INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Empirical Observations of Probabilistic Heuristics for the Clustering Problem

Jeff Bilmes*, Amin Vahdat†, Windsor Hsu ‡, Eun-Jin Im §

TR-97-018

May 1997

Abstract

We empirically investigate a number of strategies for solving the clustering problem under the minimum variance error criterion. First, we compare the behavior of four algorithms, 1) randomized minimum spanning tree, 2) hierarchical grouping, 3) randomized maximum cut, and 4) standard k-means. We test these algorithms with a large corpus of both contrived and real-world data sets and find that standard k-means performs best. We found, however, that standard k-means can, with non-negligible probability, do a poor job optimizing the minimum variance criterion. We therefore investigate various randomized k-means modifications. We empirically find that by running randomized k-means only a modest number of times, the probability of a poor solution becomes negligible. Using a large number of CPU hours to experimentally derive the apparently optimal solutions, we also find that randomized k-means has the best rate of convergence to this apparent optimum.

*ICSI and CS Division, Department of EECS, U.C. Berkeley

†CS Division, Department of EECS, U.C. Berkeley

‡CS Division, Department of EECS, U.C. Berkeley

§CS Division, Department of EECS, U.C. Berkeley

1 Introduction

Consider the following problem. A large collection of wooden boards must be separated into groups consisting of the same tree species. The only available tool, however, is a digital camera and a controllable robot. One approach to solving this problem entails parsing each board's image and obtaining characteristic features such as luminance, grain density, texture, mass per unit volume, etc., and then grouping the boards together that have similar features. In general, material world objects can often be described by a list of numerical features, and a classification of these objects can be achieved by grouping together objects whose features are "similar" in some way.

Such tasks are the original motivation behind the CLUSTERING problem. Since these tasks arise occur in practice, the CLUSTERING problem has been given much attention. However, real-world clustering problems are quite varied and many seemingly unrelated problems are essentially CLUSTERING in nature.

- Vector quantization: Given a high dimensional vector space and a data set of vectors, the problem is to identify and enumerate the interesting sections of this space (according to the data set). For computational and other practical reasons, it is advantageous, in subsequent analysis, to identify a data point using its section number rather than the data point itself.
- Grouping: Given a description of a collection of objects, group them together in such a way that "similar" objects are placed in the same clusters. An example of a grouping problem is the board example described above.
- Informational search: given a set of informational sources (such as articles), group them into broad categories (e.g., politics, business, arts, music, etc.). Next, select several of the articles, and further group them into broad sub-categories (e.g., how the Republican takeover affects the arts, policies toward Bosnia, etc.). This process can be continued down to the individual articles.
- Population profile: Deducing socio-economic, political, or behavioral organizations or affinities by grouping together data from respondents to a widely dispersed questionnaire. This type of problem occurs regularly in the social sciences.

A wide class of real-world questions can be reduced to the CLUSTERING problem. Unfortunately, the problem is NP-complete in many of its definitions. Formally, the clustering problem is described as follows:

THE CLUSTERING PROBLEM

INSTANCE: A finite set X, a distance measure $d(x,y) \in \mathcal{R}_0^+$ for $x, y \in X$, two positive integers k and B, and a criterion function $J_e(\mathcal{X}_1, \ldots, \mathcal{X}_k, d(\cdot,\cdot))$ on a k-partition $\{\mathcal{X}_1, \ldots, \mathcal{X}_k\}$ of X and measure $d(\cdot,\cdot)$.

QUESTION: Is there a partition of X into disjoint sets $\mathcal{X}_1, \ldots, \mathcal{X}_k$ such that $J_e(\mathcal{X}_1, \ldots, \mathcal{X}_k, d(\cdot,\cdot)) \leq B$?

OPTIMIZATION: Find the partition of X into disjoint sets $\mathcal{X}_1, \ldots, \mathcal{X}_k$ that minimizes the expression $J_e(\mathcal{X}_1, \ldots, \mathcal{X}_k, d(\cdot,\cdot))$

Observe that the number of k-clusters for a data set of size $n > k$ is given by the recurrence:

$$\begin{aligned} &R(k,n) = kR(k,n-1) + R(k-1,n-1) \quad n > k > 0 \\ &R(k,k) = 1 \\ &R(1,n) = 1 \end{aligned}$$

and has solution [JD88],[DH73]

$$R(k,n) = \frac{1}{k!}\sum_{i=1}^{k}(-1)^{k-i}\binom{k}{i}(i)^n \approx k^n/k!.$$

which clearly can not efficiently be exhaustively iterated in a search procedure.

The problem remains NP-complete regardless of the distance measure $d(\cdot,\cdot)$ used [B78], [GJ79]. Furthermore, there are various versions of J_e for which the problem remain NP-complete including:

1. $J_e(\mathcal{X}_1,\ldots,\mathcal{X}_k,d(\cdot,\cdot)) = d(x,y)\ \forall x,y \in \mathcal{X}_i\ \forall i = 1\ldots k$. This criterion function asks that no two points in any cluster be greater than B.

2. $J_e(\mathcal{X}_1,\ldots,\mathcal{X}_k,d(\cdot,\cdot)) = \sum_i \max_{x,y\in\mathcal{X}_i} d(x,y)$. This criterion function asks that the maximum intra-cluster point distance summed over all clusters be less than B.

3. $J_e(\mathcal{X}_1,\ldots,\mathcal{X}_k,d(\cdot,\cdot)) = \max_{x,y\in\mathcal{X}_i,\,\forall i} d(x,y)$. This criterion function asks that the maximum distance between any two points in any cluster be less than B. Under this J_e and a distance *metric* (i.e., $d(\cdot,\cdot)$ satisfying the triangle inequality), it has been shown [HS86] that the problem is efficiently approximable within 2, but is not efficiently approximable within $2-\epsilon$ for any $\epsilon > 0$.

4. $J_e(\mathcal{X}_1,\ldots,\mathcal{X}_k,d(\cdot,\cdot)) = \sum_i \sum_{x,y\in\mathcal{X}_i} d(x,y)$. This criterion function asks for the sum of the intra-cluster point distances to be less than B with a penalty for many points in a cluster. It has been shown [SG78] that under this J_e, the problem is not ϵ-approximable unless P=NP. That is, we can not find an efficient algorithm such that $|(J_e^{opt} - J_e^{alg})/J_e^{opt}| \le \epsilon$ for any ϵ unless P=NP, where J_e^{opt} is the optimal cost and J_e^{alg} is the result of the algorithm.

A closely related criterion function is $J_e(\mathcal{X}_1,\ldots,\mathcal{X}_k,d(\cdot,\cdot)) = \sum_i \frac{1}{|\mathcal{X}_i|}\sum_{x,y\in\mathcal{X}_i} d(x,y)$. This one reduces the penalty for clusters that contain many proximal points. In this paper, we chose to investigate clustering under this criterion function because it is algebraically equivalent to:

$$J_e(\mathcal{X}_1,\ldots,\mathcal{X}_k,d(\cdot,\cdot)) = 2\sum_i \sum_{x\in\mathcal{X}_i} d(x,\mu_i)$$

where

$$\mu_i = \frac{1}{|\mathcal{X}_i|}\sum_{x\in\mathcal{X}_i} x$$

is the ith intra-cluster mean.

This criterion function, called the sum-of-squared-error or minimum variance criterion [DH73], is the most widely used in practice. Attempts to optimize it correspond to finding a set of well-separated clusters whose intra-cluster distances are small. Unfortunately, it has not yet been proven that under this J_e the problem remains NP-complete. Nevertheless, there are no known polynomial-time algorithms that provide the optimal solution, and intuitively, it appears to be as difficult as the other J_e functions. Therefore, we chose not to sacrifice pragmatics and used the minimum variance criterion for our investigation.

In this paper, we report on our experimentation with various heuristics for minimizing the clustering problem under this J_e criterion. We implement three algorithms found in the literature: a hierarchical method, minimum spanning tree approximations, and k-means. We also develop and implement a fourth algorithm called Max-Cut. Using a large corpus of both contrived and real-world data sets, we compare the relative performance of these algorithms. In practice, k-means produces the best results most quickly. Furthermore, it is simplest both in coding effort and computational complexity. We show that adding randomness to k-means can significantly improve its performance. Using approximately 3 CPU-months to determine apparently near-optimal clusterings, we compare the performance of each of these algorithms relative to the best solution found.

We find that the CLUSTERING problem may be easier to approximate than other NP-Complete problems since elements of the solution space tend to converge to a small number of discrete candidate solutions. This result indicates that running k-means on the CLUSTERING problem with a relatively small number of random executions is likely to produce near-optimal results in practice.

The rest of this paper is organized as follows. In section 2, we present an overview of previous work done for the CLUSTERING problem and describe the algorithms which we chose for experimentation. Section 3 describes the data sets used to evaluate the performance of the various algorithms. Section 4 quantifies the performance of the algorithms on these data sets. Section 5 describes empirical results regarding the convergence of of the most promising of the algorithms, and section 6 describes our efforts to improve its performance. And lastly, section 7 outlines future work and presents our conclusions.

2 Background

There are a large collection of approximation algorithms for the clustering algorithm [JD88], [DH73],[DO74],[A73], [H75],[DJ87],[Z71],[T89], each with performance sensitive to particular classes of data sets. Many of these algorithms were designed for the problem of detecting gestalt clusters, i.e., finding those data point collections (often in 2 or 3 dimensions) that, when plotted in some way, tend to perceptually "look" like they should be together. Because we are not interested in detecting gestalt clusters, we choose four algorithms we believed to show promise for the optimization of the minimum criterion function. Those algorithms include i) the method of minimum spanning tree, ii) maximum cut, iii) a hierarchical method and iv), the k-means algorithm. These algorithms are described in the following subsections.

2.1 Minimum Spinning Tree

The version of the minimum spanning tree algorithm presented in the literature [Z71] works as follows:

- Construct a complete graph with each element as a node and inter-element distances as edge weights.
- Compute the minimum spanning tree of the resulting graph.
- Choose the k longest edges in the minimum spanning tree and delete them.
- The remaining k connected components form the k clusters in the solution.

Upon implementing this algorithm, we discovered that while it produced good gestalt clusterings, it tended to perform quite poorly on the minimum variance criterion. For example, if the two dimensional data input consisted of four long thin lines, this algorithm separates each line into its own cluster. While this may be the visually natural, the resulting J_e could be quite high since data points may be quite distance from their cluster mean.

Randomness was introduced to this algorithm to improve the clusterings. We randomly chose k points from the minimum spanning tree, and deterministically grew k clusters by picking the nearest neighbors from each of the k clusters. The clusters were grown until all points had been clustered. By running the algorithm repeatedly, we found that the algorithm performed reasonably well (as shown in the next section). Unfortunately, the algorithm is computationally expensive ($O(n^2)$), and quite sensitive to the initial points chosen for the clusterings.

2.2 Maximum Cut

The *Maximum Cut* method, one we developed ourselves, consists of the following steps:

- Choose k elements as the initial k clustering.
- Arbitrarily pick one of the remaining elements and place it in the cluster with the nearest mean.
- Repeat until all the elements have been assigned clusters.

On each iteration, Maximum Cut places a element into the cluster that currently has the nearest mean. This is equivalent to greedily maximizing the inter-cluster distances, i.e., the sum of distances between elements in different clusters. When k=2, this scheme is a greedy approach to solving the standard maximum cut problem. In this sense, this algorithm attempts to solve a generalization of the standard maximum cut problem, hence its name.

Observe that once a element has been assigned to a cluster, Maximum Cut will never reassign it to another cluster. Since a element is assigned to a cluster on each iteration, Maximum Cut takes time linear in the number of elements. However, since the maximum cut algorithm does not ever reassign elements, the algorithm is unable to recover from poor initial decisions.

One way to alleviate this problem is to run Maximum Cut several times with random initial clusterings. Intuitively, this will perform better when the optimal clusters are of about the same size – in this case, there is a higher chance that the initial k elements chosen at random will be from different optimal clusters. For n elements and k equal size clusters, the probability that the clusters in a random initial clustering will each contain a element from a different optimal cluster is at least $k!/k^4$. On the other hand, if there are $k-1$ clusters of 1 element each and 1 cluster of $n-k+1$ elements, the probability that a random initial clustering is good drops to about $k!/n^{(k-1)}$. However, such pathological cases are unlikely to occur in real life and thus Maximum Cut performs rather well in practice.

2.3 Hierarchical Method

The hierarchical method is a deterministic algorithm consisting of the following steps.

- Assuming n elements and an $n \times n$ distance matrix, partition the elements into n clusters each containing 1 element.
- The algorithm consists of a number of iterations, each of which reduces the total number of clusters by one. Thus, the algorithm is repeated until the desired number of clusters are produced.
- During each iteration, the two clusters which are closest together (as determined by the distance matrix) are merged to form a new cluster. The distance matrix is then updated to include the distance from this new cluster to all other existing clusters. A number of measures used for computing the distance matrix may be used.

1. MIN: for each cluster, take the minimum of the distances to the two merged clusters.
2. MAX: for each cluster, take the maximum of the distances to the two merged clusters.
3. AVG: for each cluster, take the average of the distances to the two merged clusters.

Unfortunately, the algorithm is quite sensitive to the measure used for updating the distance matrix when clusters are merged. Furthermore, none of these measures produces the best results in all cases. In fact, for some of our data sets the three measures produce results which vary by an order of magnitude. In general, MAX tends to produce the best results because it optimizes against very spread out clusters.

One advantage of this algorithm is that it produces clusterings for all values of k in one pass. Thus, if the target number of clusters is not known ahead of time, the hierarchical method can pick the clustering which minimizes J_e during one execution of the algorithm. On the negative side, the algorithm is hindered by the fact that it cannot recover from early errors: once a clustering assignment is made, it cannot be changed. Further, the algorithm is fairly expensive since in practice it requires significant bookkeeping effort to achieve efficient execution (it is $O(n^2)$ in either space or time).

2.4 K-Means

The k-means algorithm was developed specifically for the optimization of the minimum variance criterion [T89,M67]. The following describes one execution of the k-means algorithm.

```
Step 1: Choose an arbitrary $k$-clustering.
Step 2: Compute mean of each cluster.
Step 3: Assign each element to the cluster with the closest mean.
Step 4: If any elements have moved in step 3, goto step 2.
Step 5: Output the resulting clustering.
```

Needless to say, one main advantage of k-means is that it is designed specifically to optimize our J_e. Further advantages include that it can recover from earlier errors, and that convergence to a stable state is guaranteed [M67]. See section 5 for our empirical results regarding convergence. Furthermore, the algorithm is relatively simple to understand and implement, and it runs in $O(n)$ time and space complexity.

3 Data Sets

In order to test the performance of the four algorithms described above, we used a suite of both contrived and real-world data sets. The contrived data sets comprise Gaussian clusters, curves (both regular and Gaussian smeared), a number of "optimum" data sets where the best clustering is obvious, and shapes (both regular and Gaussian smeared). Figure 1 pictorially shows the 16 data sets and the following table provides the data set complexities.

	gaussian1	gaussian2	gaussian3	gaussian4	gaussian5	horiz1	lines1	lines2
Dimension	2	2	2	2	2	2	2	2
Samples	2000	2000	2000	2000	2000	2000	1010	2324
	optim1	optim2	shapes1	shapes2	smile	unif1	unif2	spread1
Dimension	2	2	2	2	2	2	2	2
Samples	2000	4000	1202	3005	564	1000	2000	2000

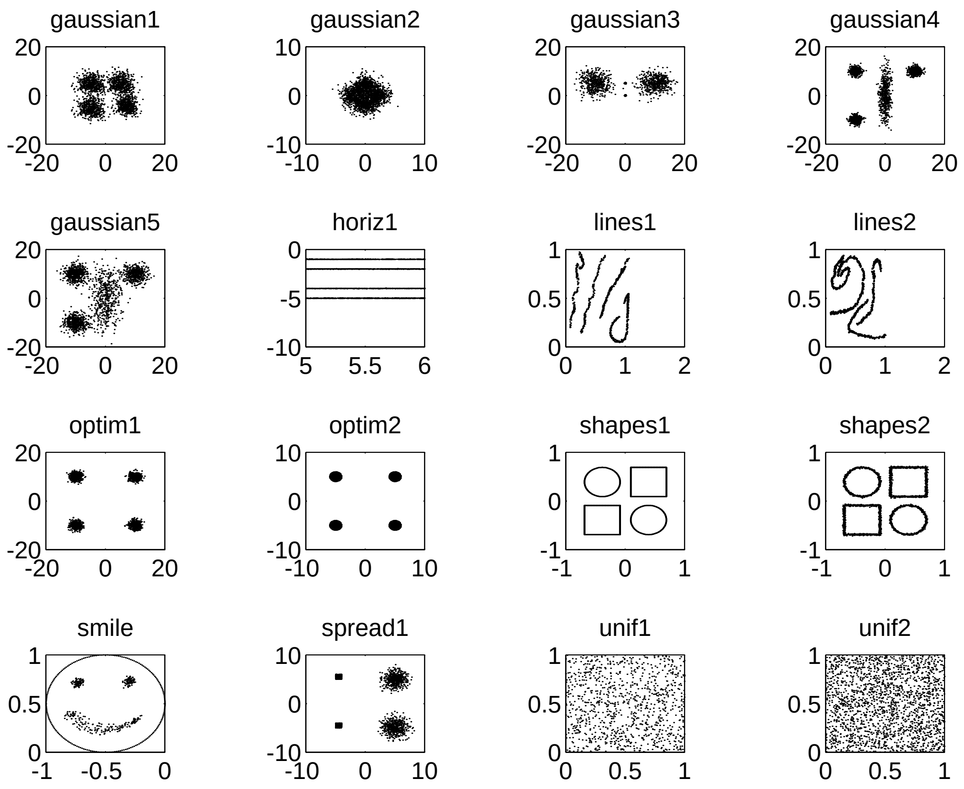


Figure 1: *Summary of the 16 contrived data sets used to test the performance of the heuristics.*

Random contrived data sets are not sufficient to establish a relative performance baseline between the different algorithms. Thus, a number of "real-world" data sets were used to determine how the algorithms might perform in practice. This second set of data consists of:

- Audio feature vectors generated by a short-time constant-Q filter bank approximation over segments of a musical sound signal. The sound data files consist of 1652, 936, and 936 sample points of 28, 31, and 31 dimensions respectively.

- Speech data consisting of PLP feature vectors from frames (short time segments) of sampled speech data. The speech data files consist each of 3000 sample points of 31, 30, and 30 dimensions respectively.

- Data consisting of feature vectors that describe textures obtained by parsing still pictures [1] The data set complexities are described in the following table:

	fabric_sar	faces	pieces_healey	pieces_sar	sar	tamura	tev
Dimensions	15	20	6	15	5	3	99
Samples	720	7561	6272	6272	1008	1008	1008

Each of the data sets were input to the four algorithms, producing a final four-clustering and an associated J_e. Figure 2 shows a sample data set and its clustering. The left side of the figure shows the input to the k-means approximation algorithm, while the right side shows the final clustering.

[1]We wish to thank Thomas P. Minka (tpminka@media.mit.edu) for the use of his visual data sets.

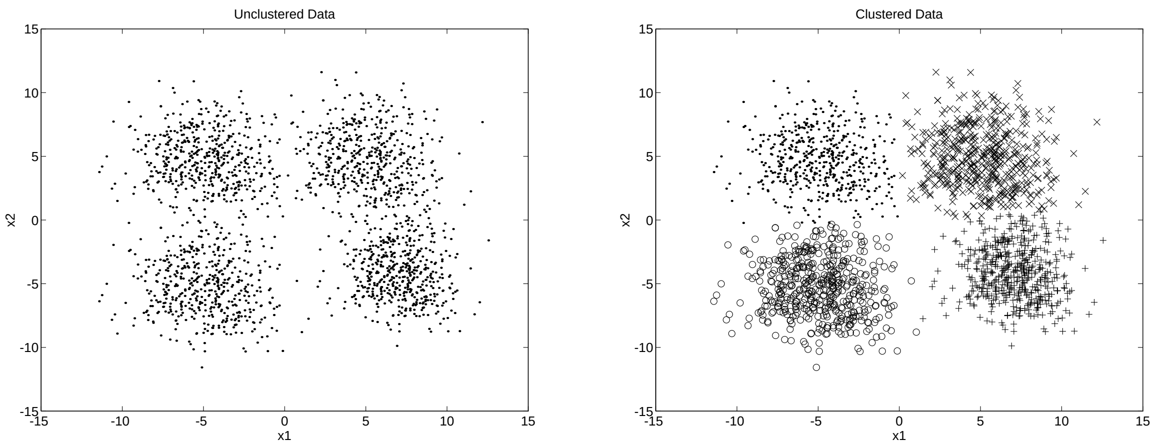


Figure 2: An example of partitioning random Gaussian clusters into four clusters. On the left we see the data input to the clustering algorithm, while the right shows an example clustering.

4 Initial Results

After implementing the four algorithms, we ran our test suite over each one. The randomized algorithms (Max-Cut and minimum spanning tree) were run 2000 times, taking the best result seen to that point as the final J_e. Figure 3 graphically depicts the relative performance of the four algorithms on our data sets. For each data set, the minimum J_e produced by any of the four algorithms is normalized to 1; all other values are shown as a multiple of this best result. In the graph, the relative J_e is plotted along the y-axis. The data sets are shown along the x-axis, while the z-axis spans the four algorithms.

Figure 3 clearly shows that the k-means algorithm uniformly produces the best results. All the algorithms perform reasonably well in comparison to k-means, but show no appreciable advantage in any category. As expected, the minimum spanning tree algorithm performed poorly on input sets (such as lines) where obvious gestalt clusterings were present. The Max-Cut is quite similar to the k-means algorithm and thus showed comparable performance. However, unlike the k-means algorithm, a single execution of it has no capacity to reassign clusterings once initial decisions are made and is therefore more sensitive to initial randomized starting points. The hierarchical algorithm does not contain any randomness and has no way of recovering from early decision errors. Thus, it shows relatively poor performance.

Finally, none of the related algorithms are more efficient or simpler to implement than k-means. The hierarchical method is computationally expensive and cannot recover from early errors in clustering decisions. Both the minimum spanning tree and the maximum cut algorithms perform reasonably relative to k-means, however they are more sensitive to the initial choice of k points, even when run multiple times with different random starting points. The k-means algorithm appears less sensitive to its initial clustering than the other three algorithms.

5 Convergence of K-Means

This section briefly describes empirical results regarding the convergence of the k-means algorithm—the number of k-means iterations until it converges to a final clustering.

Figure 4A shows a histogram of the k-means iterations until convergence across all the data sets. The figure shows a general trend toward a small number of iterations until convergence.

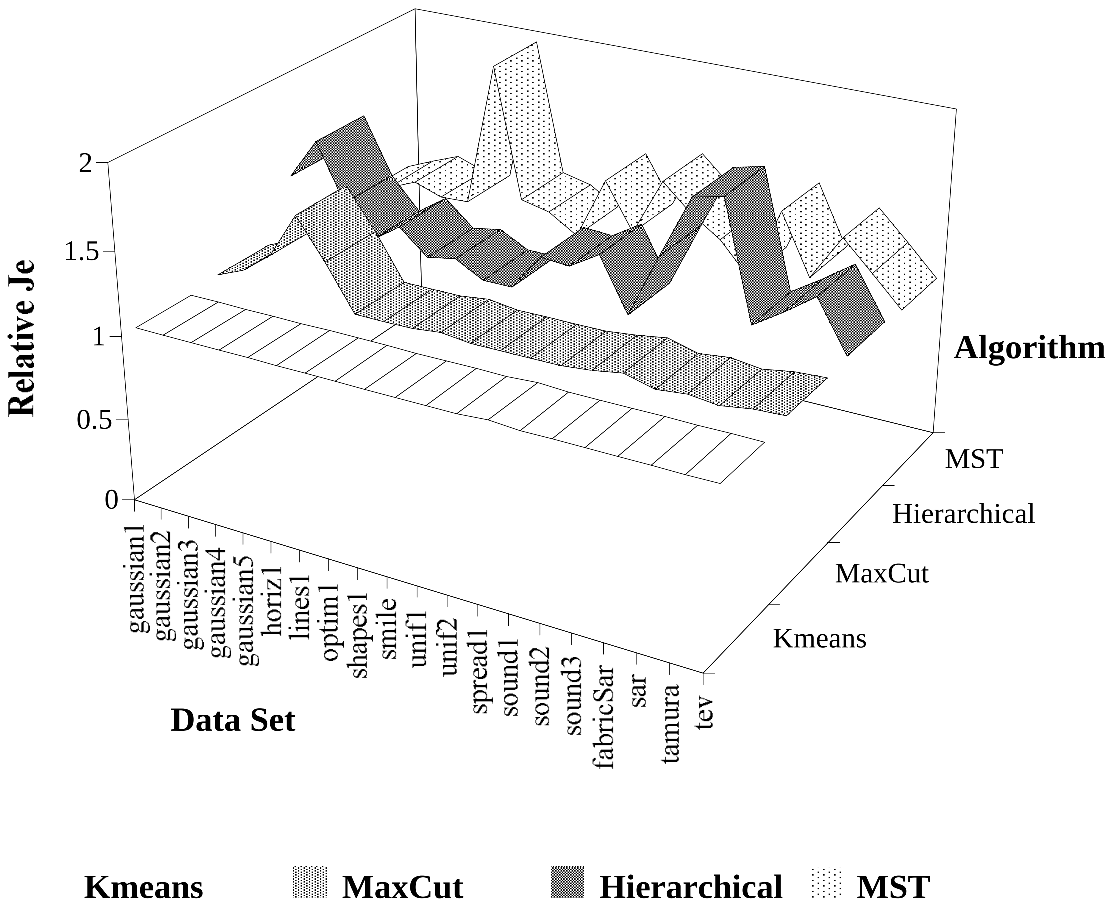


Figure 3: *Performance of the four algorithms on our data sets.* The k-means algorithm uniformly shows the best performance. Max-Cut also performs well, while minimum spanning tree and the hierarchical method perform relatively poorly.

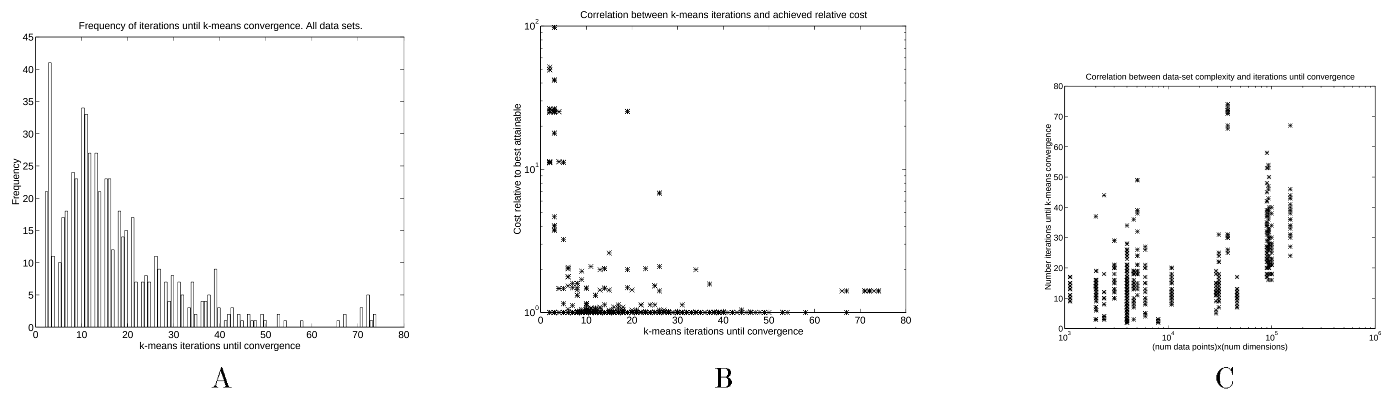


Figure 4: Empirical analysis of k-means convergence.

Figure 4B shows the correlation between the iterations until convergence and the achieved relative J_e for each execution. The plot shows that the high relative costs result only when the number of iterations until convergence is small – a large number of iterations most often means that we are headed towards a relatively good solution.

Figure 4C shows the correlation between the complexity of the data sets and the number of iterations until convergence. The data set complexity is defined as the product of the number of data points with the number of dimensions. The plot shows a small positive correlation between complexity and iterations, but the correlation is small. In general, it appears that we can assume that k-means converges quickly regardless of the data-set complexity.

6 Probabilistic K-Means

We have already seen that k-means shows the best overall performance of the trial algorithms. Running k-means only once however might not always lead to good solutions. In fact, even k-means can be quite sensitive to the initial arbitrary clustering. Thus, k-means can provide a solution that is significantly worse than the best it can do, as Figure 5 demonstrates.

Consider `optim1` in Figure 5. As can be seen from Figure 3, `optim1` was designed to have clearly defined optimum and a very low J_e, about 40. The k-means algorithm achieves this J_e about 35% of the time. However, about 16% of the time, k-means results in a J_e that is 4000, a factor of 100 off from the optimum. This result follows from choosing a symmetrical initial clustering that converges very quickly to the final solution. Consequently, it could be a mistake to run k-means once and accept the resulting clustering.

The other examples in Figure 5 are similar. `speech1` is typical of most of the data sets, a high probability to achieve close to the apparent optimum. `speech2` and `speech3` shows a tendency to land at about 4 or 5 points, with the *best being less likely then other points*, very disconcerting for one-run k-means performance. `pieces_haeley` is particularly strange because there are two points, the worst of which has over an 80% chance of being achieved. Finally, `tev` shows almost a Gaussian distribution centered at something that is not the optimal. We believe that k-means has its own idea of what is optimal (something near this center), whereas the real optimum is somewhere to the left of the leftmost point.

To summarize, to achieve a better clustering, it is crucial to run k-means many times with different initial clusterings. The following subsections describe variations on introducing randomness into the k-means algorithm and empirically determines the necessary number of executions with different initial clusterings.

6.1 Heuristics

We begin by describing two heuristics used for multiple executions of the k-means algorithm:

- Memoryless: Run k-means multiple times. On each k-means execution, uniformly at random choose an initial cluster. Output the clustering that achieves the lowest J_e.

- Retention: Run k-means multiple times. On each k-means execution, choose an initial cluster using the p-perturbation rule. Starting with the best clustering seen so far, choose each sample point with probability p, the perturbation probability. If a sample is chosen, pick a cluster from 1 to k uniformly at random, and move the sample point to that cluster. Output the clustering that achieves the lowest J_e.

  We experiment with two variations to this algorithm:

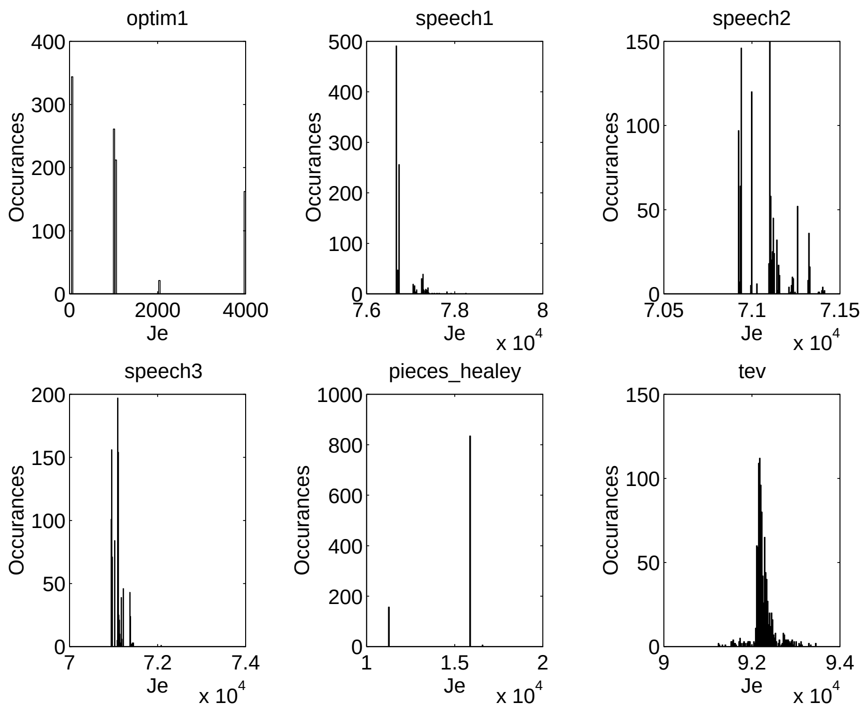


Figure 5: Histograms of J_e for the data sets with the greatest (i.e., worst) J_e variance. 1000 executions per data set, each execution uses a different initial cluster.

– Fixed p: p is fixed across all executions.

– Linearly-decaying simulated annealing: Start p at some initial value, and then decrease it linearly in the number of iterations.

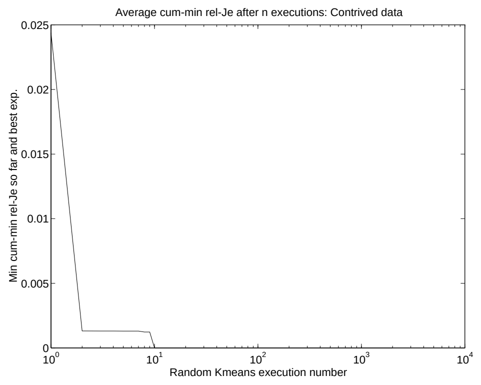
In subsequent analysis, we will refer to these three variants as the *Memoryless*, the *Fixed-Retention*, and the *Decaying-Retention*[2] methods.

6.2 Discovering The Optimum J_e

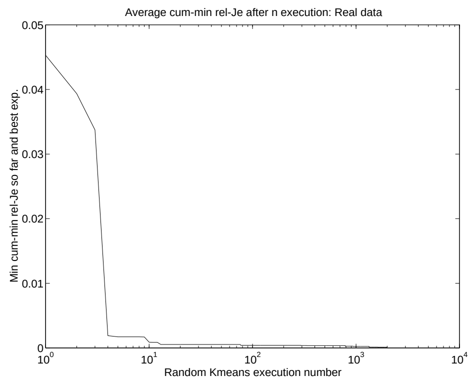
Since the input data sets are large, the optimal solution is not known. Several of the contrived data sets (`optim1`, `optim2`, and `spread1`) were designed specifically such that the optimal clustering is obvious by inspection. Unfortunately, clusterings obvious to human inspection appears to also be obvious for k-means – for these data sets, k-means usually found the optimum immediately. Therefore, an estimate of the optimum clustering was derived.

For each data set, when considering the different algorithmic variants, the underlying k-means algorithm was run a total of 41,000 times, each with a different initial clustering. Our estimate for the optimal solution for each data set is the best out of those 41,000 independent executions. Observe that running k-means this many times for a reasonably sized data set is computationally impractical. In fact, for the larger data sets, 41,000 executions of k-means took about 24 CPU-hours of compute time on a Sparc-2 workstation. The minimum of the 41,000 executions on each data set we will denote by $\hat{J}_e^{opt}$.

[2]This variant might be called *Increasing-Retention* because as p decreases, the algorithm tends to use previously found clusters more strongly. Regardless, we chose the word "decaying" to indicate that p is decreasing.



A: Contrived



B: Real

Figure 6: Average cumulative minimum relative J_e plots. 2000 executions.

In order to compare the performance of the different algorithms on different data sets, we compare the relative error. That is, we look at the relative J_e defined as $|J_e - \hat{J}_e^{opt}|/\hat{J}_e^{opt}$, where J_e is the minimum variance criterion output of a k-means execution.

6.3 Determining Needed Iterations

In this section, we empirically determine the number of executions of the randomized k-means methods necessary before we get "close" to the computed optimum estimate. We define "close" as a relative error less than or equal to 10^{-7}. This number is chosen because once the cumulative minimum of the relative J_e is within this bound, it either would be zero, or would stay where it is for the remaining executions. This empirical method is akin to finding the point at which we are fairly certain that with high probability we are within a small percentage of the best answer k-means is likely to to ever find (at least within 41,000 executions).

Figure 6 shows the results of the average cumulative minimum of the relative J_e plots for 2000 executions averaged over the contrived and the real data sets separately.

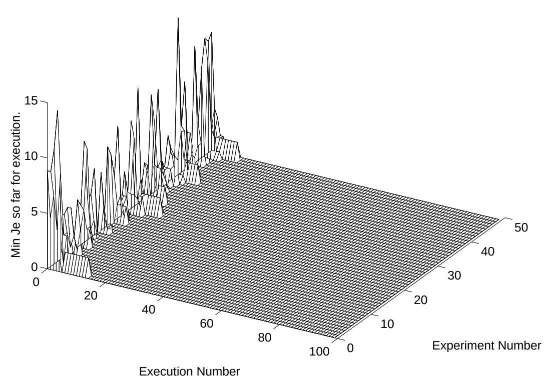
Clearly, in both the real and contrived data, by 100 executions, we are very close to the computed minimum. For the contrived example, we are within our desired 10^{-7} bound, and for the real example, the decay after 100 executions is negligible compared to that before 100 executions. Therefore, in all subsequent analysis we use experiments consisting of 100 executions.

6.4 Comparison of the Three Methods

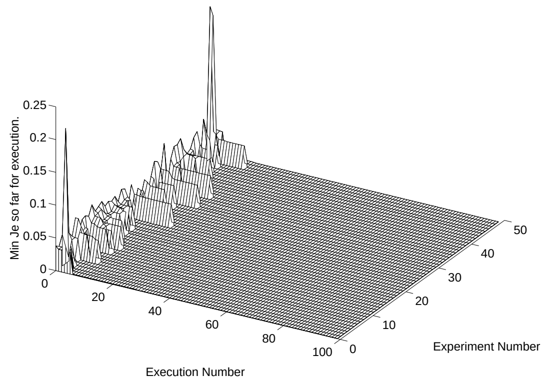
In this section we compare the performance of the three randomized heuristics mentioned above.

For each data set, we perform an *experiment* 50 times. Each experiment consists of running the particular randomized k-means for 100 *executions* and calculating the cumulative minimum (the minimum seen so far) of the relative J_e. Therefore, each particular randomized variant of k-means is run a total of 5000 times. Finally, the resulting matrices are divided into separate groups: contrived and real-world. Within each group, the results are averaged and displayed as the surface plots shown below.

Figure 7 shows the results for the memoryless randomized algorithm. For both the real and contrived data sets, some initial noise quickly becomes insignificant (about 10^{-7}). Further, no experiment required more than 30 executions to achieve this error, independent of the data set. Therefore, running memoryless randomized k-means a modest number of times can greatly increase the chances of obtaining the best result likely to be obtained, at least within 41,000 executions.

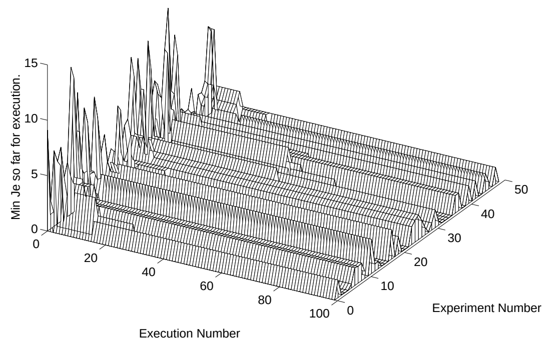


A: Contrived

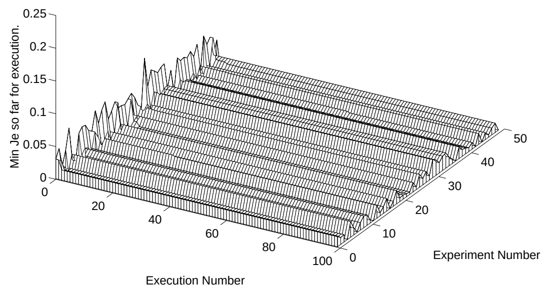


B: Real

Figure 7: Performance of memoryless randomized k-means algorithm. The x-axis shows the number of executions of the algorithm. The y-axis demonstrates the error relative to the minimum scene after 41000 executions (roughly a day) of the algorithm, while the z-axis shows the experiment number. For this graph, 50 experiments of 100 executions each were executed.

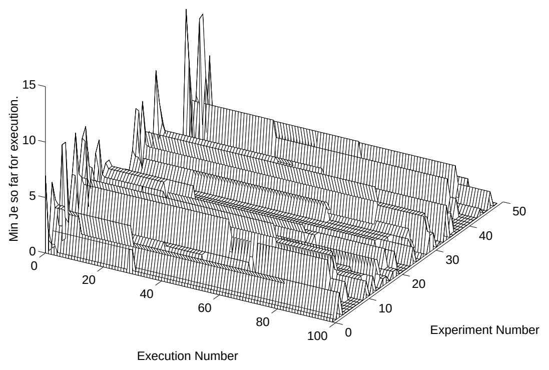


A: Contrived

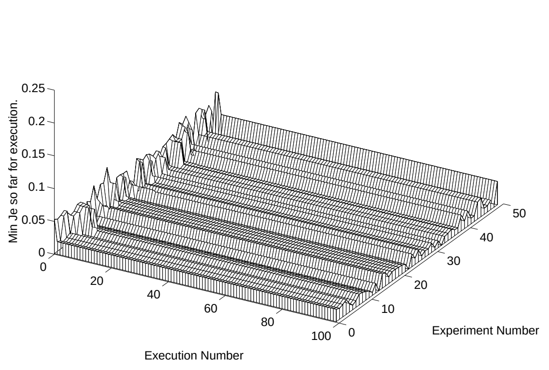


B: Real

Figure 8: Performance of fixed-retention $p = 0.1$ randomized k-means algorithm.



A: Contrived



B: Real

Figure 9: Performance of decaying-retention randomized k-means algorithm. Initial p is 0.4.

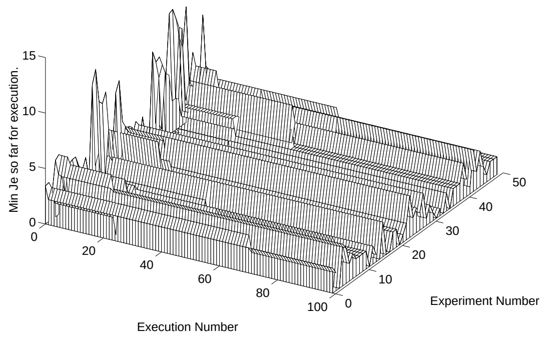
Figure 8 shows a similar plot for the fixed-retention strategy where p, the perturbation probability, is fixed at 0.1. Observe that the scales of Figure 8 and Figure 7, and in fact all subsequent surface plots, are identical, so if something looks large in one, it is large in another. As can be seen, the fixed-retention strategy with $p = 0.1$ never quite achieves the same relative error on average as the memoryless strategy. An explanation for this behavior is that with such a small perturbation probability, there is a low likelihood of escaping from local minima. That is, once we are at a reasonable but not optimal solution, a perturbation probability of 0.1 is not large enough to get us out of the valley in the solution space. In later examples, we experiment with a larger fixed perturbation probability.

Figure 9 shows the plot for the decaying-retention strategy with the starting perturbation probability equaling 0.4. The plot looks slightly worse than fixed-retention for the contrived data, and about the same for the real-world data. Our explanation is similar – a perturbation probability of 0.4 or less is not high enough to get us out of local minima. This implies, backing up evidence provided by Figure 5, that the space consists only of a small number of attractors with very large surrounding valleys. This seems true regardless of the data set. Thus, it is best to explore the entire solution space rather than further exploring previously discovered valleys.

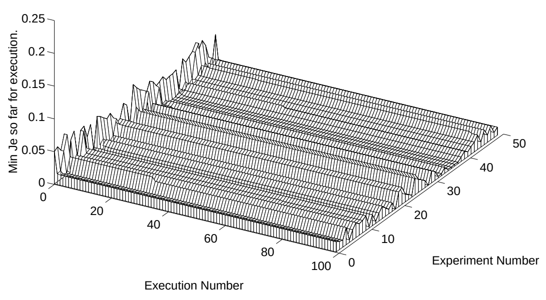
As a quick aside, a common question regarding these plots is why do they at times look so discrete? The primary reason is that in all cases, many of the data sets have achieved a relative error that is essentially zero by an early execution number. The jumps are caused in a particular experiment by a data set with a lingering non-zero relative error. Because of the large number of relative errors already zero, a sudden jump in the relative error for such a data set can produce a large jump in the average, and therefore the graph.

The key to these graphs is considering an execution number across all experiments. If at a particular execution, say n, all experiments show no relative error, then all the data sets at that execution number have a zero relative error. We can therefore be confident that after n executions of that particular algorithm, we have achieved a zero relative error.

The results of these plots seem to demonstrate that the memoryless k-means outperforms the other two, at least when using the relative J_e achieved after 100 iterations. But, we have yet to try larger perturbation probabilities. Certainly when p is fixed at 1, we should do as well as the memoryless strategy since they are equivalent. This notion is further explored in the next subsection.

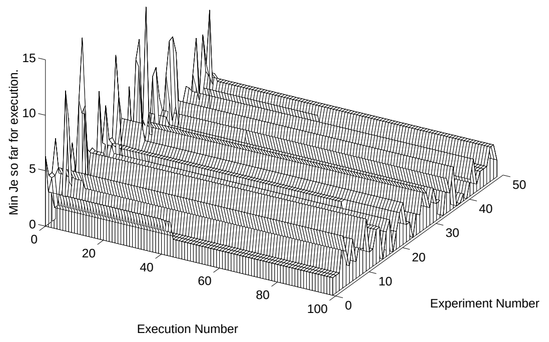


A: Contrived

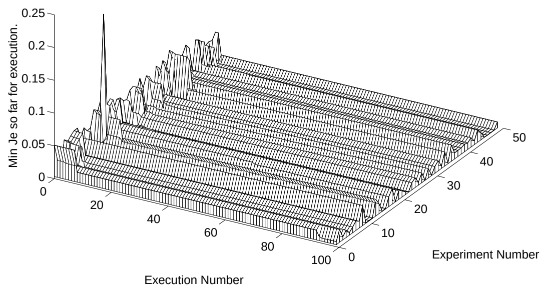


B: Real

Figure 10: Performance of fixed-retention randomized k-means algorithm. p is 0.25.



A: Contrived



B: Real

Figure 11: Performance of fixed-retention randomized k-means algorithm. p is 0.5.

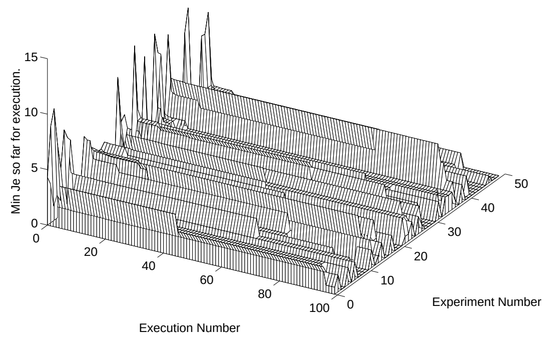
6.5 Different Values of p

We now compare the results of the fixed-retention randomized k-means algorithm with perturbation probabilities equaling 0.25 and 0.5 and the decaying-retention method with initial perturbation probabilities equaling 0.75 and 1.0.

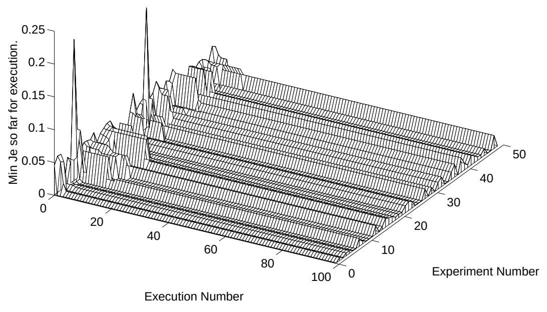
Figure 10 shows the performance of the fixed-retention strategy, where p is fixed at 0.25. The graphs show no obvious improvement from the case where $p = 0.1$, and in fact looks worse for the contrived data. Increasing p even further, Figure 11 shows the performance of the fixed-retention strategy when $p = 0.5$. Still, no obvious improvement is present, although the contrived data looks slightly better.

Figure 12 shows the performance of the decaying-retention strategy where the initial p is 0.75. Immediately, we see no obvious improvement over the previous fixed-p attempts. Figure 13 shows the results when the initial p is 1.0. Observe that when $p = 1.0$, the retention strategy degenerates into the memoryless strategy. The results look better, and in fact are second only to Figure 7, the memoryless strategy. Clearly, this must be owing to the large initial p which allowed the algorithm to “climb out” of any initial local minima. But as the results of Figure 13A show, even this method is not quite as good as the simple memoryless strategy.

Although we haven't exhausted the space of p values, it seems likely that the memoryless

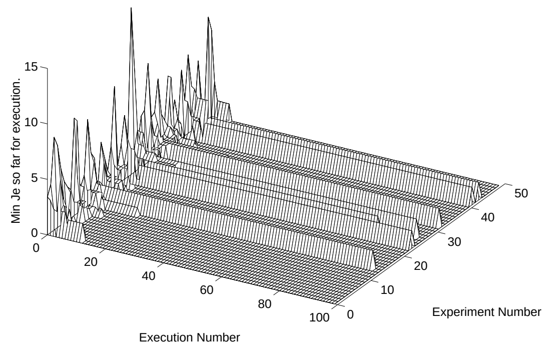


A: Contrived

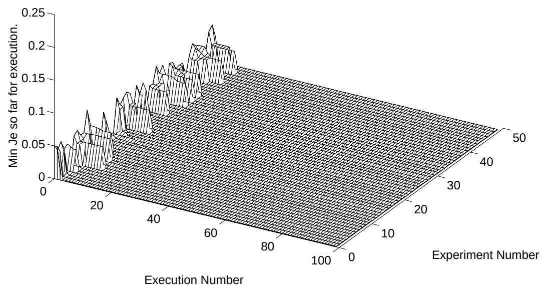


B: Real

Figure 12: Performance of decaying-retention randomized k-means algorithm. Initial p is 0.75.



A: Contrived



B: Real

Figure 13: Performance of decaying-retention randomized k-means algorithm. Initial p is 1.0.

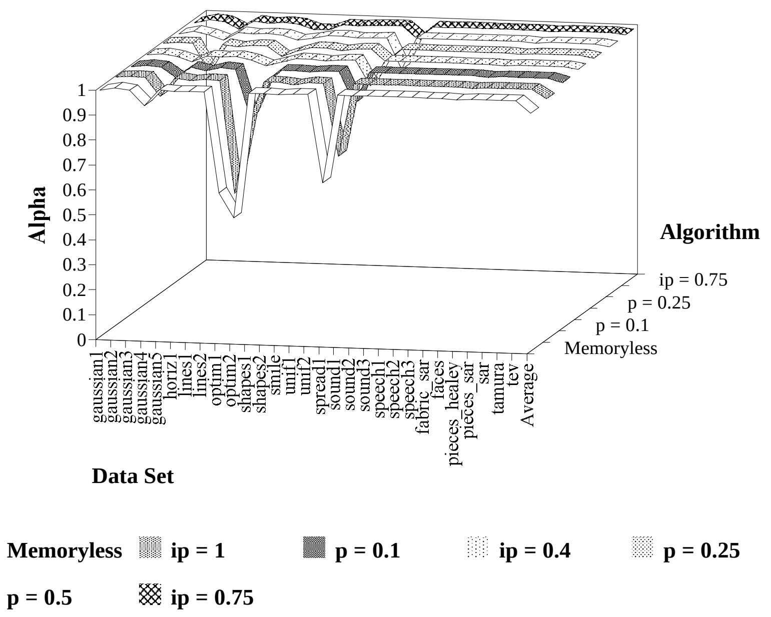


Figure 14: Graph of data values $\alpha = e^{-1/\tau}$ for auto-regressive fit of a function of the form $e^{-t/\tau}$

strategy is best. Thus, using the previously achieved results to generate new initial clusterings only hampers our progress in exploring the remainder of the space. Furthermore, relatively distant local minima are present in the initial cluster space; otherwise, the small perturbations would have allowed us to escape from these local minima.

6.6 Rate of Convergence to Best Solution

Although it is important to ultimately achieve a small relative error, an additional useful criterion for judging the relative merit of each of these algorithms is the rate of convergence to the apparent optimum. If, for example, we are interested in using a clustering algorithm in a time-critical application (such as real-time video), we might perhaps choose an algorithm that converges slightly faster and be willing to sacrifice, if necessary, the quality of the final solution.

Figure 14 compares an estimate of the rate of convergence of each of our probabilistic methods. For each algorithm, we find the average for each execution across the experiments of both the real-world and contrived data sets. Admittedly, this could "smooth" away interesting information. Nevertheless, we choose to simplify the display of what was already a very large amount of data.

Once we have these averages, we wish to find the value τ such that the curve $y(t) = e^{-t/\tau}$ best fits the data averages. We do this using the auto-regressive all-pole method of linear prediction [J89] rather than non-linear least squares because the former is much simpler to implement than the latter, and the results are comparable. Because of the enormous variations in the computed τ values, rather than τ, the plots show $\alpha = e^{-1/\tau}$, a value necessarily between zero and one because τ is always positive. A smaller τ (correspondingly a smaller α) indicates faster convergence.

Figure 14 shows a large drop in convergence time for some of the data sets. In fact, the drop corresponds to the three contrived data sets (`optim1`, `optim2`, and `spread1`, see Figure 1) designed specifically to exhibit an obvious optimum. It is not surprising that all of the k-means algorithms converge quickly for these particular data sets.

Ignoring the large dips, we see that the memoryless method tends to converge slightly faster than any of the other methods. So, not only does the memoryless method achieve better solutions, it converges faster to that solution. Clearly, the memoryless method is superior overall.

7 Conclusions

We have shown that the application of randomness to some well-known approximation heuristics for the CLUSTERING problem can be quite effective in producing better results. In particular, the k-means algorithm shows good promise as an approximation algorithm. However, the algorithm can be sensitive to the choice of initial clustering—sometimes producing results 100 times worse than the optimal. We improved the deterministic version of the algorithm by choosing initial clusterings for the algorithm uniformly at random and running the algorithm repeatedly. We demonstrate that k-means quickly converges to the best result it is likely to achieve. For our data sets, the error criterion was minimized within only 30 executions of the algorithm.

As a further note, the CLUSTERING problem might be easier to approximate than other NP-complete problems because of the structure of the solution space – i.e., it seems like the solution space contains only a small number of attractors (minima) of which the k-means algorithm is strongly inclined to approach. Thus, the space of initial clusterings need not be explored in its entirety. It seems to consist of a small number of large "basins". Given such a structure to the solution space, k-means appears to produce reasonably good solutions in a small number of executions.

Ideally, one would develop a probabilistic approximation theorem for the k-means algorithm. That is, a theorem that for any $\epsilon, \delta > 0$ gives the expected number of random executions of k-means necessary to get an approximation within ϵ with probability greater than $1 - \delta$. The data we have presented, therefore, should be confirmed by the theory, and vice versa.

References

[GJ79] M.R. Garey and D.S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," W.H. Freeman and Co, 1979.

[B78] P. Brucker, "On the complexity of clustering problems," in R. Henn, B. Korte, and W. Oletti (eds.), Optimization and Operations Research, Lecture Notes in Economics and Mathematical Systems, Springer, Berlin 157.

[HS86] Hochbaum, D. S., and Shmoys, D. B. (1986), "A unified approach to approximation algorithms for bottleneck problems", J. ACM 33, 533-550.

[SG78] Sahni, S. K., and Gonzalez, T. (1976), "P-complete approximation problems", J. ACM 23, 555-565.

[DO74] B.S. Duran and P.L. Odell "Cluster analysis; a survey", Lecture notes in economics and mathematical systems, 100, Berlin, New York, Springer-Verlag, 1974.

[JD88] Jain, A.K. and R.C. Dubes, "Algorithms for Clustering Data." Englewood Cliffs, N.J.: Prentice Hall, 1988

[A73] Michael R. Anderberg, "Cluster Analysis for Applications." Probability and mathematical statistics, 19. Academic Press, 1973.

[H75] John A. Hartigan, "Clustering Algorithms", John Wiley & Sons, 1975.

[DJ87] E. Diday, M. Jambu, C. Hayashi, and N. Ohsumi eds. "Recent Developments in Clustering and Data Analysis." Proceedings of the Japanese-French Scientific Seminar, March 24-26, 1987, Academic Press, inc.

[DH73] Richard O. Duda and Peter E. Hart. "Pattern Classification and Scene Analysis", 1973, John Wiley & Sons.

[Z71] Charles T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters", IEEE Transactions on Computers, Vol. C-20, No. 1, January 1971

[T89] Charles W. Therrien, "Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics," John Wiley & Sons, 1989.

[M67] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in Proc. Fifth Berkeley Symposium on Math. Stat. and Prob., **I**, 281-297, L.M. LeCam and J. Neyman, eds. University of California Press, 1967.

[J89] Leland B. Jackson, "Digital Filters and Signal Processing, Second Edition", Kluwer Academic Publishers, 1989.