0.03	0.79±0.03	0.90±0.04	0.92±0.02
			Mcut	0.16±0.01	0.42±0.03	0.69±0.05	0.73±0.05	0.79±0.04
	Diff Volume	Same Size	Sos	0.20±0.03	0.57±0.13	0.86±0.04	0.93±0.04	0.93±0.04
			Mcut	0.22±0.05	0.63±0.12	0.91±0.02	0.98±0.01	0.99±0.00
		Diff Size	Sos	0.24±0.05	0.59±0.11	0.82±0.06	0.91±0.04	0.92±0.05
			Mcut	0.24±0.05	0.57±0.07	0.72±0.05	0.74±0.05	0.78±0.05
Non-sphere	Same Volume	Same Size	Sos	0.70±0.05	0.94±0.04	0.91±0.07	0.90±0.04	0.89±0.05
			Mcut	0.73±0.04	0.99±0.00	0.99±0.01	0.96±0.02	0.91±0.03
		Diff Size	Sos	0.67±0.05	0.88±0.05	0.87±0.06	0.89±0.05	0.87±0.06
			Mcut	0.68±0.04	0.76±0.05	0.78±0.04	0.79±0.04	0.79±0.06
	Diff Volume	Same Size	Sos	0.75±0.09	0.92±0.06	0.88±0.09	0.92±0.04	0.88±0.06
			Mcut	0.84±0.06	0.99±0.00	0.98±0.02	0.91±0.03	0.88±0.04
		Diff Size	Sos	0.73±0.09	0.87±0.06	0.89±0.05	0.88±0.04	0.88±0.06
			Mcut	0.70±0.05	0.76±0.04	0.79±0.04	0.78±0.05	0.81±0.06

Table 4.2 Percentage of Win for Greedy

	SOS	MCUT
Spherical	43%	57%
Non-spherical	59%	41%
Same volume	49%	51%
Diff volume	53%	47%
Same size	19%	81%
Diff size	83%	17%

5.2 The Size Factor

From Table 4.1 and Table 4.2 it is clear that for different sized clusters the min-max cut objective does not do a good job in terms of the f-score measure. To illustrate the behaviors of the objectives in respond to the size factor, we divide all synthetic datasets into two groups: datasets with same-sized clusters and those with different-sized clusters. The datasets in these two groups can be matched in pairs: a pair of datasets shares the same remaining data characteristic factors (shape and volume), in the same round and the same separation. There is an f-score associated with each dataset. Therefore for an algorithm, there are 200 pairs of f-scores. The contrast between a pair of f-scores illustrates the different behavior of the algorithm in respond to the size factor. The pairs are sorted in ascending order of the f-scores for the same-sized cluster. We plot the sorted pairs of f-scores for the algorithm min-max cut with greedy in Figure 5.1.

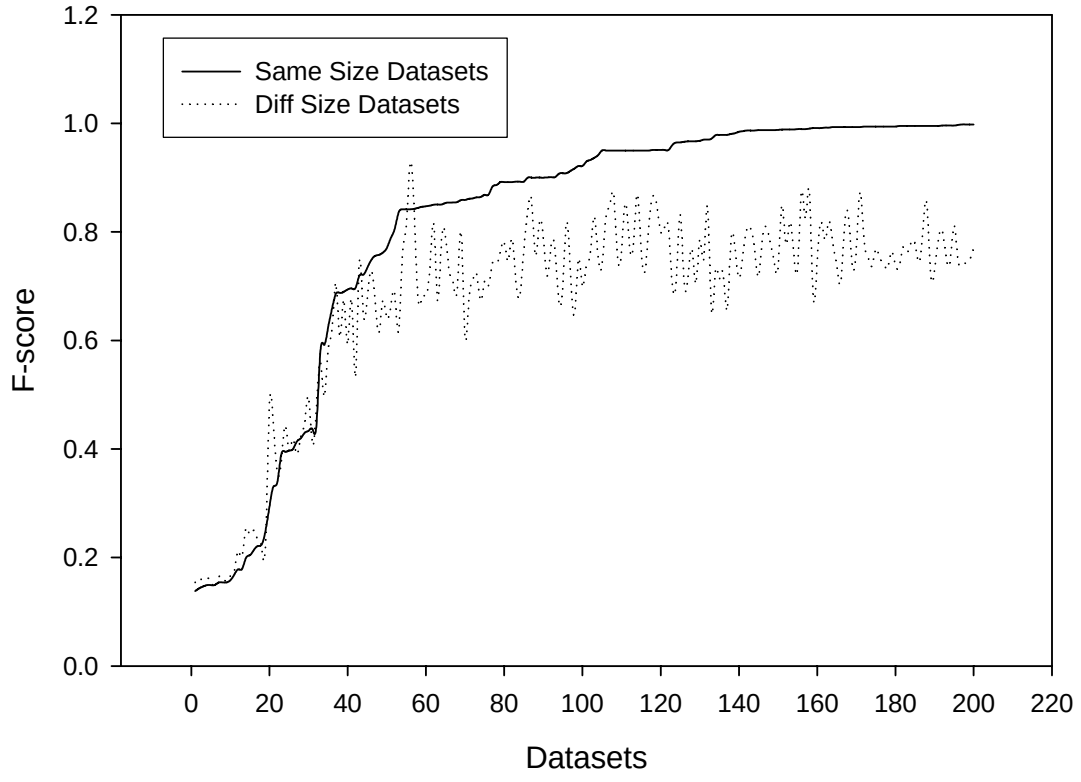


Figure 5.1 Contrast Between Same and Diff Sized Clusters for MCUT with Grd

We observe in Figure 5.1 the min-max cut greedy algorithm produces better results 80% of the time on same-sized datasets. The f-scores for same-sized clusters, running the min-max cut with greedy algorithm, can reach close to 1. In contrast, the f-scores for different-sized clusters, for the same algorithm, stay around 0.8 in the end.

For purpose of comparison, we show in Figure 5.2 the same plot, but with the algorithm sum-of-squares with greedy. From Figure 5.2, we observe there is not consistent superiority of better quality clusters for same sized datasets over different sized datasets. The f-scores for different sized datasets fluctuate around the curve for same sized

datasets. At the end of the curve, for roughly 20% of all datasets, the f-scores for same-size datasets approach the perfect score while the f-scores for different-sized classes lag behind.

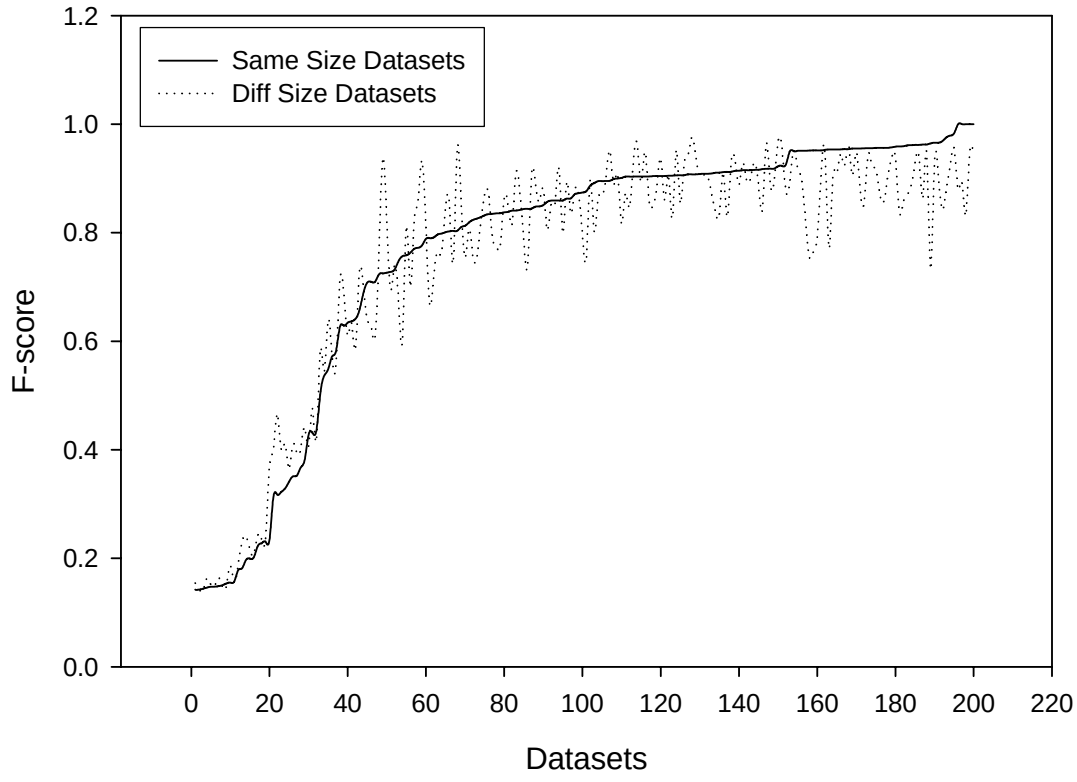


Figure 5.2 Contrast Between Same and Different Size Datasets, SOS w/ Grd

In Figure 5.3 we show the same plot for the contrasting algorithmic behavior between same and different sized datasets for the min-max cut with Kernighan-Lin algorithm. Correspondingly, the plot for the algorithm sum-of-squares with Kernighan-Lin is shown in Figure 5.4.

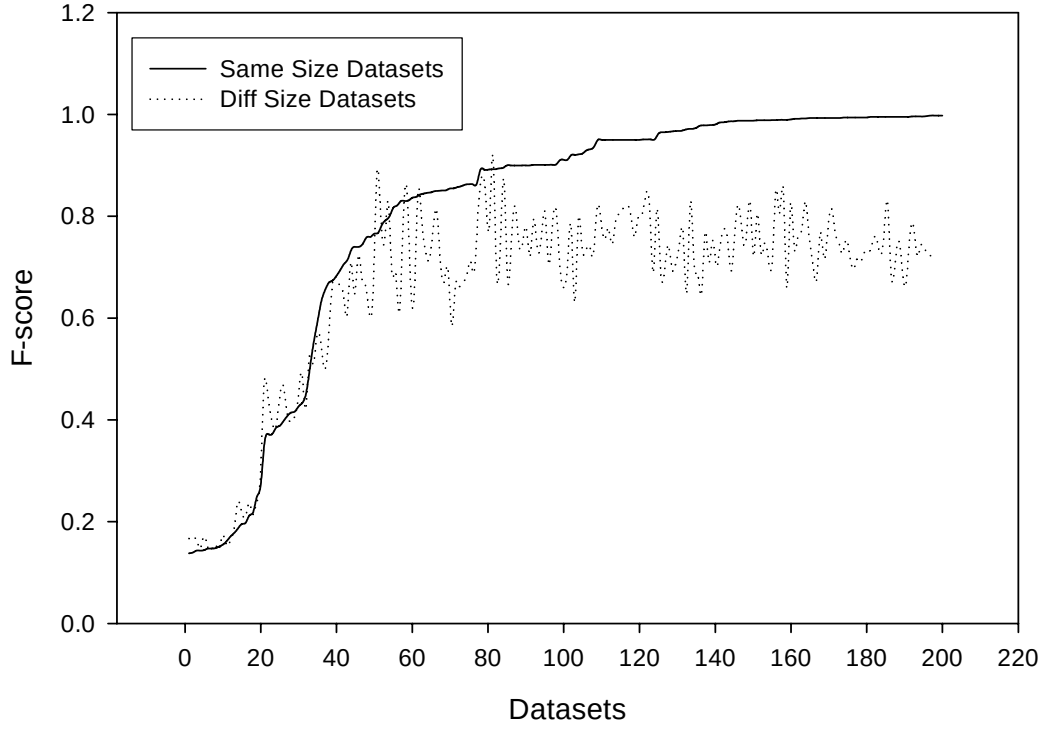


Figure 5.3 Contrast Between Same and Different Sized Datasets, MCUT w/ KL

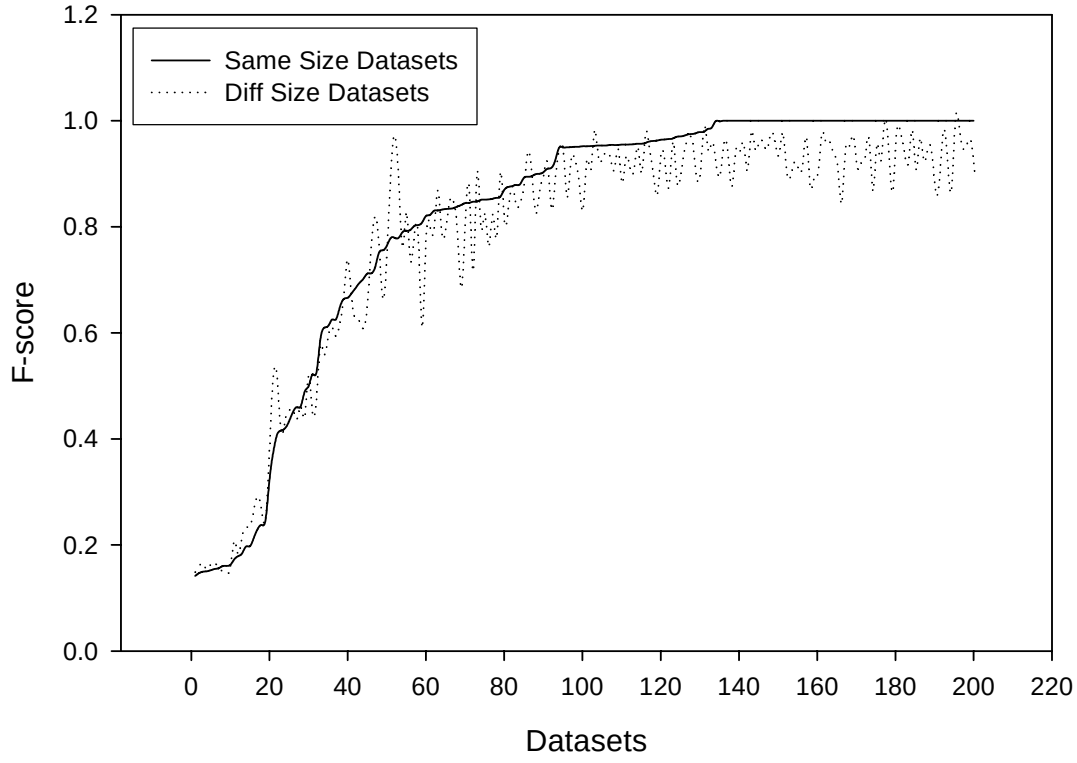


Figure 5.4 Contrast Between Same and Different Sized Datasets, SOS w/ KL

The curve in Figure 5.3 looks almost identical with that in Figure 5.1, because substituting Kernighan-Lin does not result in much difference compared with using greedy with min-max cut. For the algorithm sum-of-squares with Kernighan-Lin, there are around 70 datasets, an increase from Figure 5.2, for which f-scores for different sized datasets are lower than those for same sized datasets. However, the gap is not as large as the gap between f-scores for min-max cut on same and different sized datasets.

It is clear from the previous four figures that the difference in the behavior of the objective min-max cut is more extreme in responding to the class sizes. In this section we analyze the reasons why this is happening.

We conduct the following experiment to further understand the behaviors of these two objective functions. For each generated data set, we calculate the values of both objectives for the underlying classes and designate this value the class value. We compare the objective values that the optimization approaches achieve with the class values for the same data set. In Figure 5.5 and Figure 5.6 we compare the class values and the optimized values for min-max cut, plotted in ascending order of class values. We report curves of min-max cut for same-sized and different-sized classes in separate figures to highlight that the behaviors are quite different. In these two figures we only plot the comparison between greedy optimization values and the class values.

We make the following conclusions from our data. For same-sized classes, the optimization values for min-max cut under greedy are close to the class values. Even though producing the same objective values as the class values does not necessarily lead to the perfect f-score, the f-scores for these same-sized datasets are relatively good, indicating the underlying class structure has been recovered well, which in turn means the definition of the objective min-max cut captures the essential properties of the underlying structures.

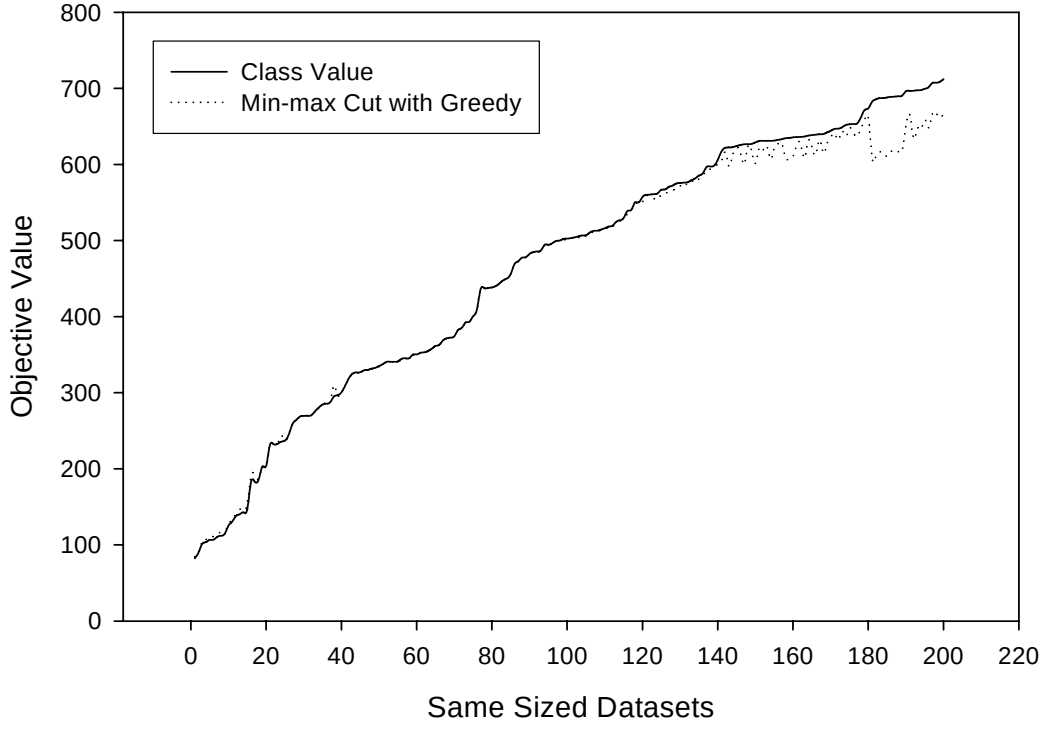


Figure 5.5 Greedy vs Class Values, MCUT, Same-sized classes

For different-sized classes the class values are in fact higher than and far from the optimization values. This indicates even when the optimization approach does a very good job to minimize the min-max cut and produces ending objective values well below the class value, the resulting clusters are not close to the labeled classes, reflected by the poor f-scores. Therefore under these situations optimizing min-max cut cannot produce the original structure of the dataset because the better the ending objective value, the further away from the class values. This is strong evidence that min-max cut is not a

suitable objective function to be chosen to capture the properties of the class structure of data with different-sized classes.

We did not show figures for sum-of-squares because there is no behavior difference for same-sized versus different-sized classes. Most of the optimization values for sum-of-squares are larger than their corresponding class values. However there are datasets whose class values are higher than optimization values, suggesting the objective sum-of-squares occasionally does not capture the class structures either.

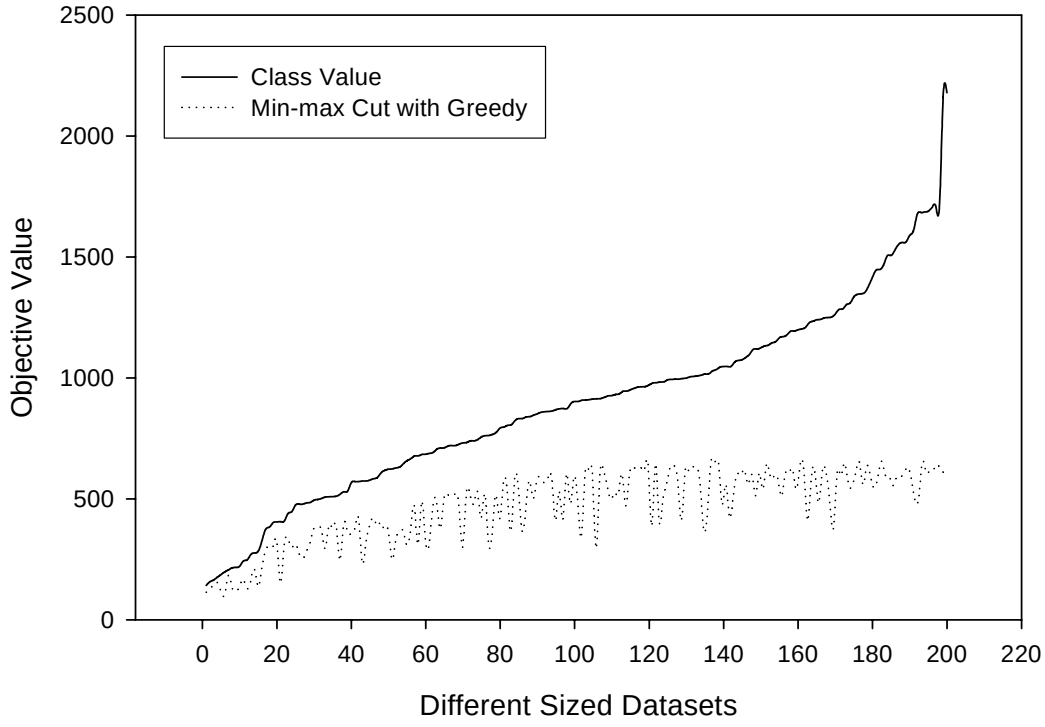


Figure 5.6 Greedy vs Class Values, MCUT, Different-sized Classes

5.3 The Shape Factor

We would like to study how the two objectives respond to the shape of classes in this section. We could compare the winning percentages of the two objectives in the two categories of synthetic datasets: spherical and non-spherical. However, the datasets in both categories include both same-sized and different-sized classes. The objective min-max cut does so poorly on the datasets that have different-sized classes that it loses no matter what the shapes of the classes are. Therefore we focus on the datasets with same-sized classes to calculate the winning percentages of both objectives and list the results using the greedy optimization approach in Table 5.1.

Table 5.1 Percentage of Win, Grd, Same-size Classes

	SOS	MCUT
Spherical	8%	92%
Non-spherical	30%	70%
Same volume	18%	82%
Diff volume	20%	80%

Under the greedy approach, the objective min-max cut wins with large margins. However, looking into the margin numbers, we realize min-max cut actually has its least winning margin in the category of datasets with non-spherical classes. This is an interesting observation. The minimum-cut based objectives are favored because they are believed to be shape independent. If min-max cut does not favor any shapes, it should give comparable performance in both spherical and non-spherical categories, if the rest of the factors are the same. If sum-of-squares were to produce clusters of comparable

quality in both spherical and non-spherical datasets, the winning margin for min-max cut for the two categories of spherical and non-spherical would be roughly the same. Moreover, the sum-of-squares objective is believed to favor spherical shapes and therefore we would expect it produces clusters with lesser quality for non-spherical datasets, comparing with the quality of clusters for spherical datasets. If the claim is true, the winning margin for min-max cut in the non-spherical category should actually go up. Therefore it is intriguing why the min-max cut would have the least winning margin in the category of non-spherical datasets.

This leads us to suspect some aforementioned beliefs may not be true. We further investigate the responses of the objectives to the shape factor by directly comparing performance on spherical versus non-spherical datasets in similar fashion to our comparisons for the size factor in last section.

In short, we divide the synthetic datasets into two groups: those with spherical classes and those with non-spherical classes. We match one dataset from a group with one from the other group. The pair of datasets has the same data characteristics except for the shapes of their classes. An f-score is associated with each dataset. We sort the pairs of f-scores according to ascending order of the f-score of the spherical dataset in the pair. We plot the pairs of curves for all the four algorithms in Figure 5.7 to Figure 5.10.

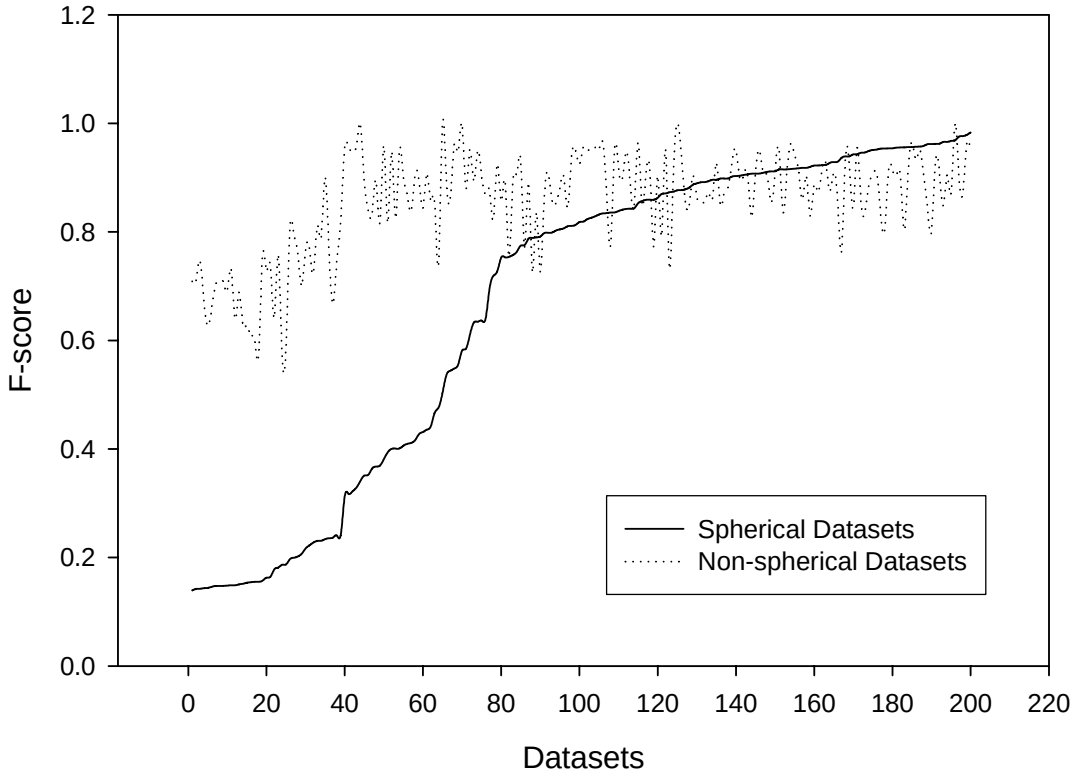


Figure 5.7 Contrast between Spherical and Nonspherical Datasets, SOS with Greedy

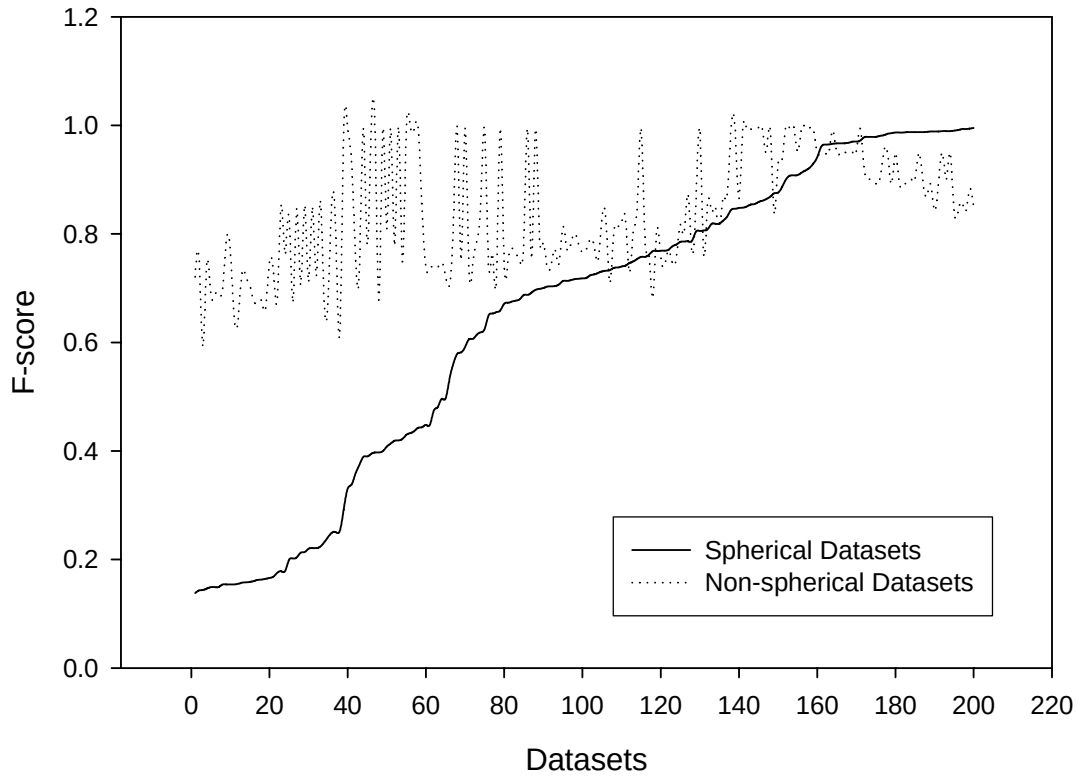


Figure 5.8 Contrast between Spherical and Non-spherical Datasets, MCUT with Greedy

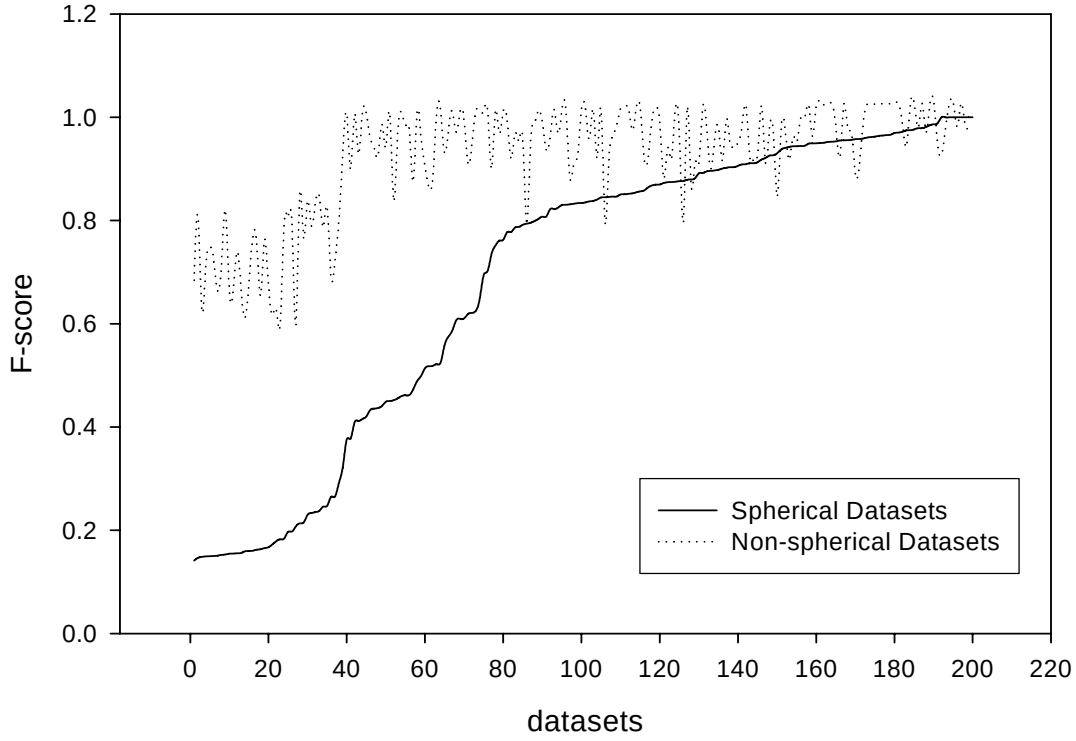


Figure 5.9 Contrast between Spherical and Non-spherical Datasets, SOS with KL

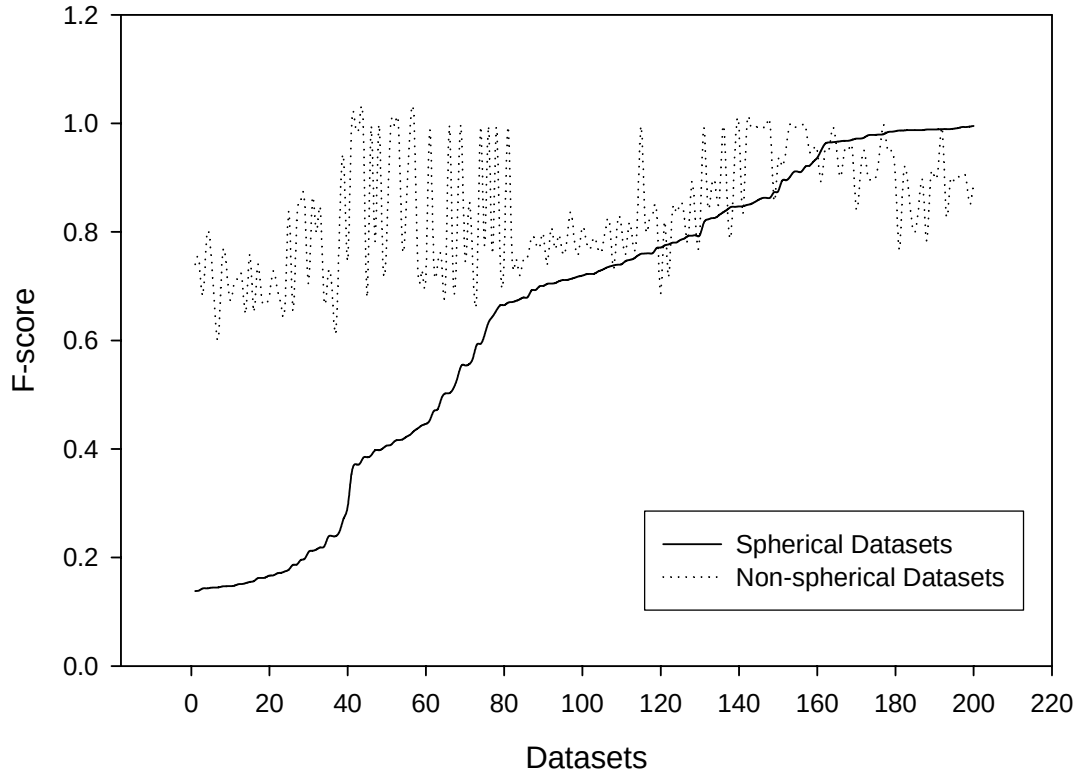


Figure 5.10 Contrast between Spherical and Non-spherical Datasets, MCUT with KL

Note in all four figures, Figure 5.7 to Figure 5.10, initially the f-scores for non-spherical datasets are much better than those for spherical datasets. The reason is that the definition of separation we use from theoretical literature is not reflecting the same level of clustering difficulty for spherical and non-spherical classes. We discuss this problem in detail in the next section.

After the initial superiority, the f-scores for non-spherical datasets drop below those for spherical datasets in all four figures. We cannot discern any peculiarities in the algorithmic behaviors of the two objectives in response to the shape factor for this

experiment, except in Figure 5.9, where the discrepancy between f-scores for spherical and non-spherical datasets is smaller at the end of the curves than those in the other three figures. All in all, non-spherical classes are harder to optimize than spherical classes. It is true that the sum-of-squares objective produces lesser quality clusters when classes are of non-spherical shapes. However, it is also true for the min-max cut objective. Therefore we cannot conclude that min-max cut is a shape-independent objective as claimed in previous literature.

In Table 5.2, we count the number of times that the f-score for the spherical dataset is higher than the f-score for the non-spherical datasets. The first row gives the number for all the 200 pairs of datasets, while the other five rows give the breakdown of the counts for each separation. There are 40 pairs of datasets in each separation.

Table 5.2 Number of Times F-Score for Spherical Datasets Larger Than F-Score for Non-spherical Dataset

	SOS+Grd	SOS+KL	MCUT+Grd	MCUT+Kl
All 200 Pairs of Datasets	68/200	33/200	50/200	45/200
Separation 0.25	0	0	0	0
Separation 0.5	0	0	0	0
Separation 0.75	8/40	1/40	0	0
Separation 1.00	31/40	9/40	20/40	19/40
Separation 2.00	29/40	23/40	30/40	26/40

Looking at the times when f-scores for spherical datasets are higher for the objective min-max cut, we observe that the number of times increases from separation 1.0 to separation 2.0. In fact, both algorithms, min-max cut with greedy and min-max cut with Kernighan-Lin, have higher f-scores for spherical datasets for roughly half of all the

datasets in separation 1.0. However, the number of higher f-scores for spherical datasets increases to roughly three quarters of all the datasets in separation 2.0. This indicates the objective min-max cut handles spherical datasets better than non-spherical ones for the well-separated situations. Therefore, our study does not support the shape-independency claim of the min-max cut objective.

The algorithm sum-of-squares with greedy, that is, k-means, does give better f-scores for spherical datasets, at the roughly three-quarter of times for both separation 1.0 and 2.0. This conforms to the criticism for the k-means algorithm. However, for separation 1.0, the algorithm sum-of-squares with Kernighan-Lin gives better performance on non-spherical datasets than spherical ones. This is in contrast to the belief that sum-of-squares favors spherical shapes. In contrast, it implies given good optimization approach, the objective sum-of-squares can work with both spherical and non-spherical shapes of clusters.

5.4 Alternative Definition of Separation

In Table 4.1 we note that the f-scores for the non-spherical cases are higher than those of the spherical cases. This is due to the fact that this particular definition of separation does not reflect the same level of difficulty for clustering for spherical and non-spherical cases. This happens because the separation is defined to be the ratio of distance between two centers and the longest radius of the two ellipsoids. In fact given the same separation, two spherical clusters with the same radius are overlapping while

two non-spherical clusters could be well separated, as illustrated in Figure 5.11. This explains the jump in f-scores for the non-spherical cases and also the smaller jump for the different volume cases in Table 4.1.

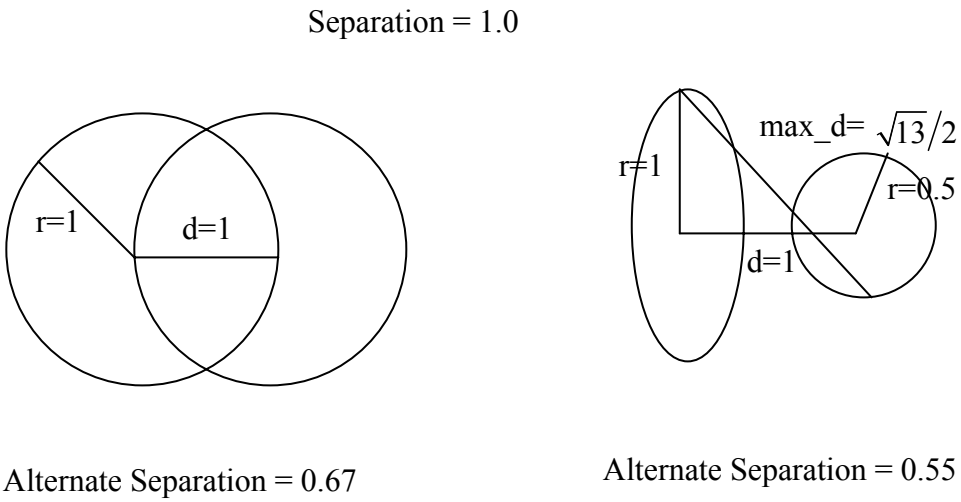


Figure 5.11 Same Separations for Spherical and Non-spherical Cases

We introduce Alternate Separation as a different, potentially more accurate indicator of clustering difficulty. Alternate Separation takes into account the maximum diameter of the input data set.

Definition: Given one cluster of d-dimensional points S. Let D(S) denote the diameter of S, which is the maximum distance between any two points in S. Let $S_1 \cup S_2$ represent the union of two clusters S1 and S2. The Alternate Separation for the two clusters is

$$\frac{\min(D(S_1), D(S_2))}{D(S_1 \cup S_2)}$$

The Alternate Separation of k clusters is the maximum pair-wise Alternate Separation of all possible pairs.

Note in contrast to the separation previously defined, the smaller an Alternate Separation is, the more separated the two clusters are. The Alternate Separation has a range from zero to one, while the previous definition of separation ranges from zero to positive infinity.

We define the Alternate Separation to see if it can help represent the same level of difficulty for both spherical and non-spherical clusters. In Table 5.3 we summarize the average f-scores and their standard deviations for spherical and non-spherical synthetic datasets. Observe the difference in f-scores for spherical and non-spherical datasets on the four iterative algorithms in separation 0.25 and 0.5. There are smaller gaps in f-scores in separation 0.75 too. For larger separations of 1.0 and 2.0, the gap disappears into small variations.

Table 5.3 Average F-scores and Standard Deviations for Spherical and Non-spherical datasets Using Old Separation

Separation	Category	# of datasets	SOS/Grd	SOS/KL	MCUT/Grd	MCUT/KL
0.25	Spherical	40	0.18±0.05	0.20±0.05	0.19±0.05	0.18±0.04
	Non-spherical	40	0.71±0.08	0.74±0.08	0.74±0.08	0.73±0.08
0.5	Spherical	40	0.48±0.13	0.54±0.11	0.50±0.12	0.49±0.11
	Non-spherical	40	0.90±0.06	0.96±0.04	0.88±0.12	0.88±0.12
0.75	Spherical	40	0.82±0.05	0.84±0.04	0.80±0.10	0.80±0.10
	Non-spherical	40	0.89±0.07	0.96±0.04	0.89±0.10	0.88±0.11
1.0	Spherical	40	0.92±0.04	0.94±0.04	0.86±0.12	0.86±0.12
	Non-spherical	40	0.90±0.04	0.95±0.05	0.86±0.09	0.87±0.07
2.0	Spherical	40	0.92±0.04	0.94±0.05	0.89±0.11	0.89±0.11
	Non-spherical	40	0.88±0.06	0.93±0.05	0.84±0.07	0.84±0.06

Alternatively, we calculate the Alternate Separation for each of the synthetic datasets and regroup these datasets by the Alternate Separation. We find the minimum and maximum Alternate Separation for these datasets and divide the range equally into five buckets. Every dataset is assigned to one of the buckets. For each bucket, we separate the datasets into spherical and non-spherical datasets and calculate the average and standard deviation of the f-scores for the two subgroups. In Table 5.4 we give the statistics for the four algorithms respectively. In this table each bucket corresponds to a certain range of Alternate Separation values, with Bucket 0 corresponds to the smallest range. Note in the Alternate Separation definition, the smaller the value, the more separated the clusters are. We observe for the first four buckets the gap in f-scores for

spherical and non-spherical datasets is small, roughly the same level as the gap observed in Table 5.3 for larger separations. The last bucket, which represents those datasets that are not well separated, displays large standard deviations.

Table 5.4 Average and Standard Deviation for Spherical and Non-spherical Subgroups within a Alternate Separation Bucket

Alt. Sep.	Category	# of datasets	SOS /Grd	SOS /KL	MCUT /Grd	MCUT /KL
Bucket 0 [0.34, 0.47]	Spherical	0	n/a	n/a	n/a	n/a
	Non-spherical	38	0.88±0.06	0.93±0.05	0.84±0.07	0.84±0.07
Bucket 1 [0.47, 0.60]	Spherical	2	0.92±0.02	0.94±0.01	0.85±0.03	0.86±0.01
	Non-spherical	7	0.89±0.05	0.97±0.04	0.88±0.07	0.86±0.07
Bucket 2 [0.60, 0.73]	Spherical	38	0.92±0.04	0.94±0.05	0.89±0.11	0.89±0.11
	Non-spherical	48	0.90±0.05	0.95±0.05	0.87±0.09	0.87±0.08
Bucket 3 [0.73, 0.87]	Spherical	23	0.91±0.05	0.92±0.05	0.82±0.12	0.83±0.12
	Non-spherical	54	0.89±0.06	0.96±0.05	0.87±0.11	0.87±0.12
Bucket 4 [0.87, 1.00]	Spherical	137	0.55±0.29	0.58±0.29	0.55±0.28	0.54±0.29
	Non-spherical	53	0.76±0.11	0.79±0.12	0.78±0.11	0.77±0.11

Next we take the last bucket from Table 5.4, which consists of 190 datasets out of 400 and further divide these files into five buckets according to a finer level of distinction among Alternate Separations of the datasets. The statistics are presented in Table 5.5.

Table 5.5 Average and Standard Deviation for Spherical and Non-spherical Datasets for Last Bucket in Table 5.4

	Category	# of datasets	SOS/Grd	SOS/KL	MCUT/Grd	MCUT/KL
Bucket 0	Spherical	23	0.88/0.07	0.90/0.06	0.84/0.12	0.84/0.12
	Non-spherical	7	0.95/0.04	0.98/0.02	0.93/0.11	0.93/0.10
Bucket 1	Spherical	21	0.83/0.11	0.84/0.10	0.80/0.14	0.80/0.14
	Non-spherical	10	0.81/0.11	0.97/0.04	0.84/0.12	0.78/0.12
Bucket 2	Spherical	23	0.71/0.15	0.74/0.13	0.73/0.15	0.72/0.16
	Non-spherical	23	0.73/0.08	0.75/0.10	0.76/0.09	0.76/0.09
Bucket 3	Spherical	21	0.49/0.18	0.53/0.16	0.49/0.15	0.48/0.14
	Non-spherical	13	0.68/0.05	0.72/0.05	0.72/0.06	0.72/0.06
Bucket 4	Spherical	49	0.23/0.10	0.25/0.12	0.24/0.10	0.23/0.10
	Non-spherical	0	N/a	n/a	n/a	n/a

In Table 5.5, the standard deviation of datasets in the same bucket is smaller than in Table 5.4, for spherical and non-spherical datasets respectively. Observe the first four buckets in Table 5.4 have relatively small standard deviation, with the last bucket, containing the most number of datasets, exhibiting large variances. When further dividing the last bucket, the standard deviation does become smaller. This leads us to conclude that the Alternate Separation has the effects of compressing the lower end and stretching the higher end of the old definition of separation.

The last bucket in Table 5.4 contains the bulk of separation 0.25 and 0.5 datasets. The rest of the datasets, which result in relatively concentrated f-scores, are distributed across the remaining Alternate Separation buckets. As a result, in four out of the five Alternate Separation buckets there are small gaps in contrast to only two out of five older separation buckets. This illustrates that changing the definition of separation can help smooth the f-score difference between spherical and non-spherical datasets.

The incentive to devise a new definition of separation is that under the old definition the non-spherical datasets have higher f-scores than spherical datasets when clusters are overlapping. With this new definition, we now revisit the discussion on the shape factor for the two objectives. We compare the average values for algorithms and objectives in Table 5.3, instead of the pair-wise counting in Table 5.2. The observations focus on the responses from two objectives to the shape factor.

The columns on sum-of-squares in Table 5.4 shows that except in the last bucket, f-scores for the objective sum-of-squares on spherical datasets are higher than non-spherical ones for the greedy approach. However, f-scores for non-spherical datasets are higher than spherical ones for the Kernighan-Lin approach. This supports the reputation of k-means (and our findings in Section 5.3) that it favors spherical shapes. Moreover, the same objective optimized by the Kernighan-Lin optimization approach reveals that the objective sum-of-squares can work well with non-spherical classes, which is observed in Section 5.3, too.

Looking at the objective min-max cut for Bucket 1 and 2, we observe for both optimization approaches, the f-scores for spherical datasets are for the most part no less than those for non-spherical datasets. This illustrates the point that min-max cut works relatively better with spherical shapes for larger separations in comparison to sum-of-squares.

However, if we compare the f-scores in Table 5.3 in alternate columns, the f-scores in the column of algorithms using sum-of-squares are higher than those in the column using min-max cut in both non-spherical and spherical datasets. This is in contrast with

the claim when algorithms using min-max cut are developed that they work better than the sum-of-squares. In fact, it supports our conclusion that k-means can produce clusters with fairly good quality.

This definition of Alternate Separation gives a more consistent indicator of clustering difficulty across spherical and non-spherical clusters for more separated datasets, at the cost of obliterating the differences in clustering difficulty for very hard situations. There could be other definitions. It would be nice to have both consistency across different shapes and high distinguishing powers in lower ends in a single definition. If such a definition exists, it would be a better choice for future theoretical and experimental analysis.

Chapter 6 Spectral Clustering Algorithms

In this chapter, we introduce two spectral clustering algorithms. The first one is called Spectral Cut algorithm, which is developed in computer vision to help image segmentation. We compare the algorithm with the k-means algorithm from previous chapter and show the reason that the Spectral Cut is not as good.

The second algorithm is called Cookie Cutter, which is a practical version of an existing algorithm specified for a provable guarantee of correctness. We modify the original algorithm so that it can work with real-world datasets. We show by systematic evaluation that Cookie Cutter can find the perfect solution when classes are well separated, inn contrast to the iterative clustering algorithms such as k-means, which tends to be trapped in a neighborhood of the initial seeds.

Moreover, we examine the role of the singular value decomposition (SVD) used in the middle of Cookie Cutter. The algorithm has two main components: SVD and distance-based clustering. We conclude that SVD makes learning of mixture of Gaussians more efficient to achieve a provable approximation. However, we are not gaining much quality improvement from it. Instead, it makes Cookie Cutter runs in a speed that is not

acceptable in practice. Removing the SVD from the algorithm and applying the distance-based clustering directly gives all the benefits of Cookie Cutter without the slow speed.

6.1 Spectral Cut Algorithm

The first spectral algorithm we consider seeks to optimize a minimum-cut based objective by spectral methods. It is heavily cited in the recent literature and shows evidence of good performance in the area of image segmentation. We apply the algorithm with minor modifications to the synthetic datasets and compare its performance with the iterative clustering algorithms in previous chapters. In this section we describe the algorithm, including its background and implementation details. The original algorithm can be found in this paper [96].

Given a set of points S in R^d, we can treat points as nodes V in a graph and label an edge between two nodes with the similarity value as its weight. The problem of finding clusters is then equivalent to find partitions in the graph $G(V, E, W)$. The Spectral Cut algorithm seeks to optimize the objective of normalized cut, described and compared to min-max cut in Chapter 4, by partitioning the graph G using the second smallest eigenvector of a generalized eigenvalue system, which is based on W.

By choosing to optimize the normalized cut, the algorithm can simultaneously optimize both the within-cluster associations and the separations between clusters. The algorithms we studied in the previous chapters start with a so-so solution and try to

improve the solution by iterative refinement. In contrast, the Spectral Cut algorithm takes a top-down approach: starting from the whole graph, using the second smallest eigenvector to partition the graph into two parts and recursively partition the resulting two sub-graphs until a stopping condition is satisfied.

Note that the top down approach produces a hierarchy of the input dataset. The root node of a hierarchy contains all data points. Every time a partition is found, two resulting sets of data points are the two children of the partitioned node. The final partition consists of the leaves of the hierarchy. In some area of research, such as psychology, a hierarchy is the goal of clustering [12, 34, 40]. However, we study the final partition to investigate the cluster quality, and do not utilize the hierarchy that comes with the graph partition algorithms.

6.1.1 Original Algorithm

Given a set of points S in R^d, the points can be viewed as nodes V in a graph G. We can associate a weight $w(i, j)$ with every edge. The set of edges is represented by E. The weights are recorded in a $n \times n$ matrix W. A graph $G=(V, E)$ can be partitioned into two disjoint sets A and B: $A \cup B = V, A \cap B = \varnothing$. The normalized cut is defined as

$$Ncut(A, B) = \frac{\sum_{i \in A, j \in B} w(i, j)}{\sum_{i \in A, j \in V} w(i, j)} + \frac{\sum_{i \in A, j \in B} w(i, j)}{\sum_{i \in B, j \in V} w(i, j)}$$

We define another measure for total normalized association within groups for a given partition

$$Nassoc(A,B) = \frac{\sum_{i,j\in A} w(i,j)}{\sum_{i\in A, j\in V} w(i,j)} + \frac{\sum_{i,j\in B} w(i,j)}{\sum_{i\in B, j\in V} w(i,j)}$$

The normalized cut describes the degree of similarity between the two groups of the partition, while the normalized association reflects how tightly on average nodes within a group are connected. The difference between normalized cut and min-max cut lies at the denominators in the definition. The min-max cut uses the total weights within a group to normalize the weights across the cut. On the contrary in normalized cut, the total weights from nodes in a group to every node in the graph are used. The advantage of defining the normalized cut this way is that it relates to the normalized association naturally. In fact,

$$Ncut(A,B) = 2 - Nassoc(A,B)$$

Therefore, minimizing the across-group similarity and maximizing the within-group association can be satisfied simultaneously. An algorithm that optimizes the normalized cut can optimize the normalized association at the same time.

In Chapter 4 we compared the objective of the normalized cut with the min-max cut using hill-climbing algorithms and concluded the min-max cut yields better results. In this chapter we optimize the normalized cut using spectral techniques instead of the greedy approach studied in the previous chapters. Changing to another optimization approach may help to give better results even though we use an inferior objective: the normalized cut.

Given a partition of a graph *G* into two sets of nodes *A* and *B*, we can define an indicator *n*-dimension vector $\vec{x} : x_i = 1$ if node *i* is in *A* and $x_i = -1$ otherwise.

Let $\vec{d}$ represent the total weights from each node to all other nodes.

$$d_i = \sum_j w(i, j)$$

Let *D* be a $n \times n$ diagonal matrix with $\vec{d}$ on its diagonal. Let *W* be $n \times n$ symmetrical matrix with *w(i,j)* being the weight between node *i* and node *j*. The problem to solve is given *D* and *W*, find the indicator vector $\vec{x}$ to minimize $Ncut(\vec{x})$.

It can be shown that [96]

$$\min_x Ncut(\vec{x}) = \min_y \frac{\vec{y}^T (D - W)\vec{y}}{\vec{y}^T D\vec{y}}$$

with the condition $y_i \in \{1, -b\}$, $b = \frac{\sum_{x_i > 0} d_i}{\sum_{x_i < 0} d_i}$ and $\vec{y}^T D\vec{1} = 0$, $\vec{1} = [1\ 1\ 1 \ldots 1]^T$. The above right hand side is the expression of Rayleigh Quotient. If $\vec{y}$ is relaxed to take on real values, solving the right hand side is equivalent to solve the following generalized eigenvalue system

$$(D - W)\vec{y} = \lambda D\vec{y}$$

The smallest eigenvalue of the system is zero and the corresponding eigenvector is $\vec{1}$. Therefore the eigenvector corresponding to the second smallest eigenvalue is the real valued solution to the normalized cut problem. We call this eigenvector the second

smallest eigenvector from this point on. The above generalized eigenvalue system can be transformed into a standard one.

$$D^{-1/2}(D-W)D^{-1/2}\vec{z} = \lambda\vec{z}$$

where

$$\vec{z} = D^{1/2}\vec{y}$$

Before the relaxation, the sign of each component of y indicates to which partition the corresponding node belongs. However, by relaxing to take on real values, it is not easy to find the partition from the solution vector. It is not a basic sign test any more because there is a neighborhood of candidate positions around zero. If we sort the nodes in the order of their corresponding components in the second smallest eigenvector, nodes are partitioned based on their components being less than or greater than a neighboring dividing node. The heuristic solution is to check the nodes to see at which dividing node the normalized cut objective takes the minimum value. The partition happens at the node where minimum normalized cut value is achieved. Usually, it happens near the value zero. In the original paper, they checked every five nodes [96]. In our implementation, we calculate the normalized cut value for every node.

The algorithm starts from the whole graph and partition the graph into two parts. Then the two parts are recursively partitioned. The algorithm stops if a stopping condition is met. We modify the stopping condition of the original algorithm. Originally, the algorithm stops if the normalized cut value is below a threshold value. However, the number of clusters is hard to determine using this stopping method. Since we compare

this algorithm with the iterative ones, which are given the desired number of clusters, we control the Spectral Cut algorithm also by the number of partitions it creates, that is, the number of clusters it finds.

Specifically, we maintain a partition queue. Initially there is the whole graph as the only partition in the queue. We pop the head of the queue and bi-partition it into two parts. The two parts are then inserted at the tail of the queue. Once the number of partitions in the queue reaches our pre-specified number of clusters, the algorithm stops. We feel the modification brings easy control to the algorithm, while not affecting the quality of clusters it can find.

The pseudo-code for the Spectral Cut algorithm is as follows.

Empty the partition queue;
Enter the entire graph into the partition queue;
While number of partitions in queue not reach desired number
    Current graph is the head of the queue;
    Calculate the weight matrix W and the degree matrix D of the current graph;
    Calculate the second smallest eigenvector z of the
    matrix $D^{-1/2}(D-W)D^{-1/2}$ and transform z back to y;
    Sort the components of the vector y in ascending order;
    For every component
        Calculate the normalized cut value if splitting the nodes of the graph at this point
    End For

Record the position where the minimum normalized cut value is;

Split the nodes at that position: every node before that position in the sorted *y* belongs to one group, while every node after that position belongs to another group;

Enter the two sub-graphs into the partition queue;

End While

Clusters are in the partition queue.

6.1.2 Results on Synthetic Data

We implemented the algorithm in Matlab and therefore used the subroutines provided by Matlab to calculate the eigenvectors of a given matrix. We applied the algorithm to the synthetic datasets; the f-scores are listed in Table 6.1, together with the f-scores for k-means for the purpose of comparison. From Table 6.1 we observe the spectral cut algorithm cannot produce clusters matching in quality of those produced by k-means. Especially for the more separated cases, the spectral cut algorithm can only yield clusters with f-scores as high as 0.6, where the k-means can find clusters with f-score as high as 0.9.

Table 6.1 Average F-scores for K-means and Spectral Cut Algorithms on Synthetic Datasets

				0.25	0.5	0.75	1.0	2.0
Sphere	Same Vol	Same Size	k-means	0.15±0.00	0.36±0.04	0.83±0.01	0.93±0.04	0.94±0.04
			Spec-cut	0.14±0.01	0.25±0.01	0.40±0.02	0.47±0.02	0.60±0.05
		Diff Size	k-means	0.15±0.01	0.41±0.03	0.79±0.03	0.90±0.04	0.92±0.02
			Spec-cut	0.15±0.01	0.27±0.01	0.42±0.05	0.44±0.03	0.46±0.03
	Diff Vol	Same Size	k-means	0.20±0.03	0.57±0.13	0.86±0.04	0.93±0.04	0.93±0.04
			Spec-cut	0.17±0.02	0.33±0.06	0.43±0.07	0.47±0.06	0.52±0.10
		Diff Size	k-means	0.24±0.05	0.59±0.11	0.82±0.06	0.91±0.04	0.92±0.05
			Spec-cut	0.19±0.03	0.36±0.05	0.43±0.04	0.45±0.03	0.48±0.03
Non-sphere	Same Vol	Same Size	k-means	0.70±0.05	0.94±0.04	0.91±0.07	0.90±0.04	0.89±0.05
			Spec-cut	0.26±0.01	0.46±0.02	0.53±0.02	0.73±0.04	0.79±0.04
		Diff Size	k-means	0.67±0.05	0.88±0.05	0.87±0.06	0.89±0.05	0.87±0.06
			Spec-cut	0.27±0.02	0.47±0.02	0.51±0.05	0.58±0.06	0.51±0.08
	Diff Vol	Same Size	k-means	0.75±0.09	0.92±0.06	0.88±0.09	0.92±0.04	0.88±0.06
			Spec-cut	0.30±0.02	0.44±0.03	0.57±0.08	0.61±0.13	0.55±0.15
		Diff Size	k-means	0.73±0.09	0.87±0.06	0.89±0.05	0.88±0.04	0.88±0.06
			Spec-cut	0.37±0.04	0.50±0.04	0.57±0.06	0.56±0.10	0.58±0.10

One reason for the relatively poor performance of the Spectral Cut algorithm is the fluctuations of the components of the eigenvector. To illustrate the point, we show the second smallest eigenvector for the spherical, same-volume, same-size dataset with separation 2.0, with components' values plotted in sorted order.

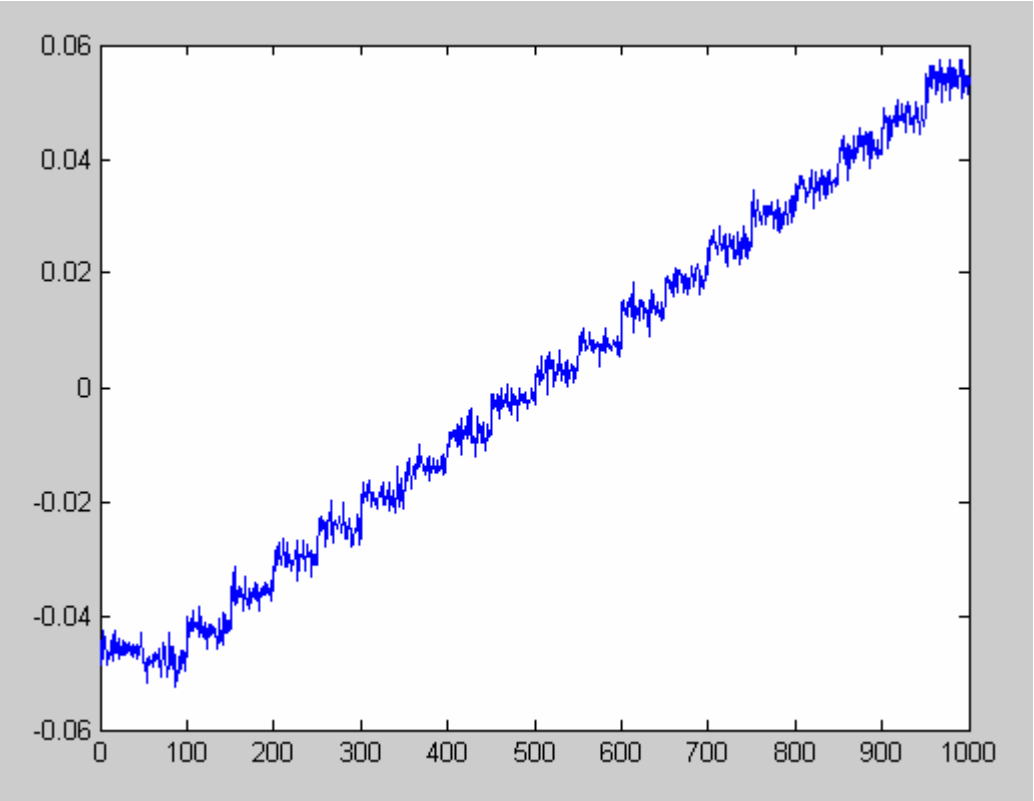


Figure 6.1 Second Smallest Eigenvector for a Synthetic Dataset

From Figure 6.1 we show the fluctuations of the components. For a human reader, it is relatively easy to discern the 20 clusters roughly by grouping components with similar values into segments. For a computer, it has to sort the components first and looking for a cutting point. Inside a single class, the component values can take a large range. For example, points from the two classes, whose component values are closest to zero in Figure 6.1, are easily mingled with each other. It is not unusual for a few points in a class to get cut off. The errors from each cut accumulate and lead to the relatively poor quality of the final clusters.

The original spectral cut algorithm shows promise in image segmentation. However, we observe the original paper only shows a few images to demonstrate the effects of

segmentation. It is probably hard to evaluate the segmentation because of its subjectivity.

6.2 Cookie Cutter Algorithm

The second spectral algorithm we choose to compare was recently developed to have provably good performance in a theoretical model. The original algorithm stands out because with high probability it finds the global optimum for the problem of learning a mixture of Gaussians, instead of a local optimum [109]. It improves the time complexity of a previous algorithm with a provable guarantee to find a globally optimal solution for the problem of learning a mixture of Gaussians [109]. We have to change some details so that it can be implemented and applied to real datasets. The modified version, which we call Cookie Cutter, to separate from the original theoretical version, contains heuristic steps. However, we believe the Cookie Cutter algorithm, as a heuristic, is doing the right thing in essence, because of the asymptotic guarantee for the original algorithm. This section focuses on the implementation and application of the algorithm.

6.2.1 Problem Description

We repeat the definition of a mixture of Gaussians from Chapter 3. A mixture of Gaussians N_1, N_2, … N_k with mixing weights w_1, w_2, … w_k , where $\sum_i w_i = 1$ and k is the number of Gaussians, is the distribution in which a sample is produced by first picking a

component Gaussian – the *i*th one is picked with probability w_i – and then producing a sample from that Gaussian. The problem of learning mixture of Gaussians is to reconstruct the component Gaussians and their mixing weights, given samples from the mixture distribution.

The density of a multivariate Gaussian distribution is

$$N_{\mu_i,\Sigma_i}(\vec{x}) = \frac{1}{(2\pi)^{d/2}\left|\Sigma_i\right|^{1/2}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^t \Sigma^{-1}(\vec{x}-\vec{\mu}_i)\right]$$

Let (w_1, N_1, w_2, N_2, …, w_k, N_k) be a mixture of Gaussians in R^d. One can associate k numbers $(N_i(x))_{i=1...k}$ with any point $x \in R^d$, corresponding to the probabilities of the point belonging to the component Gaussians. For any sample set $S \subseteq R^d$, this imposes a natural partition: each point $x \in S$ is labeled with an integer $l(x) \in \{1...k\}$ indicating the distribution that assigns the highest probability to x. Ties are broken arbitrarily. The likelihood of the sample set is

$$\prod_{x \in S} N_{l(x)}(x)$$

In other words, given a set S of n samples, each sample is described by a d-dimensional vector. The sample matrix M is defined as the $d \times n$ matrix, where each column is a sample. We like to group the columns of the sample matrix M into groups such that the likelihood of the set S is maximized.

A related concept is the sample covariance matrix. This matrix helps explain the technique Principle Component Analysis, which shows up in later sections of this chapter. For the sake of description continuity, we define the matrix here. Given a

sample set S of n vectors $\vec{x}_i,\ i \in [0..n-1]$, the sample mean is the vector $\vec{q} = \frac{1}{n}\sum_{i=1}^{n} \vec{x}_i$.

Define a matrix X , whose columns are the vectors $\vec{x}_i - \vec{q},\ i \in [0..n-1]$. The sample covariance matrix is the matrix $\frac{1}{n} X^T X$.

6.2.2 Original Algorithm

The original algorithm has two stages: (1) the projection stage; and (2) distance-based clustering stage. The projection stage projects the samples to the subspace defined by the top k singular vectors of the sample matrix *M*. A distance-based clustering algorithm is then applied to these projected samples. The projection can increase the distance between the centers of component Gaussians while making their shapes more spherical [109]. Therefore the task of clustering becomes easier in the projected space.

Distance-based clustering finds a cluster by starting from a point and grouping all points with distance smaller than a threshold into the same cluster. The point to start with and the radius to include are two parameters provided by the algorithm. We give the name "Cookie Cutter" because the algorithm uses a template to cut off clusters. At this point the template is a hyper-sphere because of the starting point and the fixed radius.

The idea works because distance-based clustering assumes the distribution of inter-cluster pair-wise distances is separated from intra-cluster distances. Therefore a gap exists between the largest intra-cluster distance and the smallest inter-cluster distance. If Gaussians in the mixture are well separated, probing for the radius may yield the perfect

solution. In fact the theoretical analysis assumes a large separation and proves the correctness of distance-based clustering.

Assume *S* is the sample set of *n* points, *d* is the dimension for each point and *k* is the number of underlying Gaussians. In the algorithm, δ is the confidence parameter while ε is the accuracy parameter. Assume the minimum mixing weight of the sample set is $w_{\min}$. The pseudo-code of the algorithm is as follows.

(1) Project original sample set to the rank *k* subspace spanned by the top *k* right singular vectors. We still call this set *S*.

(2) Let $R = \max_{x \in S} \min_{\substack{y \in S \\ y \neq x}} \|x - y\|$. Discard all points from *S* whose closest point lies at squared distance at most $3\bar{\varepsilon}R^2$, where $\bar{\varepsilon} = \frac{d-k}{w_{\min}} \cdot \varepsilon$. The remaining point forms the set S'.

(3) Let *x*, *w* be the two closest points in S', with $r = \|x - w\|^2$; Let *H* be the set of all points at squared distance at most $l = r(1 + 8\sqrt{\frac{6 \ln \frac{n}{\delta}}{k}})$ from *x*.

(4) Report H as a Gaussian, and remove the points in H from S'. Repeat Step (3) until there are no points left in S'.

(5) Output all Gaussians learned in Step (4) with variance greater than $3\bar{\varepsilon}R^2/k$.

(6) Remove the points from step (5) from the original sample set *S*, and repeat Step (1) to (5) on the rest.

In plain English, after the projection of the sample set to its best rank *k* subspace, the points that are close to each other are removed so that all the remaining points are from

Gaussians with large variance. Centered at the closest point pair, the algorithm sweeps away all points that are within the given radius and report it as a Gaussian candidate. The sweeping continues until no points are left. Then of all the Gaussian candidates found, those whose variances exceed a threshold are reported as Gaussians. The above process then is repeated on all the points not yet belonging to a reported Gaussian until all points are assigned to some Gaussian.

6.2.3 Modification

There are two constants in the algorithm. One is the threshold $3\bar{\varepsilon}R^2$ to remove those points that belong to Gaussians with smaller variances. The other is the fixed radius l the algorithm uses to gather a Gaussian candidate. The constant used to threshold candidate Gaussians is a function of the first constant. Vempala and Wang prove by calculating the constants by the formula they give that the algorithm can correctly identify all Gaussians within the original sample set given the accuracy and confidence parameters.

However the main problem is that the separation for Gaussians in their assumptions is too large to be practical. Given a common setting of clustering, such as 20 clusters, unit spherical Gaussians, the proof requires a sample complexity of more than 21 million points and the separation between clusters more than 300. Playing with accuracy and confidence parameters does not help much because the dominating terms have nothing to do with them. In fact, in calculating the sample complexity, there is a large constant

coefficient that multiplies 5000 with the ratio between the number of dimensions and the minimum mixing weight.

The same happens with the calculation of the aforementioned constants. Suppose we have 1000 points equally divided into 20 clusters. The points are distributed uniformly in a 2000-dimension space. Every point is normalized so the maximum distance between them is $\sqrt{2}$. Let the accuracy parameter ε be 0.05. In order for the threshold $3\bar{\varepsilon}R^2$ to be smaller than the maximum distance, the minimum pair-wise distance would be 0.0002. If the minimum pair-wise distance is larger than 0.0002, which is very likely in practice, all the points in the sample set would be removed in the first step of the algorithm. Therefore we have to find the constant empirically, instead of following their formula.

The second problem is that the algorithm does not specify the number of clusters it produces. In reality most applications want a fixed number of clusters or at least an algorithm should be capable of producing a desired number of clusters. Otherwise the application has to deal with whatever the clustering algorithm is able to produce.

We offer two modifications to adapt the algorithm to the real world. First of all we do not remove the points that are close to each other any more. In the original version, the points are removed so that the remaining points are from Gaussians with large variances. However, since we pick Gaussians with largest variances as the output, keeping those points that belong to small-variance clouds does not matter much. Practically speaking, we have one less parameter to tune for the algorithm. Secondly for any given *l*, a set of

clusters will be produced. The larger *l* is, the fewer clusters there are. We know the parameter *l* cannot be larger than the ratio of the maximum pair-wise distance to the minimum pair-wise distance. Thus we can binary-search the parameter space of *l* and return the clustering that has the closest number of clusters to the desired number.

The pseudo-code for the Cookie Cutter algorithm is as follows. The sample set *S* and the number of clusters *k* are given.

Clear the set of result cluster list *res_list*;

All points in the sample set *S*;

Set minimum radius *min_R* to minimum pair-wise distance in *S*;

Set maximum radius *max_R* to maximum pair-wise distance is *S*;

Set *R* to be the mid point between *min_R* and *max_R*;

While min_R < max_R

Clear the result cluster list *L*;

While sample set S not empty

Create a sample matrix *M* and calculate the top *k* singular vectors of *M*;

Project the points in *S* into the subspace defined by the top *k* singular vectors; call the set of projected points S_p;

Clear the list of Gaussian candidates *C*;

While S_p is not empty

Find the pair of points that are closest to each other in S_p;

Start from one point of the pair, calculate the distances from every other point to it; every point with a distance

smaller than *R* is included in the current Gaussian candidate *gc*;

Put *gc* into list *C*;

Remove points in *gc* from S_p;

End While

Calculate the variance for each Gaussian candidate in *C*;

The candidate with the largest variance is the reported Gaussian *g*;

Add *g* to list *L*;

Remove points in *g* from *S*;

End While

Add the result list *L* to the result list array *res_list*;

If the number of Gaussians in L is smaller than k

max_R=R;

R=(min_R+max_R)/2;

Else if the number of Gaussians in L is larger than k

min_R=R;

R=(min_R+max_R)/2;

Else if the number of Gaussians in L is equal to k

Return L as the result;

Break out of the outer while loop;

End If

End While

Search through *res_list*; Report the one list with the closest number of clusters to *k* as the final clustering result. In the case there are two lists whose number of

clusters is at the same distance to *k*, the one with larger number of clusters will be returned.

6.2.4 Results with Synthetic Datasets

We count a win for an algorithm on an input file if the f-score evaluation index has a higher value for that algorithm than the other algorithm on the same input file. Please refer to Chapter 3 for details of the methodology to compare two algorithms. We calculate the percentage that each algorithm wins in small separation files in Table 6.2. The average f-scores and their standard deviations for the ten files per case in smaller separations are shown in Table 6.3. Note in Table 6.3, Cookie Cutter actually wins more than 10% of all the table entries. The discrepancy in the percentage wins between counting by category and by file is due to the large standard deviations of the f-scores for k-means.

Table 6.2 Algorithm Comparison for Small Separations

	k-means	Cookie cutter
Win	90%	10%

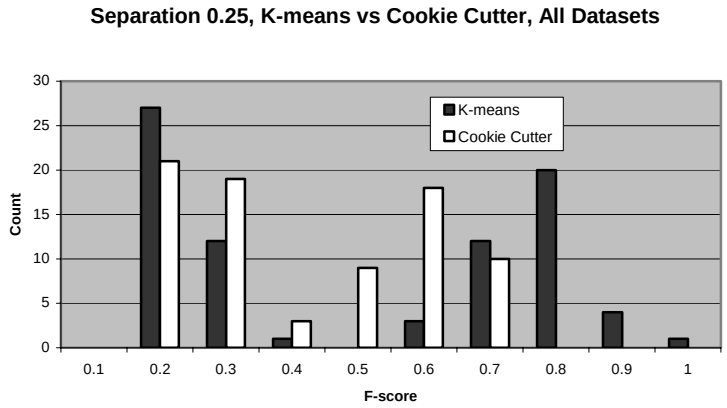
Table 6.3 Average F-scores for K-means and Cookie Cutter on Synthetic Datasets

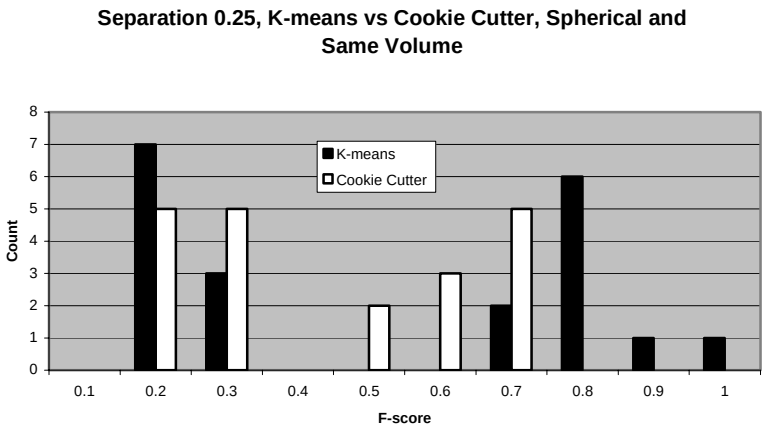
				sep=0.25	sep=0.5	sep=0.75	sep=1.0	sep=2.0
Sphere	Same Vol	Same Size	k-means	0.14±0.01	0.35±0.04	0.83±0.02	0.92±0.03	0.92±0.05
			cookie	0.13±0.01	0.19±0.01	0.35±0.03	0.55±0.06	0.92±0.05
		Diff Size	k-means	0.15±0.01	0.40±0.04	0.78±0.04	0.88±0.04	0.88±0.06
			cookie	0.14±0.01	0.22±0.02	0.39±0.04	0.65±0.02	0.96±0.03
	Diff Vol	Same Size	k-means	0.19±0.04	0.54±0.15	0.84±0.06	0.91±0.03	0.88±0.08
			cookie	0.24±0.03	0.37±0.07	0.58±0.10	0.73±0.08	0.96±0.03
		Diff Size	k-means	0.21±0.04	0.54±0.14	0.79±0.08	0.90±0.04	0.89±0.05
			cookie	0.25±0.02	0.41±0.07	0.62±0.08	0.76±0.08	0.94±0.04
Non-sphere	Same Vol	Same Size	k-means	0.65±0.06	0.94±0.04	0.87±0.08	0.86±0.10	0.88±0.06
			cookie	0.43±0.04	0.76±0.07	0.92±0.05	0.92±0.08	0.99±0.02
		Diff Size	k-means	0.67±0.05	0.86±0.03	0.85±0.06	0.88±0.06	0.88±0.05
			cookie	0.54±0.06	0.86±0.03	0.91±0.08	0.96±0.05	1.00±0.02
	Diff Vol	Same Size	k-means	0.81±0.05	0.83±0.07	0.83±0.06	0.77±0.10	0.81±0.08
			cookie	0.57±0.06	0.83±0.06	0.93±0.04	0.95±0.01	0.99±0.03
		Diff Size	k-means	0.80±0.06	0.86±0.04	0.82±0.05	0.86±0.07	0.85±0.08
			cookie	0.63±0.04	0.89±0.04	0.97±0.02	0.98±0.01	1.00±0.00

To illustrate the detailed behavioral change of the cookie cutter algorithm, we categorize the input datasets by their underlying separation and plot the histograms of f-scores for all the data files in a particular separation in Figure 6.1. For each separation, the histogram on the top is for all the data files while that underneath summarizes all the data files with spherical and same volume clusters. The total number of datasets for each histogram on the top is 80. There are 20 datasets for each histogram on the bottom. By excluding data files with potential different degree of separation, it makes the behavioral change of algorithms in response to the level of difficulty to cluster much

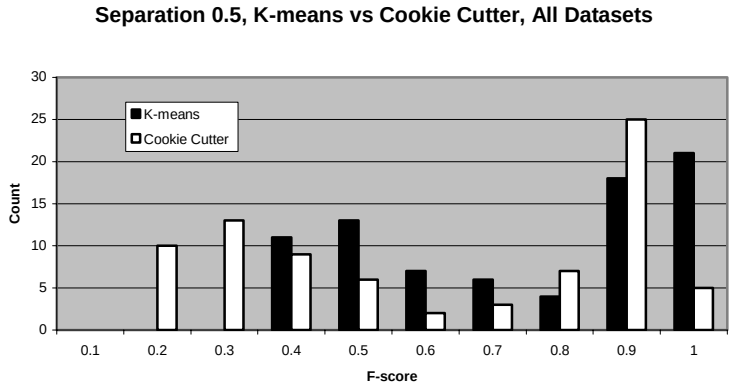
clearer. In these histograms the black bar represents k-means and the white bar is for cookie cutter.

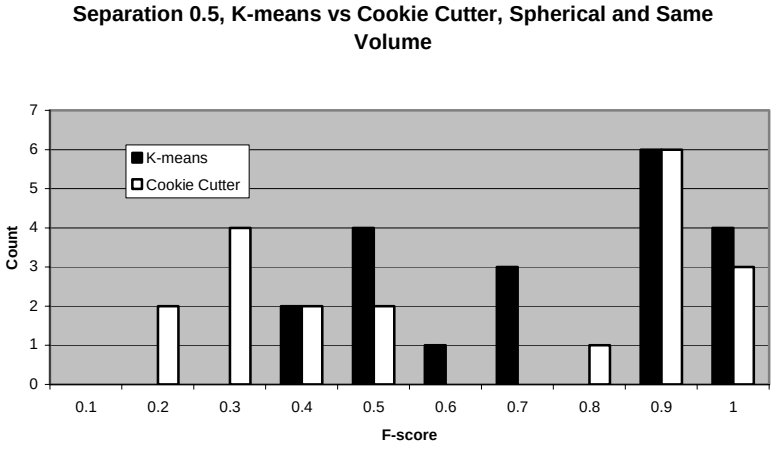
1. Separation = 0.25



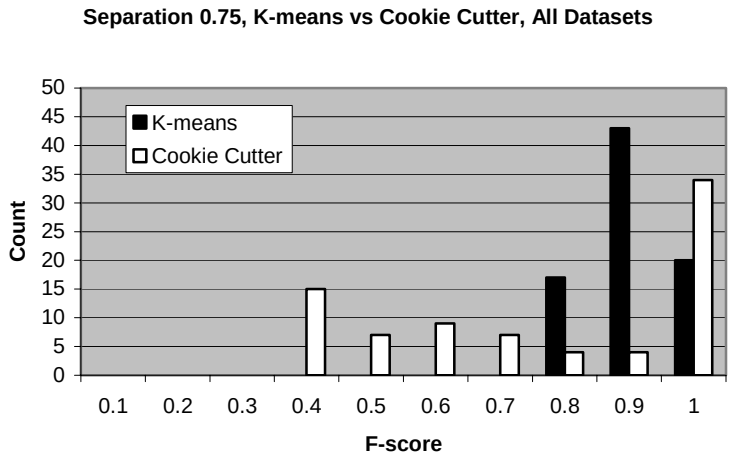


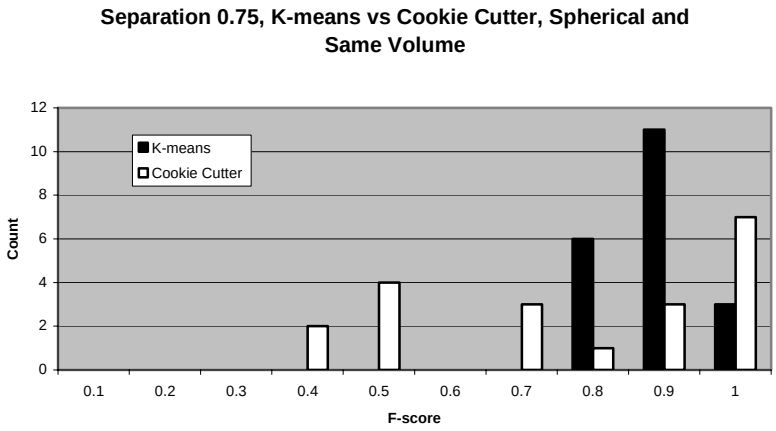
2. Separation = 0.5



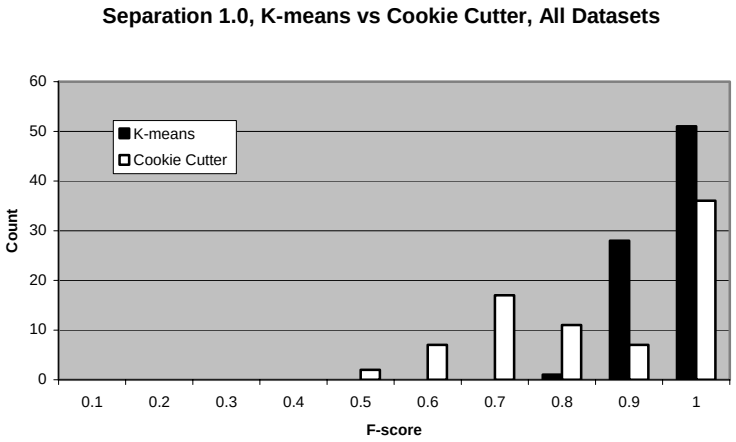


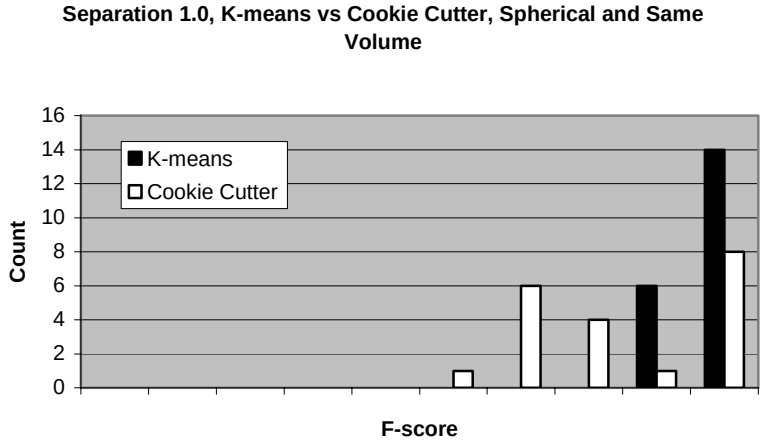
3. Separation=0.75



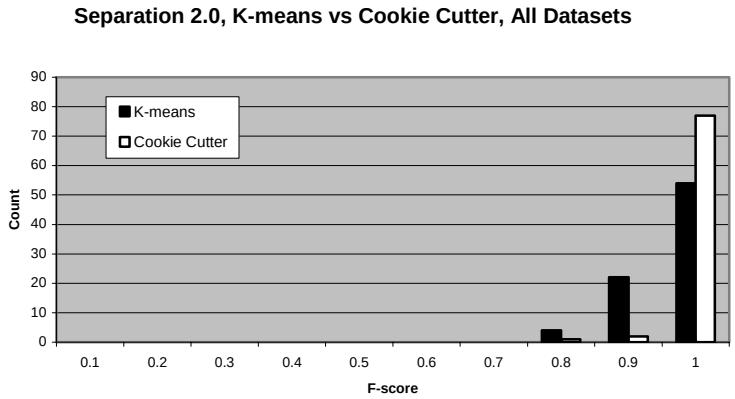


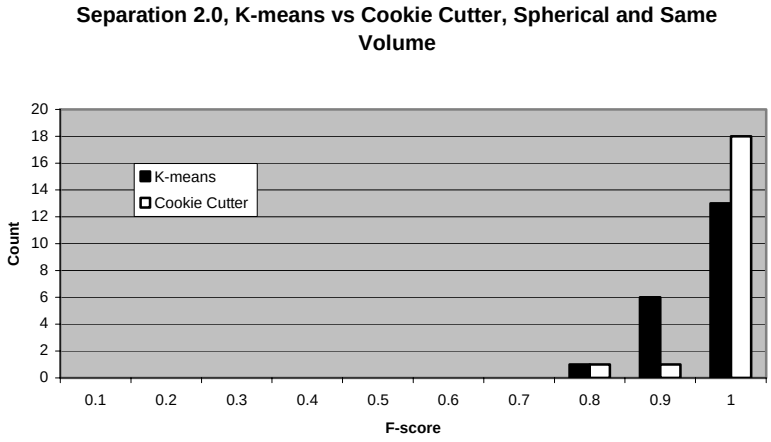
4. Separation = 1.0



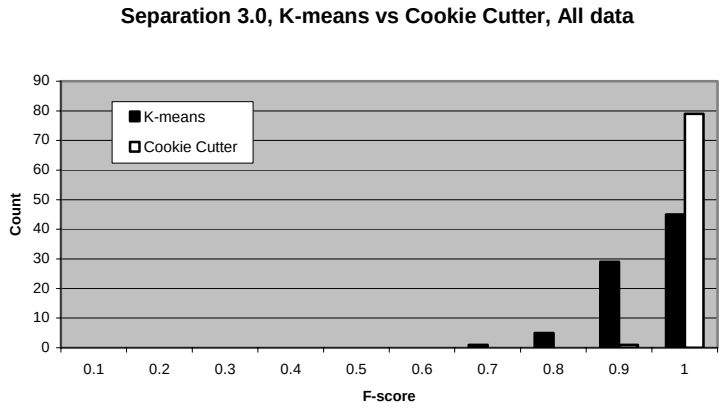


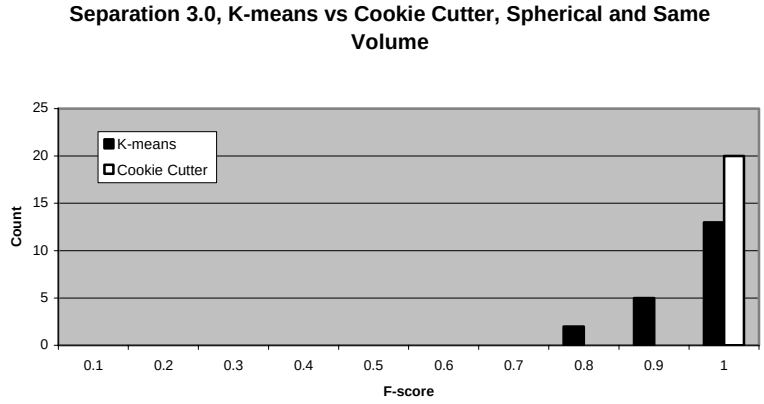
5. Separation = 2.0





6. Separation = 3.0





We make three observations from these histograms. First of all, when the clusters are on top of each other, that is, the smallest separation with which we experiment, both algorithms are not doing very well. The black and white bars that represent the

distribution of f-scores scatter across the whole span of the x-axis (f-score). Secondly, when the separation gets larger and larger, initially k-means wins over cookie cutter clearly. In the histograms for separation 0.5, the distribution of scores for only the black bars narrows somewhat to exclude the lowest 30% of the range; for separations 0.75 and 1.0, the black bars appear in only the upper half of the range of white bars and in the upper 30% of the range overall. This indicates k-means produces consistently better f-scores on these datasets than cookie cutter. Thirdly, however, starting from separation 2.0, the white bars are moving to the right inside each histogram. This shows that cookie cutter is catching up in terms of the cluster quality.

By looking into the details of winning we realize the cookie cutter starts to win at separation 2.0, when the Gaussians are bordering on each other. This leads us to investigate the cases when separation is very large, ranging from 3.0 to 16.0, when the Gaussians are well separated in space. We illustrate our findings in Figure 6.2. Cookie cutter wins clearly over k-means for large separation files. In fact cookie cutter often finds the optimal solution while k-means cannot. The reason is that k-means explore a neighboring region of the initial seeds to locate local optima. The resulting clusters are not far from where it starts. When the Gaussians are spreading in space, the effect of seed trapping exaggerates, making k-means stuck with local optima.

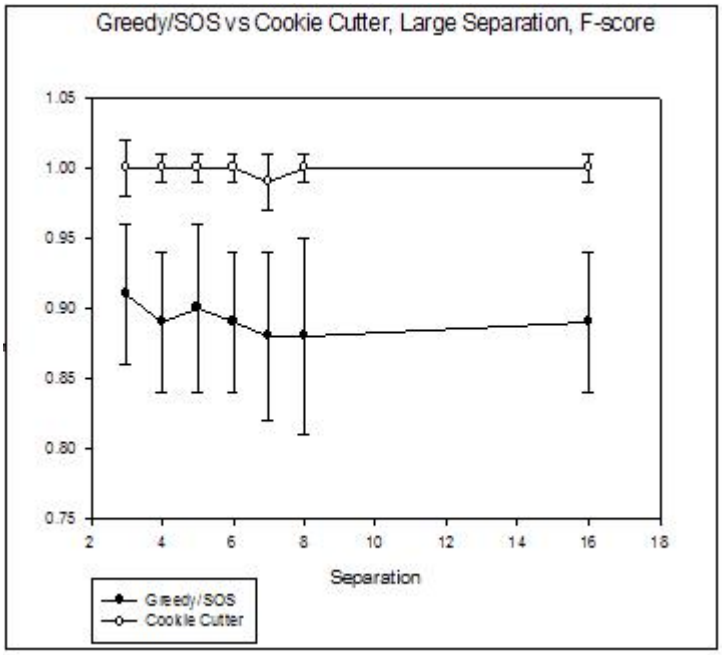


Figure 6.2 Cookie Cutter and K-means for Large Separation Gaussians

6.2.5 Results with Real Datasets

Of the six real datasets we download, our implementation of Cookie Cutter can only finish running on one of them, that is, the dataset "re0". The problem is that the algorithm requires a large amount of memory. It asks for repeated singular value decomposition, which consumes system resources quickly.

The cookie cutter behaves rather strangely on the "re0" dataset. First of all it cannot reach the desired number of clusters, which equals the number of classes in the data. Even if we run the algorithm long enough that the parameter space of possible radii to cut is exhausted in terms of machine precision, the number of clusters the algorithm can find remains rather large and therefore making the f-score very low. We also observe

that the number of clusters does not change after a certain radius. In the end the cookie cutter produces 87 clusters on the dataset “re0”. Of these clusters most of them are very small with less than 10 documents. Three out of 87 clusters have more than 100 documents. In Figure 6.3 we show the distribution of the sizes of the remaining 84 clusters.

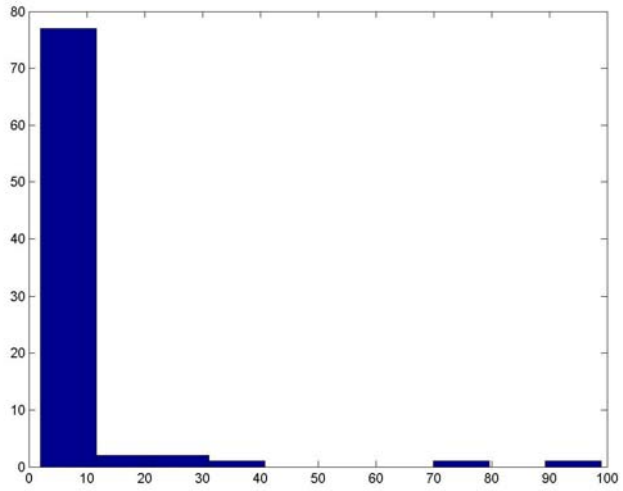


Figure 6.3 Distribution of the Sizes of Clusters Produced by Cookie Cutter on re0

In Figure 6.4 we show the distribution of the precision values for these 87 clusters. The definition of precision is the ratio between the number of documents belonging to a class and the total number of documents in a cluster. The precision of a cluster is the largest precision value over all classes existing in the cluster. In short the cookie cutter produces lots of small clusters with high precision values.

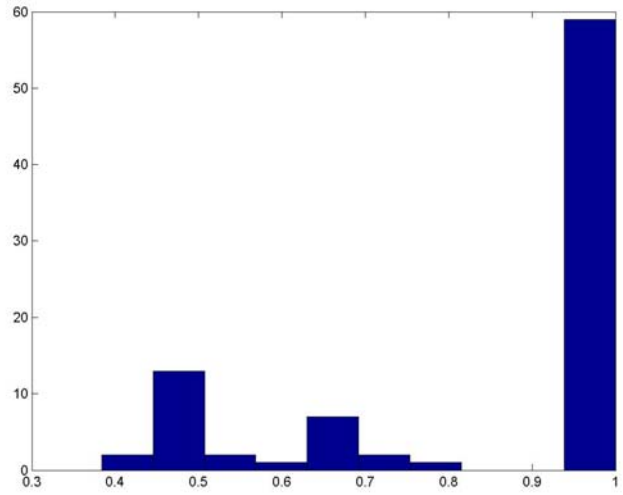


Figure 6.4 Distribution of Precision Values for Clusters by Cookie Cutter on re0

Recall the distance-based clustering assumes a gap between inter-class and intra-class distances. If there is no gap, it would be hard to probe a suitable radius to produce desired number of clusters. In Figure 6.5 we plot a histogram of the inter-class and intra-class distances for all the separation 2.0 files in the generated data sets. The x-axis is the range from minimum pair-wise distance to maximum distance in a dataset. The range is divided into a hundred buckets. The y-axis is the fraction of distances that falls into each bucket. For each input file with separation 2.0, the histograms are added together and then averaged to produce the curve in Figure 6.5. In Figure 6.6 we plot the same inter-class and intra-class distances histogram for the dataset “re0”.

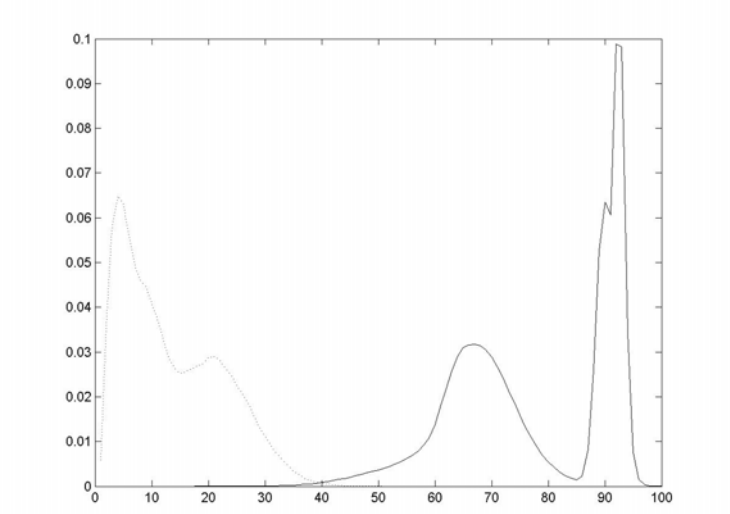


Figure 6.5 Histogram of Inter and Intra Cluster Distances for Separation 2.0 Datasets

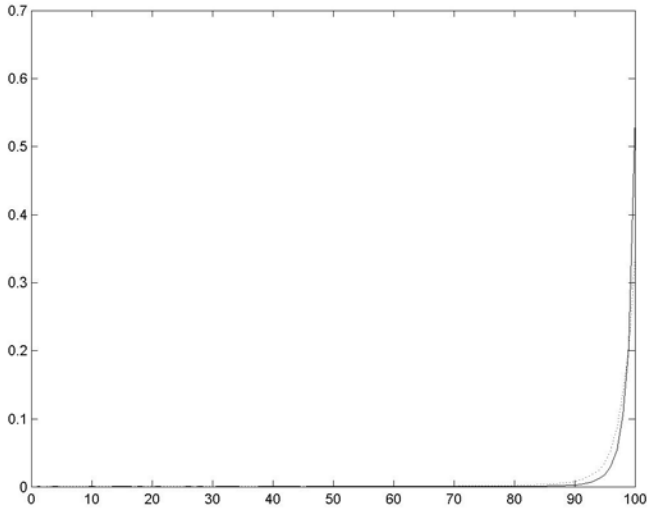


Figure 6.6 Histogram of Inter and Intra Cluster Distances for dataset “re0”

Note in Figure 6.5 the curve of inter-class distances and the curve of intra-class distances peak at separate ends of the distance spectrum and therefore they do not overlap much. However in Figure 6.6, the real dataset “re0”, the two curves almost peak at the same point in the distance spectrum, the only difference being the magnitude.

From the analysis of the data characteristics in Chapter 3, the median separation between classes of the dataset "re0" is roughly 0.6, which implies overlapping classes and therefore the closeness between the peaks.

6.2.6 Dimension Reduction

The goal of dimension reduction is to reduce the dimensions of document vectors so that Cookie Cutter can process the real datasets. It is a pre-processing step in our experiments. The real datasets are processed once and for all. The algorithms are then applied to the dimension reduced versions of these datasets. We append "svd" and the dimensions the dataset is reduced to at the end of the original dataset name. For example, the name "re0.svd.500" indicates a dataset which comes from the dataset "re0", with dimensions reduced to 500.

There are many dimension reduction approaches that can accomplish this goal [17, 25, 27, 31, 115]. Principle component analysis (PCA) is probably the most widely used technique [115]. It projects the original samples to the subspace spanned by the top *k* principle components, that is, eigenvectors of the covariance matrix of the sample set, dropping the dimensions with smaller variances, noted by the smaller eigenvalues. We work with a document-term matrix, which is normally rectangular. Therefore we resort to use the singular value decomposition (SVD) to find the top k singular vectors, instead of the eigenvectors used in PCA.

Given a matrix A, the singular value decomposition calculates its singular values λ_i, left singular vectors u_i, and right singular vectors v_i, $i \in [0, rank(A)]$.

$$A = \sum_i \lambda_i u_i v_i^T$$

The columns of a document-term matrix *A* represent documents and the rows represent the terms. An entry $A(i, j)$ counts the frequency of a term *i* appearing in a document *j*. Each entry is then weighted by some weighting scheme. In our experiments we follow the literature tradition by applying the *tf-idf* scheme. The left and right vectors have several interpretations. For example, the left vectors can be viewed as the underlying concepts of the input corpus because each left vector is a column vector whose dimension is the number of terms in the corpus and the components indicate the degree to which each term participates in the concept. The right vectors can be described as the topics that are represented by combination of documents. Under this construction the right vectors can be interpreted as the approximation to the original document vectors. We chop the dimensions of the right vectors to the desired number of reduced dimensions *r*. The chopped right vectors are used as the dimension reduced version of the original documents.

We give the histogram of inter and intra class distances for the reduced dataset "re0" in Figure 6.7, with dimensions reduced to 500 from 2886. We observe the dimension reduction moves the position of the peaks from the vicinity of the maximum pair-wise distance to the proximity of the minimum pair-wise distance. However, the two peaks for inter-class and intra-class pair-wise distances remain close. Also, the shape of the

distance distributions of the dimension reduced datasets remains similar to those of the distance distributions for the original datasets. This helps to justify the idea of using dimension-reduced datasets to simulate the algorithmic behaviors on the original datasets.

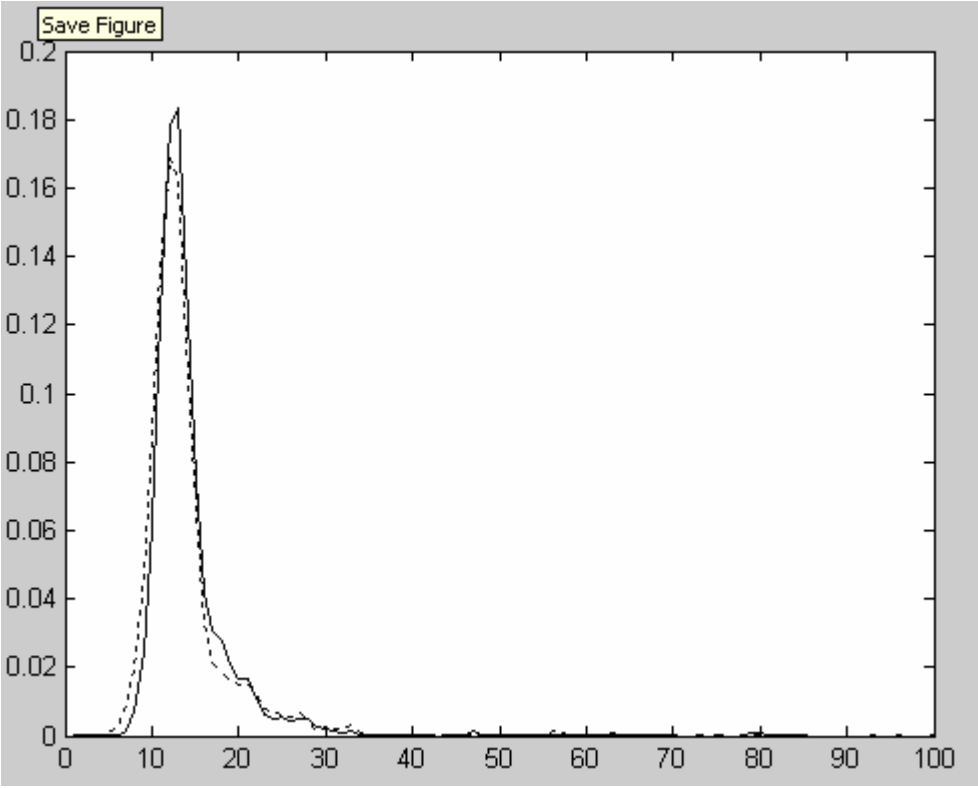


Figure 6.7 Histogram of Inter and Intra Class Distances for Dimension Reduced Dataset re0, Reduced Dimension = 500

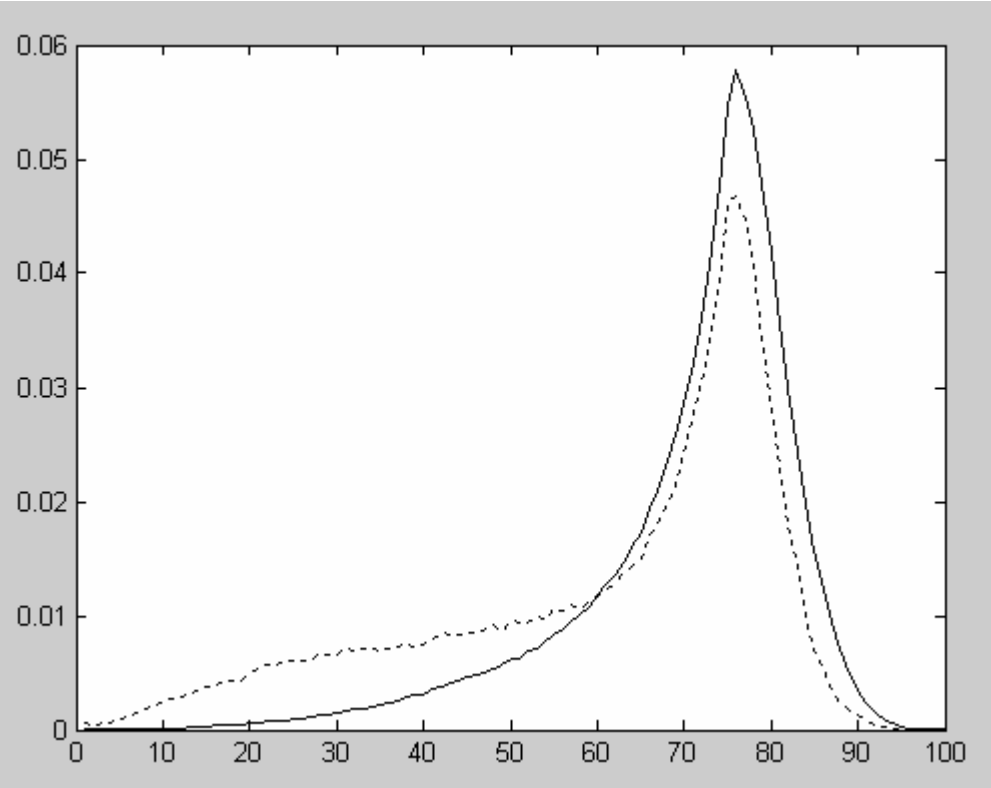


Figure 6.8 Histogram of Inter and Intra Class Pair-wise Distances for Reduced Dataset "re0", Reduced Dimension =10

It is not very clear to how many dimensions each dataset should reduce. In latent semantic indexing the rank of the approximation matrix is determined empirically. It is usually set to several hundreds. We choose to reduce to 500 dimensions for all the datasets. Knowing the arbitrariness of the choice we further reduce more drastically to 10 dimensions, to compare with the 500-dimension reduced dataset. The histogram of inter- and intra- class distances for the reduced dataset "re0" with 10 dimensions is shown in Figure 6.8. The two peaks for inter-class and intra-class distances remain close, but move to the right of the spectrum between minimum and maximum pair-wise distances.

Although we only show the histogram for the effects on the distance distribution for the dataset “re0”, for the other five real datasets, we observe similar effects as in Figure 6.7 and Figure 6.8.

We show the f-scores of both k-means and cookie cutter on the reduced real datasets in Table 6.4. It is not always possible for cookie cutter to produce the same number of clusters as there are classes for the reduced sets. Therefore the equilibrium number of clusters for cookie cutter, which is the number of clusters when cookie cutter stops, is also listed in the table. Specifically for two out of six reduced datasets (the reduced version of dataset “re0” and “re1”), cookie cutter cannot find the desired number of clusters. It happens for both reduced versions, suggesting some intrinsic characteristics of the datasets prevent the cookie cutter from converging.

Generally speaking the algorithms produce the best results in terms of quality on the original datasets. For the algorithm k-means, the quality is better with the original version on all but one dataset. Interestingly all algorithms reach better quality in the reduced version with only 10 dimensions per document than those with 500 dimensions for each document. It is tempting to argue that singular value decomposition reduces inherent noise in the original datasets and therefore leads to better quality in reduced datasets. However it cannot explain why clusters generated on the reduced version with 10 dimensions are not as good as those produced with the original version. It remains an open problem to explain the phenomenon.

It is clear in Table 6.4 that the cookie cutter algorithm does not generate as good clusters as the k-means on both sets of reduced datasets. Given the distance distributions

of these datasets it confirms our expectation that cookie cutter produces cluster of poor quality when there is no gap between the intra- and inter- class distance distributions.

Table 6.4 F-scores of Cookie Cutter and K-means for Reduced Datasets

dataset name	# of classes	k-means (f-scores)	cookie cutter		distance-based clustering	
			# of clusters	f-scores	# of clusters	f-scores
Re0	13	0.50	87	0.34	13	0.33
Re0.svd.500	13	0.32	59	0.33	56	0.35
Re0.svd.10	13	0.43	80	0.34	80	0.34
Re1	25	0.48	n/a	n/a	25	0.25
Re1.svd.500	25	0.18	31	0.20	27	0.20
Re1.svd.10	25	0.44	33	0.20	33	0.20
Tr31	7	0.70	n/a	n/a	7	0.50
Tr31.svd.500	7	0.33	9	0.38	7	0.30
Tr31.svd.10	7	0.55	7	0.43	7	0.52
Tr41	10	0.64	n/a	n/a	10	0.29
Tr41.svd.500	10	0.26	13	0.29	10	0.29
Tr41.svd.10	10	0.72	10	0.50	10	0.46
Wap	20	0.46	n/a	n/a	20	0.30
Wap.svd.500	20	0.16	26	0.17	20	0.18
Wap.svd.10	20	0.53	20	0.37	20	0.37
Fbis	17	0.55	n/a	n/a	17	0.30
Fbis.svd.500	17	0.19	17	0.20	17	0.22
Fbis.svd.10	17	0.53	17	0.44	17	0.44

6.2.7 Speed up of Cookie Cutter – Distance-based Clustering

Cookie cutter is not a fast algorithm due to the repeated singular value decomposition. Note in our experiments, we use SVD in two separation contexts. Once, the SVD is used as a pre-processing step to reduce dimensions of real datasets that cannot run because of their sizes. The other use of SVD is within the main loop of Cookie Cutter:

before the points are grouped by a hyper-sphere with a fixed radius, they are projected into the spectral space. The usage of SVD for the second case happens every round of the cutting, therefore takes up a significant portion of the total running time. We discuss the possibility of removing this step in this section.

It is proved that using SVD to project to the spectral space can make learning of mixture of Gaussians more efficient to achieve a provable approximation. It is not clear how much quality we gain by using it. If the quality of directly applying distance-based clustering without SVD is acceptable, it makes sense to skip the repeated SVD to save time.

We apply the distance-based clustering directly to both the original and reduced datasets. Comparing with the quality of clusters produced by cookie cutter, the quality from directly applying distance-based clustering is almost as good. The difference in quality is small between cookie cutter and distance-based clustering on these real datasets. Taking into account the distance distribution of indistinguishable inter and intra class distances for these datasets, it suggests the singular value decomposition does not help much when the mixture of Gaussians are not well separated. If for any reason cookie cutter is the preferred algorithm on this kind of data, distance-based clustering can be substituted because it finishes much faster with almost the same quality.

The above observations are based on Table 6.4, which details the quality of clusters for the algorithms k-means and cookie cutter on the real datasets and their variants. In Table 6.5 we apply the distance-based clustering algorithm directly on the synthetic datasets.

Table 6.5 Average F-scores for Cookie Cutter and Distance-based Clustering on Synthetic Data

				0.25	0.5	0.75	1.0	2.0
Sphere	Same Vol	Same Size	Cc	0.13±0.01	0.19±0.01	0.35±0.03	0.55±0.06	0.92±0.05
			Dist	0.14±0.01	0.18±0.01	0.35±0.02	0.54±0.05	0.92±0.05
		Diff Size	Cc	0.14±0.01	0.22±0.02	0.39±0.04	0.65±0.03	0.96±0.03
			Dist	0.14±0.01	0.22±0.02	0.38±0.04	0.65±0.03	0.96±0.03
	Diff Vol	Same Size	Cc	0.24±0.03	0.37±0.07	0.58±0.10	0.73±0.08	0.96±0.03
			Dist	0.23±0.02	0.37±0.08	0.58±0.09	0.72±0.08	0.96±0.03
		Diff Size	Cc	0.25±0.02	0.41±0.07	0.62±0.08	0.76±0.08	0.94±0.04
			Dist	0.25±0.02	0.42±0.06	0.61±0.08	0.76±0.07	0.95±0.04
Non-sphere	Same Vol	Same Size	Cc	0.43±0.04	0.76±0.07	0.92±0.05	0.92±0.08	0.99±0.02
			Dist	0.43±0.04	0.75±0.07	0.92±0.05	0.92±0.08	0.99±0.02
		Diff Size	Cc	0.54±0.06	0.86±0.03	0.91±0.08	0.96±0.05	1.00±0.00
			Dist	0.55±0.06	0.86±0.03	0.91±0.08	0.96±0.05	1.00±0.00
	Diff Vol	Same Size	Cc	0.57±0.06	0.83±0.06	0.93±0.04	0.95±0.01	0.99±0.03
			Dist	0.56±0.06	0.82±0.06	0.93±0.04	0.95±0.01	0.99±0.03
		Diff Size	Cc	0.63±0.04	0.89±0.04	0.97±0.02	0.98±0.01	1.00±0.00
			Dist	0.63±0.04	0.89±0.04	0.97±0.02	0.98±0.01	1.00±0.00

Table 6.6 Win-Lose Relationship between Cookie Cutter and Distance-based Clustering

	Cookie cutter	Distance-based clustering	tie
Spherical	29%	24%	47%
Non-spherical	11%	5%	84%
Same volume	23%	16%	61%
Different volume	17%	13%	70%
Same size	18%	16%	66%
Different size	22%	14%	64%

We observe that between the algorithms cookie cutter and distance-based clustering, there are a large percentage of ties on synthetic datasets, indicating the distance-based clustering can produce clusters of comparable quality. Given the fact that distance-based clustering runs at least an order of magnitude faster than Cookie Cutter, we

consider distance-based clustering a more practical choice than Cookie Cutter. The SVD acts significantly in the theoretical proof by substituting k for n in a dominating term of the time complexity. However, it does not yield much gain in terms of the quality of clusters in reality.

6.3 Conclusion

We do not consider the Spectral Cut algorithm a serious competitor to other algorithms we present, because of its poor performance on the synthetic datasets. The Cookie Cutter algorithm, with the projection, is not attractive either, because of its long running time. The singular value decomposition helps to lower the theoretical lower bound for the worst case scenario, while not helping much in terms of the cluster quality in reality. However, distance-based clustering shows moderate promise based on our experiments. It produces clusters with roughly the same quality as the Cookie Cutter algorithm.

Chapter 7 Conclusion and Future Work

7.1 Contributions

In this thesis, we present a framework to compare clustering algorithms and evaluate six clustering algorithms using this framework. There are six major contributions in this work.

7.1.1 Comparison and Analysis of Three External Evaluation Indices

The first contribution involves comparison and analysis of three external evaluation indices. In the literature most papers use a single evaluation index to determine the ranking of algorithms. However, it is not clear whether the ranking would change if switching to another evaluation index. We choose three external evaluation indices, f-score, Hubert's Γ and Q_0, from the area of text classification, information retrieval and information theory respectively, and compare their performance. We conduct three experiments to investigate the converging values for random cluster labels, the difference in rankings by the three indices for two algorithms, and the distribution of the index values. We are able to plot two indices, f-score and Hubert's Γ. They both satisfy

the requirements of responsiveness and stability. We discover that two of the indices, Hubert's Γ and Q_0, converge to the middle of their value range. While it makes perfect sense, having only half the range to distinguish among good and better is not as good as a larger range. When we check the rankings of two algorithms by the three indices, most disagreement happen when the clusters are either too hard or too easy to find. Datasets in real life usually fall into the category of neither too hard nor too easy. Therefore, given the observations from the experiments, when using a single index, we recommend the use of f-score because of its larger usable range, acceptable value distribution and popularity in recent literature.

7.1.2 Comparative Investigation of Four Iterative Clustering Algorithms

Evaluation of clustering algorithms is a hard problem. We identify three components that can influence the quality of the clustering results: (1) objective functions; (2) optimization approaches; and (3) data characteristics. The objective functions and optimization approaches belong to the clustering algorithms. The objective function defines what a cluster is. Changing the objective function can affect the type of clusters found by an algorithm. The optimization approach determines the quality of the local optima found by the algorithm. If the objective function is defined to find clusters with similar data characteristics as those of the datasets, the output quality is good. Otherwise, if the objective function does not capture the characteristics well, the quality is unlikely to be good.

Our second contribution is the comparative investigation of four iterative clustering algorithms. We change the components of the popular k-means algorithm. The algorithms are sum-of-squares with greedy (k-means), sum-of-squares with Kernighan-Lin, min-max cut with greedy, and min-max cut with Kernighan-Lin. We use standard implementations of these algorithms, but explore their sensitivity as probabilistic algorithms.

The approach Kernighan-Lin, when applied to optimize sum-of-squares, improves the greedy results significantly. The algorithm sum-of-squares with Kernighan-Lin produces clusters with best quality of the four candidates. Meanwhile, in our experiments, k-means produces clusters with 95% of the quality achieved by the best algorithm, but with an order of magnitude faster running time. Therefore, we conclude that k-means is a viable option for large datasets, not only because of its time complexity but also its reasonably good quality.

When we substitute Kernighan-Lin for greedy when using the min-max cut objective, the resulting algorithm gives slightly worse quality of clusters. We show evidence that it is possible, due to the fact that greedy can optimize the objective min-max cut rather well that there is not much room for Kernighan-Lin to improve upon greedy. These two algorithms give 91% of the best quality achievable among four candidates with running times about seven times longer than k-means.

7.1.3 Comparison Between Sum-of-squares and Min-max Cut

Our third contribution is that we investigate the objective functions sum-of-squares and min-max cut by controlling the data characteristics of a generative model based on a mixture of Gaussians. We choose to control the shape, volume and size of a Gaussian cluster. We discover that the objective function min-max cut does not capture the essence of the class structure when classes are different sized. Therefore the algorithms with min-max cut lose overwhelmingly to algorithms optimizing sum-of-squares when presented with datasets with different sized classes.

The objective sum-of-squares is criticized for splitting large classes when classes have uneven sizes. In fact, algorithms with sum-of-squares win over algorithms with min-max cut in the category of different sized classes. The objective sum-of-squares may not be the best alternative to deal with different sized classes, however, in our experiments, it is at least better than a minimum-cut based objective in handling uneven sized classes.

Moreover, we observe that the objective min-max cut is not shape-independent. We choose a minimum-cut based objective because of its claim of not favoring a particular shape, in contrast to the criticism of sum-of-squares favoring spherical shapes. However, from our experiments, when classes are of the same size -- therefore the effect of the size factor is eliminated -- min-max cut with greedy wins the least in the category of non-spherical datasets comparing with sum-of-squares with greedy. Comparing the f-scores for spherical and non-spherical datasets of the four algorithms, we observe that the algorithms using min-max cut give better f-scores for spherical datasets when

classes are well separated. This indicates the min-max cut objective does not produce clusters with similar quality for different shaped classes.

The objective sum-of-squares are long known for its favoring spherical shapes. However, in our experiments, we discover while it is true that sum-of-squares with greedy favors spherical shapes, the same objective, when working with Kernighan-Lin, gives better f-scores for non-spherical datasets than spherical ones. This suggests the favoring is due to the poor quality of the local optima found by the greedy optimization, not necessarily the objective function itself.

7.1.4 Cookie Cutter

Our fourth contribution is that we implement a fairly new theoretical algorithm with a provably correct guarantee on the quality of clusters it finds and analyze its strengths and weaknesses when systematically applying to synthetic and (transformed) real datasets. The algorithm uses distance-based clustering in the projected space spanned by the top k singular vectors of the sample matrix. The projection allows better worst-case running time to be proven. When we apply the algorithm to datasets, we discover that the projection adds significantly long execution time while not gaining much in terms of the cluster quality, despite its prominent position in the theoretical analysis. The main reason is that the theoretical analysis assumes the classes are well separated. However, in real world datasets it is rarely the case. It happens that applying distance-based clustering directly to the original space can achieve comparable quality and therefore making the projection only necessary for the theoretical analysis.

7.1.5 Comparison Framework

Our fifth contribution is the framework we propose to compare clustering algorithms. Previously in the literature, there are similar studies that choose algorithms, run on some datasets, and evaluate by some standard. However our framework differs in the generative model and the consideration of multiple evaluation indices. Our generative model extends those in the literature because we can control data characteristics directly and explicitly. We test the algorithms on the full spectrum of the data characteristics of underlying classes. In addition to the systematic test on the synthetic datasets, we apply the algorithms to real world datasets and compare the conclusions to what we find in synthetic datasets. By tying the algorithms with data characteristics, our framework increases its prediction power because as long as the data characteristics remain the same, the algorithmic comparison conclusions can carry over to previously unseen datasets. This is the main difference between our framework and previous studies.

7.1.6 Separation and Alternate Separation

Our sixth contribution is the exploration of the notion of separation to capture clustering difficulty. We introduce separation to explore the full space of the data generative model. However, we have discovered the definition of separation borrowed from theoretical analysis is not the best choice because it does not indicate the same level of difficulty for Gaussians with different variances. Two non-spherical Gaussians can be

rather far away from each other even when the separation is only 1 because separation is defined on their longer axes and their longer axes could be parallel in space.

We define an alternative definition of separation that can alleviate the problem of different level of difficulty for Gaussians with different variances. The Alternate Separation between two classes is defined to be the ratio between the smaller class diameter to the diameter of the merged class. The larger the Alternate Separation, the more difficult the two classes to be clustered. The Alternate Separation of a clustering is the maximum pair-wise Alternate Separation of all clusters in the clustering. We compare f-scores of spherical and non-spherical datasets at similar alternate separations, and show the gap is small. Our experiments show that the Alternate Separation is not linearly related to the theoretical definition of separation, and it is a viable alternative for studying clustering difficulty.

7.2 Future Work

We have proposed a comparison framework to compare the cluster quality of clustering algorithms. We have investigated six clustering algorithms using this framework. There are many other clustering algorithms that are worth comparing with each other. For example, it is good to relate the quality performance of multiple clustering algorithms using identical testing datasets. Some algorithms are never compared with each other because they are created or become popular in different times, or belonging to different research fields. Without direct comparison, there is doubt as to the performance of a

new or under-studied algorithm on completely different datasets or using unfamiliar testing technologies; this often prevents users from trying out other candidate clustering algorithms.

To test an algorithm systematically, it is necessary to generate data. Mixture of Gaussians is one approach. However, the Gaussians generative process may introduce bias toward the sum-of-squares objective because it generates a cluster around a center: the mean of a Gaussian. Therefore it would be interesting to see the rankings of these objective functions when employing another model which favors the cut-based objective. We briefly describe an idea. Starting from a *d*-dimensional space with uniformly distributed points, we repeatedly draw random lines. For each line, the points are pushed away from the line according to their distance to the line. Thus, a random line is like a cut into the space. All the experiments described in this thesis could be repeated on datasets generated by such a model. The results can be analyzed in a similar way to see whether the Gaussian model is favoring the sum-of-squares.

Moreover, depending on the application areas, a more specific generative model can be used. For example, if document clustering algorithms are the research target, a document generative model can supplement both the Gaussian and the cut-based model. The existing models in the literature are rather naïve [5, 87]. They use a Gaussian cloud to simulate a topic and assume independence among words. It is not clear whether such models capture more essence of the application area than the mixture of Gaussians. A sophisticated document generative model can be a topic of another thesis. Therefore, we leave the integration of an advanced document generative model to future work.

"Data characteristic" is a general term which may vary across fields and applications. We choose the characteristics that are intuitive and easy to control in our experiments, hoping to shed light on the relationship between objectives and certain data characteristics. There might be other factors that are important and lead to a better explanation of behaviors observed in practice. Our study is meant to be a start point in theoretically analyzing clustering algorithms; it surely reveals only the tip of the iceberg.

A candidate for another characteristic to study is the orientation of a cluster in the *d*-dimensional space. The orientation of a Gaussian cloud can be controlled by the direction of the principle eigenvector of the covariance matrix for the Gaussian. Therefore, another characteristic can be introduced into the generative model by generating covariance matrices with different directed principle eigenvectors.

Another important feature to include in controllable data characteristics is the curvature of a cluster. All clusters we are generating are convex clusters. It would be nice if concave clusters can be included in the testing cases. In reality, many clusters in biological and clinical datasets are concave [9]. The comparative performance in terms of quality for existing algorithms on such concave clusters is unknown. Given the rising importance of computational biology and bioinformatics, including concave clusters into algorithm analysis will soon become a must.

Reference

1. *Approximation Algorithms for NP-hard Problems*. First Edition ed, ed. D.S. Hochbaum. 1996: Course Technology. 624.
2. *Yahoo!* in *www.yahoo.com*.
3. Agrawal, R., et al. *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications*. in *ACM SIGMOD Conference on Management of Data*. 1998.
4. Aslam, J., K. Pelekhov, and D. Rus. *Static and Dynamic Information Organization with Star Clusters*. in *CIKM*. 1998.
5. Azar, Y., et al. *Spectral Analysis of Data*. in *ACM Symposium on Theory of Computing*. 2001.
6. Baeza-Yates, R. and B. Ribeiro-Neto, *Modern Information Retrieval*. 1999: Addison-Wesley.
7. Baker, L.D. and A.K. McCallum. *Distributional Clustering of Words for Text Classification*. in *ACM SIGIR*. 1998.
8. Banerjee, A. *Generative Model-based Clustering of Directional Data*. in *Conference on Knowledge Discovery in Data*. 2003. Washington, D.C.
9. Banfield, J.D. and A.E. Raftery, *Model-based Gaussian and Non-Gaussian Clustering.* Biometrics, 1993. **49**(3): p. 803-821.
10. Bansal, N., A. Blue, and S. Chawla. *Correlation Clustering*. in *FOCS*. 2002.
11. Ben-Dor, A., R. Shamir, and Z. Yakhini, *Clustering Gene Expression Patterns.* Journal of Computational Biology, 1999. **6**(3/4): p. 281-297.
12. Blashfield, R.K., *Mixture Model Tests of Cluster Analysis: Accuracy of Four Agglomerative Hierarchical Methods.* Psychological Bulletin, 1976. **83**(3): p. 377-388.
13. Broder, A.Z., et al. *Syntactic Clustering of the Web*. in *the Sixth International WWW Conference*. 1997.
14. Cadez, I., et al. *Visualization of Navigation Patterns on a Web Site Using Model-based Clustering*. in *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2000. Boston, MA.
15. Cadez, I.V., et al. *Visualization of Navigation Patterns on a Web Site Using Model-based Clustering*. in *Knowledge Discovery and Data Mining*. 2000.
16. Cai, L. and T. Hofmann. *Hierarchical Document Categorization with Support Vector Machines*. in *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management*. 2004. Washington, D.C., USA.
17. Carreira-Perpinan, M.A., *A Review of Dimension Reduction Techniques*. 1997, Department of Computer Science, University of Sheffield.
18. Charikar, M., V. Guruswami, and A. Wirth. *Clustering with Qualitative Information*. in *FOCS*. 2003.
19. Chiu, T., et al. *A Robust and Scalable Clustering Algorithm for Mixed Type Attributes in Large Database Environment*. in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2001. San Francisco, CA.
20. Chung, F.R.K., *Spectral Graph Theory*. 1997: American Mathematical Society.
21. CLUTO. *A Clustering Toolkit*. in *http://www-users.cs.umn.edu/~karypis/cluto/*. 2002.

22. Cunningham, K.M. and J.C. Ogilvie, *Evaluation of Hierarchical Grouping Techniques: A Preliminary Study.* The Computer Journal, 1972. **15**: p. 209-213.
23. Cutting, D.R., et al. *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections*. in *ACM SIGIR*. 1992.
24. Czumaj, A. and C. Sohler. *Sublinear-Time Approximation for Clustering via Random Sampling*. in *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*. 2004. Turku, Finland.
25. Dasgupta, S. *Experiments with Random Projection*. in *Sixteenth Conference on Uncertainty in Artificial Intelligence*. 2000.
26. Dasgupta, S. *Learning Mixture of Gaussians*. in *IEEE Symposium on Foundations of Computer Science*. 1999.
27. Dasgupta, S. and A. Gupta, *An Elementary Proof of a Theorem of Johnson and Lindenstrauss.* Random Structures and Algorithms, 2003. **22**(1): p. 60-65.
28. Dempster, A.P., N.M. Laird, and D.B. Rubin, *Maximum likelihood from incomplete dtat via the EM algorithm.* Journal of Royal Statistics Society, 1977. **Series B**(39): p. 1-38.
29. Dhillon, I.S., S. Mallela, and R. Kumar. *Enhanced Word Clustering for Hierarchical Text Classification*. in *Conference on Knowledge Discovery in Data*. 2002. Edmonton, Alberta, Canada.
30. Ding, C., et al. *Spectral Min-max Cut for Graph Partitioning and Data Clustering*. in *Proceedings of the First IEEE International Conference on Data Mining*. 2001. San Jose.
31. Ding, C., et al. *Adaptive Dimension Reduction for Clustering High Dimensional Data*. in *Proceedings of 2nd IEEE Internatioanl Conference on Data Mining*. 2002. Maebashi, Japan.
32. Dom, B., *An information-theoretic external cluster-validity measure*. 2001, IBM Research.
33. Drineas, P., et al. *Clustering in Large Graphs and Matrices*. in *SODA: ACM-SIAM Symposium on Discrete Algorithms*. 1999.
34. Dubes, R. and A.K. Jain, *Validity Studies in Clustering Methodologies.* Pattern Recognition, 1979. **11**: p. 235-254.
35. Duda, R.O., P.E. Hart, and D.G. Stork, *Pattern Classification*. 2001: Wiley.
36. Duda, R.O., P.E. Hart, and D.G. Stork, *Pattern Classification*. 2nd Edition ed. 2000: Wiley-Interscience.
37. El-Hamdouchi, A. and P. Willett. *Hierarchic Document Classification Using Ward's Clustering Method*. in *Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1986. PIsa, Italy.
38. Ester, M., et al. *A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. in *2nd International Conference on Knowledge Discovery and Data Mining*. 1996. Portland, Oregon: AAAI Press.
39. Estivill-Castro, V. *Why So Many Clustering Algorithms - A Position Paper*. in *SIGKDD Explorations*. 2002.
40. Everitt, B.S., S. Landau, and M. Leese, *Cluster Analysis*. 4th ed. 2001, London: Arnold.
41. Fiduccia, C.M. and R.M. Mattheyses. *A Linear-time Heuristic for Improving Network Partitions*. in *Proceedings of the 19th Conference on Design Automation*. 1982.
42. Flake, G.W., S. Lawrence, and C.L. Giles. *Efficient Identification of Web Communities*. in *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*. 2000. Boston, MA.
43. Friedman, H.P. and J. Rubin, *On Some Invariant Criteria for Grouping Data.* Journal of the American Statistical Association, 1967. **62**(320): p. 1159-1178.
44. Frieze, A. and R. Kannan, *Quick Approximation to Matrices and Applications.* 1999.
45. Frieze, A., R. Kannan, and S. Vempala, *Fast monte-carlo algorithms for finding low-rank approximations.* Journal of the ACM, 2004. **51**(6): p. 1025-1041.
46. Gaussier, E., et al. *A Hierarchical Model for Clustering and Categorising Documents*. in *Proceedings of the 24th BCS-IRSG European Colloquim on IR Research: Advances in Information Retrieval*. 2002.

47. Gavrilov, M., et al. *Mining the Stock Market: Which Measure is Best?* in *the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2000. Boston, Massachusetts, United States: ACM Press, New York, NY, USA.
48. Gower, J.C. and G.J.S. Ross, *Minimum Spanning Trees and Single Linkage Cluster Analysis.* Applied Statistics, 1969. **18**(1): p. 54-64.
49. Guha, S., et al. *Clustering Data Streams*. in *IEEE Symposium on Foundations of Computer Science*. 2000.
50. Guha, S., R. Rastogi, and K. Shim. *CURE: An Efficient Clustering Algorithm for Large Database*. in *ACM SIGMOD*. 1998.
51. Guha, S., R. Rastogi, and K. Shim, *ROCK: A Robust Clustering Algorithm for Categorical Attributes.* Information Systems, 2000. **25**(5): p. 345-366.
52. Hamerly, G. and C. Elkan. *Alternatives to the K-means Algorithm that Find Better Clusterings.* in *Proccedings of the Eleventh International Conference on Information and Knowledge Management*. 2003. McLean, Virginia, USA.
53. Han, E.-H., et al. *WebACE: A Web Agent for Document Categorization and Exploration*. in *Proceedings of the 2nd International Conference on Autonomous Agents*. 1998.
54. Harel, D. and Y. Koren. *Clustering Spatial Data Using Random Walks*. in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2001. San Francisco, CA.
55. Haveliwala, T.H., A. Gionis, and P. Indyk. *Scalable Techniques for Clustering the Web*. in *Third International Workshop on the Web and Databases*. 2000. Dallas, Texas.
56. Indyk, P. *A Sublinear Time Approximation Scheme for Clustering in Metric Spaces*. in *IEEE Symposium on Foundations of Computer Science*. 1999.
57. Iwayama, M. and T. Tokunaga. *Cluster-based Text Categorization: A Comparison of Category Search Strategies*. in *SIGIR*. 1995. Seattle: ACM Press, New York, US.
58. Jain, A.K. and R.C. Dubes, *Algorithms for Clustering Data*. 1988: Prentice Hall.
59. Johnson, D.S., et al., *Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning.* Operations Research, 1989. **37**(6): p. 865-892.
60. Jung, S. and T.-S. Kim. *An Agglomerative Hierarchical Clustering Using Partial Maximum Array and Incremental Similarity Computat*. in *IEEE International Conference on Data Mining*. 2001. San Jose, CA.
61. Kamvar, S., D. Klein, and C. Manning. *Spectral Learning*. in *IJCAI*. 2003.
62. Kamvar, S.D., D. Klein, and C.D. Manning. *Interpreting and Extending Classical Agglomerative Clustering Algorithms using a Model-based Approach*. in *Procedings of the Nineteenth International Conference on Machine Learning*. 2002.
63. Kannan, R., S. Vempala, and A. Vetta. *On Clusterings -- Good, Bad and Spectral*. in *FOCS*. 2000.
64. Karypis, G., E.-H. Han, and V. Kumar, *Chameleon: A Hierarchial Clustering Algorithm Using Dynamic Modeling.* Computer, 1999. **32**(8): p. 68-75.
65. Kernighan, B.W. and S. Lin, *An Efficient Heuristic Procedure for Partitioning Graphs.* The Bell System Technical Journal, 1970. **49**(1): p. 291-307.
66. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, *Optimization by Simulated Annealing.* Science, 1983. **220**(4598): p. 671-680.
67. Kleinberg, J. *Authoritative sources in a hyperlinked environment*. in *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*. 1998.
68. Kleinberg, J. *An Impossibility Theorem for Clustering*. in *Advances in Neural Information Processing Systems (NIPS) 15*. 2002.
69. Kollios, G., et al., *Efficient Biased Sampling for Approximate Clustering and Outlier Detection in Large Data Sets.* IEEE Transactions on Knowledge and Data Engineering, 2003. **15**(5): p. 1170-1187.
70. Kuiper, F.K. and L. Fisher, *A Monte Carlo Comparison of Six clustering procedures.* Bometrics, 1975. **31**: p. 777-783.

71. Larsen, B. and C. Aone. *Fast and Effective Text Mining Using Linear-time Document Clustering*. in *Conference on Knowledge Discovery in Data*. 1999. San Diego.
72. Larsen, B. and C. Aone. *Fast and effective text mining using linear-time document clustering*. in *SIGKDD*. 1999.
73. Lewis, D.D. *Reuters-21578*. in *http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html*. 2004.
74. Lindsay, B.G., *Mixture models: theory, geometry and applications*. NSF-CBMS Regional Conference Series in Probability and Statistics. Vol. 5. 1995: Institute of Mathematical Statistics.
75. Ling, R.F., *Probability Theory of Cluster Analysis.* Journal of American Statistical Association, 1973. **68**: p. 159-164.
76. Liu, B., Y. Xia, and P.S. Yu. *Clustering through Decision Tree Construction*. in *Conference on Information and Knowledge Management*. 2000. McLean, Virginia.
77. McCallum, A., K. Nigam, and L.H. Ungar. *Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching*. in *Knowledge Discovery and Data Mining*. 2000.
78. McQueen, J. *Some Methods for Classification and Analysis of Multivariate Observations*. in *5th Berkeley Symposium on Mathematics, Statistics and Probablity*. 1967.
79. Meila, M., *Comparing clusterings*. 2003, University of Washington, Department of Statistics.
80. Milligan, Soon, and Sokol, *The Effect of Cluster Size, Dimensionality and the Number of Clusters on Recovery of True Cluster Structure.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1983. **5**(1): p. 40-47.
81. Milligan, G.W., *A Monte Carlo Study of Thirty Internal Criterion Measures for Cluster Analysis.* Psychometrika, 1981. **46**(2): p. 187-199.
82. Modha, D.S. and W.S. Spangler. *Clustering Hypertext with Applications to Web Searching*. in *ACM Hypertext Conference*. 2000.
83. Mojena, R., *Hierarchical Grouping Methods and Stopping Rules: An Evaluation.* Computer, 1975. **20**(4): p. 359-366.
84. Ness, J.W.V., *A Method for Comparing Two Hierarchial Clusterings: Comment.* Journal o fthe American Statistical Association, 1983. **78**(383): p. 576-579.
85. Ng, A.Y., M.I. Jordan, and Y. Weiss. *On Spectral Clustering: Analysis and Algorithm*. in *NIPS*. 2002.
86. Ng, R.T. and J. Han. *Efficient and Effective Clustering Methods for Spatial Data Mining*. in *Proceedings of the 20th VLDB Conference*. 1994. Santiago, Chile.
87. Papadimitriou, C.H., et al., *Latent Semantic Indexing: A Probabilistic Analysis.* Journal of Comp. and System Sciences, 2000. **61**: p. 217-235.
88. Pena, J.M., J.A. Lozano, and P. Larranaga, *An Empirical Comparison of Four Initialization Methods for the K-means Algorithm.* Pattern Recognition Letters, 1999. **20**(50): p. 1027-1040.
89. Rijsbergen, C.J.v., *Information Retrieval*. 1975: Butter Worths.
90. Rijsbergen, C.J.V. and W.B. Croft, *Document Clustering: An Evaluation of Some Experiments with the Cranfield 1400 Collection.* Information Processing and Management, 1975. **11**: p. 171-182.
91. Salton, G., A. Wong, and C.S. Yang, *A Vector Space Model for Automatic Indexing.* Communications of the ACM, 1975. **18**(11): p. 613-620.
92. Schulman, L. *Clustering for Edge-cost Minimization*. in *ACM STOC*. 2000.
93. Seppanen, J.K., E. Bingham, and H. Mannila. *A Simple Algorithm for Topic Identification in 0-1 Data*. in *Knowledge Discovery in Databases: PKDD 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*. 2003.
94. Sharan, R., A. Maron-Katz, and R. Shamir, *CLICK and EXPANDER: A System for Clustering and Visualizing Gene Expression Data.* Bioinformatics, 2003. **19**(14): p. 1787-1799.
95. Sharan, R., A. Maron-Katz, and R. Shamir, *CLICK and EXPANDER: a System for Clustering and Visualizing Gene Expression Data.* Bioinformatics, 2003. **19**(14): p. 1787-99.
96. Shi, J. and J. Malik, *Normalized Cuts and Image Segmentation.* IEEE transactions on pattern analysis and machine intelligence, 2000. **22**(8): p. 888-905.

97. Siersdorfer, S. and S. Sizov. *Restrictive Clustering and Metaclustering for Self-organizing Document Collections*. in *Proceedings of the 27th Annual International Conference on Research and Development in Information Retrieval*. 2004. Sheffield, UK.
98. Siersdorfer, S. and S. Sizov. *Restrictive Clustering and Metaclustering for Self-organizing Document Collections*. in *Annual ACM Conference on Research and Development in Information Retrieval*. 2004.
99. Slonim, N., N. Friedman, and N. Tishby. *Unsupervised Document Classification Using Sequential Information Maximization*. in *ACM International Conference on Research and Development in Information Retrieval (SIGIR)*. 2002.
100. Steinbach, M., G. Karypis, and V. Kumar. *A Comparison of Document Clustering Techniques*. in *Text Mining Workshop, KDD*. 2000.
101. Stork, D.G. and E. Yom-Tov, *Computer Manual in MATLAB to Accompany Pattern Classification*. 2nd Edition ed. 2004: Wiley-Interscience.
102. Tantrum, J. *Hierarchical Model-based Clustering of Large Datasets through Fractionation and Refractionation*. in *Conference on Knowledge Discovery in Data*. 2002. Edmonton, Alberta, Canada.
103. Tantrum, J., A. Murua, and W. Stuetzle. *Assessment and Pruning of Hierarchical Model-based Clustering*. in *Conference on Knowledge Discovery in Data*. 2003. Washington D.C.
104. Theodoridis, S. and K. Koutroumbas, *Pattern Recognition*. 1999: Academic Press.
105. Titterington, D.M., A.F.M. Smith, and U.E. Makov, *Statistical analysis of finite mixture distributions*. Wiley Series in Probability and Mathematical Statistics. 1985: John Wiley & Sons.
106. TREC. *Text REtrieval Conference*. in *http://trec.nist.gov*.
107. Tsioutsiouliklis, K., *Maximum Flow Techniques for Network Clustering*, in *Computer Science Department*. 2002, Princeton University: Princeton.
108. Vaidya, J. and C. Clifton. *Privacy-preserving k-means Clustering over Vertically Partitioned Data*. in *Conference on Knowledge Discovery in Data*. 2003. Washington D.C.
109. Vempala, S. and G. Wang. *A Spectral Algorithm for Learning Mixtures of Distributions*. in *Proc. of the 43rd IEEE Foundations of Computer Science*. 2002. Vancouver.
110. Verma, D. and M. Meila, *A Comparison of Spectral Clustering Algorithms*. 2003, University of Washington, Department of Computer Science and Engineering.
111. Willett, P., *Recent Trends in Hierarchical Document Clustering: A Critical Review*. Information Processing and Management, 1988. **24**(5): p. 577-597.
112. Willett, P., *Similarity coefficients and weighting functions for automatic document classification: an empirical comparison.* International Classification, 1983. **10**: p. 138-142.
113. Xu, W. and Y. Gong. *Document Clustering by Concept Factorization*. in *Proceedings of the 27th Annual International Conference on Research and Development in Information Retrieval*. 2004. Sheffield, UK.
114. Xu, W., X. Liu, and Y. Gong. *Document Clustering Based On Non-negative Matrix Factorization*. in *ACM Conference on Research and Development in Information Retrieval*. 2003. Toronto, Canada.
115. Yeung and Ruzzo, *Principal Component Analysis for Clustering Gene Expression Data.* Bioinformatics, 2001. **17**(9): p. 763-774.
116. Zamir, O. and O. Etzioni. *Web Document Clustering: A Feasibility Demonstration*. in *ACM SIGIR*. 1998.
117. Zha, H. *Generic Summarization and Keyphrase Extraction Using Mutual Reinforcement Principle and Sentence Clustering*. in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2002. Tampere, Finland.
118. Zhang, T., R. Ramakrishnan, and M. Livny. *BIRCH: An Efficient Data Clustering Method for Very Large Databases*. in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. 1996. Montreal, Canada.
119. Zhao, Y. and G. Karypis. *Evaluation of Hierarchical Clustering Algorithms for Document Datasets*. in *CIKM*. 2002. McLean, Viginia.