Scalability for Clustering Algorithms Revisited

Fredrik Farnstrom
Computer Science
and Engineering
Lund Institute of Technology
Sweden
fredrikf@mail.com

James Lewis
Computer Science
and Engineering
University of California
San Diego
jlewis@cs.ucsd.edu

Charles Elkan
Computer Science
and Engineering
University of California
San Diego
elkan@cs.ucsd.edu

ABSTRACT

This paper presents a simple new algorithm that performs k-means clustering in one scan of a dataset, while using a buffer for points from the dataset of fixed size. Experiments show that the new method is several times faster than standard k-means, and that it produces clusterings of equal or almost equal quality. The new method is a simplification of an algorithm due to Bradley, Fayyad and Reina that uses several data compression techniques in an attempt to improve speed and clustering quality. Unfortunately, the overhead of these techniques makes the original algorithm several times slower than standard k-means on materialized datasets, even though standard k-means scans a dataset multiple times. Also, lesion studies show that the compression techniques do not improve clustering quality. All results hold for 400 megabyte synthetic datasets and for a dataset created from the real-world data used in the 1998 KDD data mining contest. All algorithm implementations and experiments are designed so that results generalize to datasets of many gigabytes and larger.

1. INTRODUCTION

Clustering is the task of grouping together similar items in a dataset. Similar data items can be seen as being generated from the same component of a mixture of probability distributions. The clustering problem is to determine the parameters of the mixture distribution that generated a set of observed data items, where for each item its component is an unobserved feature.

The k-means algorithm is a heuristic solution to the clustering problem based on the assumption that data points are drawn from a fixed number k of spherical Gaussian distributions. The algorithm is an iterative process of assigning cluster memberships and re-estimating cluster parameters. It terminates when the data points no longer change membership due to changes in the re-estimated cluster parameters.

Under the assumption that datasets tend to be small, research on clustering algorithms has traditionally focused on improving the quality of clusterings [4]. However, many datasets now are large and cannot fit into main memory. Scanning a dataset stored on disk or tape repeatedly is time-consuming, but the standard k-means algorithm typically requires many iterations over a dataset to converge to a solution, with each element needing to be accessed on each iteration. Therefore, considerable recent research has focused on designing clustering algorithms that use only one pass over a dataset [9; 6]. These methods all assume that only a portion of the dataset can reside in memory, and require only a single pass through the dataset.

The starting point of this paper is a single pass k-means algorithm proposed by Bradley, Fayyad, and Reina [1]. This method uses several types of compression to limit memory usage. However, the compression techniques make the algorithm complicated. We investigate the tradeoffs involved by comparing several variants of the algorithm of Bradley *et al.* experimentally with a simple new single pass k-means method. Our overall conclusion is that the simple method is superior in speed, and at least equal in the quality of clusterings produced.

2. SINGLE PASS K-MEANS ALGORITHMS

The algorithm of Bradley *et al.* [1] is intended to increase the scalability of k-means clustering for large datasets. The central idea is to use a buffer where points from the dataset are saved in compressed form. First, the means of the clusters are initialized, as with standard k-means. Then, all available space in the buffer is filled with points from the dataset. The current model is updated on the buffer contents in the usual way. The buffer contents are then compressed in two steps.

The first step, called *primary compression*, finds and discards points that are unlikely ever to move to a different cluster. There are two methods to do this. The first method measures the Mahalanobis distance from each point to the cluster mean it is associated with, and discards a point if it is within a certain radius. For the second method, confidence intervals are computed for each cluster mean. Then, for each point, a worst case scenario is created by perturbing the cluster means within the confidence intervals. The cluster mean that is associated with the point is moved away from the point, and the cluster means of all other clusters are moved towards the point. If the point is still closest to the same cluster mean after the perturbations, then it is deemed unlikely ever to change cluster membership.

Points that are unlikely to change membership are removed from the buffer, and are placed in a *discard set*. Each of the main clusters has a discard set, represented by the sufficient statistics for all points belonging to that cluster that have been removed.

On the remaining points in the buffer, another k-means clustering is performed, with a larger number of clusters than for the main clustering. This phase is called *secondary compression*. The aim is to save buffer space by storing some auxiliary clusters instead of individual points. In order to replace points in the buffer by a secondary cluster, the cluster must satisfy a tightness criterion, meaning that its standard deviation in each dimension must be below a certain threshold β. Secondary clusters are combined using hierarchical agglomerative clustering [7], as long as the combined clusters satisfy the tightness criterion.

After primary and secondary compression, the space in the buffer that has become available is filled with new points, and the whole procedure is repeated. The algorithm ends after one scan of the dataset, or if the centers of the main clusters do not change significantly as more points are added.

2.1 Implementation issues

We have coded a new C++ implementation of the algorithm of Bradley *et al.* All the algorithms we compare experimentally are implemented as variants of the same code. The platform for our experiments is a dual 450 MHz Pentium II workstation with 256 megabytes of main memory, running Linux. Our program is not multithreaded, so only one of the processors is directly used in the experiments. The program is compiled with all optimizations turned on. All datasets are stored on disk as Linux binary files. Regardless of the size of any dataset, each pass of each algorithm reads the dataset afresh from disk. Therefore, our experimental conclusions generalize to very large datasets.

Some details of the implementation of their algorithm are not given by Bradley *et al.* For each primary cluster, a Mahalanobis radius must be determined that causes a certain fraction p of buffer points in that cluster to be discarded. Our implementation computes the distance between each buffer point and the cluster it is assigned to. For each cluster, the list of distances is sorted. Then it is easy to find a radius such that a certain fraction of points is discarded. However, sorting can change the time complexity of the whole algorithm. It may be possible to determine each Mahalanobis radius more efficiently, especially when the fraction of discarded points is small.

Our implementation stores the sufficient statistics (sum of elements, squared sum of elements, number of points) as well as the mean and standard deviation in each dimension of all main and secondary clusters. Means are stored so that the distance between old and new means (the new mean is computed from the sum of the elements) can be computed when doing k-means clustering. Standard deviations are stored to speed up primary compression. Representing one cluster uses four times as much space as one data point. Therefore, if a secondary cluster contains four or fewer points, the points themselves are retained instead of a representation of the cluster.

For our purposes, the sufficient statistics of a cluster are two vectors, *Sum* and *SumSq*, and one integer, n. The vectors store the sum and the sum of squares of the elements of the points in the cluster, and the integer records the number of points in the cluster. From these statistics, the mean and variance along each dimension can be calculated. Let the sufficient statistics of a cluster A be $(Sum^{(A)}, SumSq^{(A)}, n^{(A)})$. If a point x is added to the cluster, the sufficient statistics are updated as follows:

$$\begin{aligned} Sum_j^{(A)} &:= Sum_j^{(A)} + x_j \\ SumSq_j^{(A)} &:= SumSq_j^{(A)} + x_j^2 \\ n^{(A)} &:= n^{(A)} + 1. \end{aligned}$$

If clusters A and B are merged, the sufficient statistics for the resulting cluster C are

$$\begin{aligned} Sum_j^{(C)} &= Sum_j^{(A)} + Sum_j^{(B)} \\ SumSq_j^{(C)} &= SumSq_j^{(A)} + SumSq_j^{(B)} \\ n^{(C)} &= n^{(A)} + n^{(B)}. \end{aligned}$$

2.2 A simple single pass k-means method

A special case of the algorithm of Bradley *et al.*, not mentioned in their paper, would be when all points in the buffer are discarded each time. This algorithm is:

1. Randomly initialize cluster means. Let each cluster have a discard set in the buffer that keeps track of the sufficient statistics for all points from previous iterations.
2. Fill the buffer with points.
3. Perform iterations of k-means on the points and discard sets in the buffer, until convergence. For this clustering, each discard set is treated like a regular point placed at the mean of the discard set, but weighted with the number of points in the discard set.
4. For each cluster, update the sufficient statistics of the discard set with the points assigned to the cluster. Remove all points from the buffer.
5. If the dataset is exhausted, then finish. Otherwise, repeat from Step 2.

This algorithm is called the *simple single pass k-means* method. Compared to the more complicated algorithm above, it does much less computation each time the buffer is filled, and the whole buffer can be filled with new points at every fill. Following Bradley *et al.* [1], if a cluster ever becomes empty, it is reinitialized with the point in the buffer that is most distant from the centers of all other clusters. However, with a large dataset and a small number of clusters, reinitialization is almost never necessary.

Like the more complicated algorithm above, the simple method uses only one scan over the dataset and a fixed size buffer. It also satisfies all the other desiderata listed by Bradley *et al.* [1]: incremental production of better results given additional data, ease of stopping and resuming execution, and ability to use many different database scan modes, including forward-only scanning over a database view that is never materialized completely.

3. LESION EXPERIMENTS

To evaluate the contribution of each of the data compression methods, we report the results of experimental lesion studies. Comparisons are made between four variants of the algorithm of Bradley *at al.*, the standard k-means algorithm, and the simple single pass k-means algorithm described above.

Parameter	Value
Confidence level for cluster means	95%
Max std. dev. for tight clusters (β)	1.5
Number of secondary clusters	20
Fraction of points discarded (p)	20%

Table 1: Parameter settings used for the lesion studies of the k-means algorithm of Bradley *et al.*

Three variants involve adding one of the data compression methods described above to the previous variant. The first variant uses none of the data compression techniques. This variant runs until convergence on the first fill of the buffer, and then stops. This variant is similar to clustering on a small random sample of the dataset. In the second variant, the first primary compression technique is used. This involves moving to the discard set each point within a certain Mahalanobis distance from its associated cluster mean. In the third variant, the second primary compression technique is added. Confidence intervals are used to discard data points deemed unlikely ever to change cluster membership. The fourth variant includes the data compression technique of determining secondary clusters. All parameter settings used in the experiments reported here are shown in Table 1.

3.1 Synthetic datasets

The lesion experiments use synthetic datasets. Using artificial data allows the clusters found by each algorithm to be compared with known true probability distribution components. In each synthetic dataset, points are drawn from a mixture of a fixed number of Gaussian distributions. Each Gaussian is assigned a random weight that determines the probability of generating a data point from that component. Following Bradley *et al.* [1], the mean and variance of each Gaussian are uniformly sampled, for each dimension, from the intervals $[-5, 5]$ and $[0.7, 1.5]$ respectively.

In order to measure the accuracy of a clustering, the true cluster means must be compared with the estimated cluster means. The problem of discovering which true cluster mean corresponds to which estimated mean must be solved. If the number of clusters k is small, then it is possible to use the one of the $k!$ permutations that yields the highest accuracy. We do this, and for this reason the number of clusters $k = 5$ is small in our experiments. Results with a much larger number of clusters might be different.

The synthetic datasets have 100 dimensions and 1,000,000 data points. They are stored on disk in 400 megabyte files. This size is chosen to guarantee that the operating system cannot buffer a dataset in main memory. Except for the standard k-means algorithm, each clustering algorithm uses a limited buffer large enough to contain approximately 1% of the data points.

The experiments use 30 different synthetic datasets. For each dataset each algorithm generates five different clusterings from different initial conditions. The best of these five models is retained for a comparison of the accuracy of the algorithms. The best of five runs is used because k-means algorithms are known to be sensitive to how cluster centers are initialized. In applications where a good clustering is wanted, it is therefore natural to use the best of several runs.

In general one of two different situations occurs with each clustering. Either, one cluster mean in the model is close to each true Gaussian center, or, two cluster means in the model are trapped near the same center. As we measure the distance between the true and the estimated cluster means, if a center is trapped then the distance measure will be much larger than otherwise. Therefore, the cluster quality in Figure 1 is based only on datasets for which *every* algorithm produced *at least one* clustering where no center is trapped.

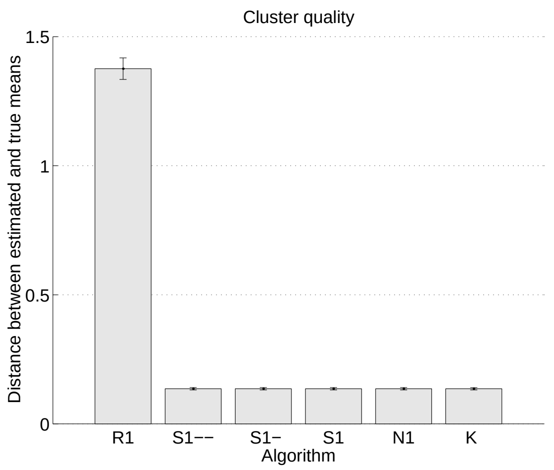

Figure 1: The graph shows the mean sum of the distances between the estimated and true cluster means, for synthetic datasets of 1,000,000 points, 100 dimensions, and five clusters. The algorithms are random sampling k-means (R1), single pass k-means with the first primary compression technique only (S1−−), with both primary compression techniques (S1−), with primary and secondary compression (S1), the simple single pass k-means method (N1), and the standard k-means algorithm operating on the whole dataset (K). Error bars show standard errors.

3.2 Lesion experiment results

Figure 1 shows that even the simplest single pass algorithm achieves the same clustering quality as the full k-means method. Random sampling k-means is less accurate because it uses only 1% of the total data points.

A clustering where no centers are trapped is highly desirable. Therefore, we also measure the fraction of clusterings where no centers are trapped, counting clusterings from all five random initial conditions. We call this fraction the *reliability* of an algorithm. Surprisingly, Figure 2 shows that the single pass algorithms are more reliable than the standard k-means algorithm, and this difference is statistically significant.

Throughout this paper, the difference between x and y is called statistically significant if $x + s_x < y - s_y$ or $y + s_y < x - s_x$, where s_x and s_y are the standard errors of x and y respectively. If x is the mean of n observations then its standard error is the standard deviation of the n observations divided by $\sqrt{n}$. For the special case where x is a proportion, its standard error is $\sqrt{x(1-x)/n}$. If n is sufficiently large then a Gaussian approximation is valid, so the null hypothesis that the true values of x and y are the same can be rejected with confidence $p < 0.05$, if $x + s_x < y - s_y$ or $y + s_y < x - s_x$. Numerical p values from specific statisti-

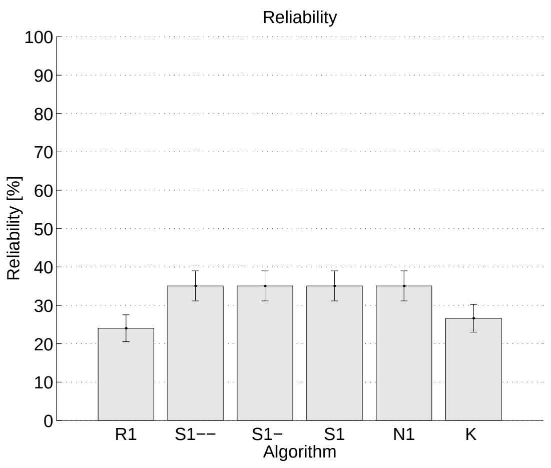

Figure 2: The graph shows the reliability of the different algorithms on the synthetic datasets. Reliability is defined as the fraction of all runs where no centers are trapped. Error bars show standard errors.

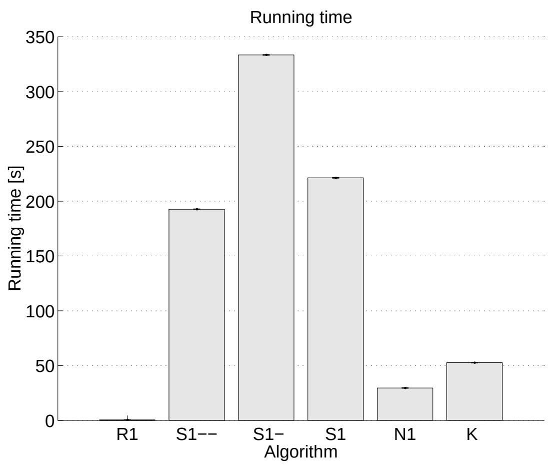

Figure 3: The graph shows the average running time of each k-means algorithm variant. Error bars show standard errors.

cal tests are not reported because their precision could be misleading, since the assumptions on which standard tests are based are often not valid when comparing performance metrics for data mining methods [3].

Figure 2 shows surprisingly that the standard k-means algorithm is not significantly more reliable than random sampling k-means. This fact indicates that the standard algorithm has difficulty escaping from a bad initialization, regardless of how many data points are available. Similarly, the more complicated single pass methods are not more reliable than the simple single pass method. This fact indicates that the more complicated methods do not have any improved ability to escape from a bad initialization.

The average running time of each algorithm is shown in Figure 3. Reported times are averages over 135 runs for each algorithm. The full algorithm of Bradley *et al.*, identified as S1 in Figure 3, is about four times slower than the standard k-means algorithm, while the simple single pass method is about 40% faster.

With the method of Bradley *et al.*, each additional data compression technique allows more points to be discarded from the buffer. Doing so should make the algorithm run faster, because then fewer refills of the buffer are needed. A balance must be maintained between the time taken to identify points to discard and the speedup gained from discarding those points. Figure 3 shows that compression based on confidence interval perturbation causes a net decrease in speed, while compression based on secondary clustering is beneficial.

4. EXPERIMENTS WITH REAL DATA

In order to experiment with real-world data, the dataset from the 1998 KDD (Knowledge Discovery and Data Mining Conference) contest is used. This dataset contains information about people who have made charitable donations in response to direct mailing requests. In principle, clustering can be used to identify groups of donors who can be targeted with specialized solicitations in order to maximize donation profits.

The dataset contains 95412 records, each of which has 481 fields. We take a subset of these fields and code each record as a real-valued vector. Numerical fields (e.g. amounts of past donations, income, age) are directly represented by a single element in the vector. Date values (e.g. donation dates, date of birth) are stored as the number of months from a fixed date. Fields with discrete values, such as an income category, are converted into several binary elements. Each vector has 56 elements in total, of which 18 are binary. To give equal weight to each feature, each feature is normalized to have zero mean and unit variance. The records in the original KDD dataset are converted to this format and saved to a binary file of about 21.4 megabytes. As mentioned in Section 2.1, the implementation of the standard k-means algorithm reads the dataset from disk at each iteration, even though the dataset is small enough to be saved in memory.

The purpose of this experiment is to compare the running time and clustering quality of standard k-means, operating on the whole dataset or on samples, the algorithm of Bradley *et al.* using all types of compression, and the simple single pass method. Experiments are performed with samples and buffers of 10% and 1% of the size of the whole dataset. The number of clusters is always 10.

First, the dataset is randomly reordered. Then it is clustered five times by each algorithm, each time with different randomly chosen initial conditions. All algorithms use the same five initial conditions. The quality of each clustering is measured as the sum of the squared distances between each point and the cluster mean it is associated with. Of the five clusterings for each algorithm, the one with the best quality is used. As above, the best of five is chosen because k-means algorithms are highly sensitive to initial conditions. The whole procedure is repeated 52 times with different random orderings of the dataset.

It is difficult to discover good parameter values for the algorithm of Bradley *et al.*, especially for the parameters that control the number of points removed by secondary compression. The values used here are given in Table 2. Note that it is difficult for a secondary cluster to have standard

Parameter	Value
Confidence level for cluster means	95%
Max std. dev. for tight clusters (β)	1.1
Number of secondary clusters	40
Fraction of points discarded (p)	20%

Table 2: Parameter settings used for the algorithm of Bradley *et al.* with the KDD dataset.

deviation $\beta \leq 1.1$ in *every* dimension, even though the whole dataset is normalized to have standard deviation 1.0 in each dimension.

Figure 4 shows the average quality of the best of five clusterings, for each algorithm. Random sampling k-means operating on a 1% sample performs much worse than all other methods. Standard k-means performs best, followed by the simple single pass method using a buffer of size 1%, followed by the algorithm of Bradley *et al.* All differences mentioned here are statistically significant.

There is no "true" clustering of the KDD dataset that can be used to define reliability in a way similar to how reliability is defined for the synthetic datasets. Therefore, the reliability of an algorithm is defined here to be the fraction of all clusterings that have a quality measure of less than $3.9 \cdot 10^6$. This number is chosen somewhat arbitrarily based on Figure 4 as a threshold for what constitutes an acceptable clustering. A reliable algorithm is one that is less sensitive to how cluster centers are initialized, and that produces a good clustering more often.

Figure 5 shows that the standard k-means method and the simple single pass method with a buffer of size 1% are the most reliable. All other methods are statistically significantly less reliable. It is surprising that the simple single pass algorithm using a buffer of size 1% of the entire dataset outperforms the same method using a 10% buffer. Similar results were found by Bradley *et al.* [1] when they varied the buffer size used by their algorithm. The reason why a smaller buffer can be better remains to be discovered.

Figure 6 shows the average running time of each method. Compared to the standard k-means method, the algorithm of Bradley *et al.* is over four times slower, while the simple single pass method is over five times faster.

5. COMPUTATIONAL COMPLEXITY

In the discussion here of the asymptotic efficiency of the algorithms, we use the following notation:

m	number of k-means passes over entire dataset
m'	number of k-means passes over one buffer refill
d	number of dimensions
n	number of data points
b	size of buffer, as fraction of n
r	number of buffer refills
k	number of main clusters
k_2	number of secondary clusters
m_2	number of passes for each secondary clustering.

The time complexity of the standard k-means algorithm is $O(nkdm)$, where empirically m grows very slowly with n, k, and d.

For the simple single pass k-means algorithm, the time complexity of clustering the buffer contents once is $O(nbkdm')$.

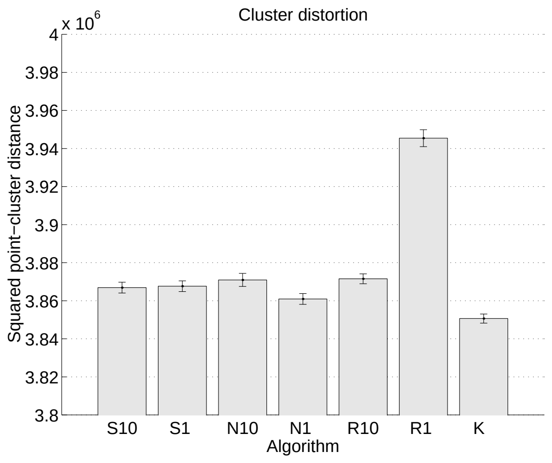

Figure 4: The graph shows the sum of the squared distances between each point in the dataset and the cluster mean it is associated with, on the KDD contest dataset of 95412 points with 10 clusters. The algorithms are due to Bradley *et al.* (S10 and S1), the simple single pass method (N10 and N1), random sampling k-means (R10 and R1), and standard k-means working on the whole dataset. Algorithms with names ending in 10 use a buffer or sample of size 10% of the whole dataset, while those with names ending with 1 use a 1% buffer or sample. Error bars show standard errors.

Algorithm	Time	Space	I/O
Standard	$nkdm$	kd	ndm
Bradley *et al.*	$nbrk_2dm_2$	$nbd + k_2^2d$	nd
Simple single pass	$nkdm'$	nbd	nd

Table 3: Order of magnitude time, memory, and disk input/output complexity for different k-means algorithms.

Because the buffer is emptied completely before each refill, the number of refills is $1/b$, so the time complexity of clustering the whole dataset is $O(nbkdm' \cdot 1/b) = O(nkdm')$.

Interestingly, m' tends to be less than m because clustering is performed over fewer data points than for standard k-means. In fact, m' tends towards one for large datasets, because when the model has stabilized, new points are simply placed in the nearest cluster. This observation is true for all the single pass algorithms.

The complicated nature of the method of Bradley *et al.* makes it difficult to analyze. The main clustering takes $O(nbkdm')$ time per fill. Measuring the Mahalanobis distance to the closest cluster for the points in the buffer is an $O(nbd)$ operation. Finding the discard radius for all main clusters takes $O(nb \log nb)$ time if sorting is used; the worst case is when essentially all points belong to one cluster. The total time complexity of the first method of primary compression is thus $O(nb(d + \log nb))$. The second method of primary compression, where the cluster means are perturbed, has time complexity $O(nbkd)$.

In the secondary compression phase, m_2 passes with k_2 clusters are performed over the points in each fill of the buffer, giving this phase $O(nbk_2dm_2)$ complexity for one fill of the buffer. Then, hierarchical agglomerative clustering is per-

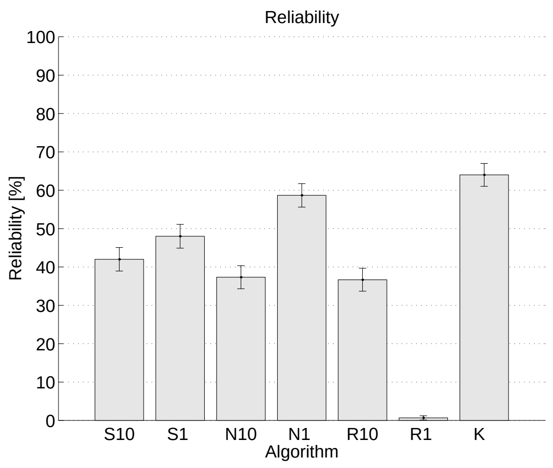

Figure 5: The graph shows the reliability of each different algorithm on the KDD contest dataset, defined as the fraction of clusterings having a distortion less than $3.9 \cdot 10^6$. Error bars show standard errors.

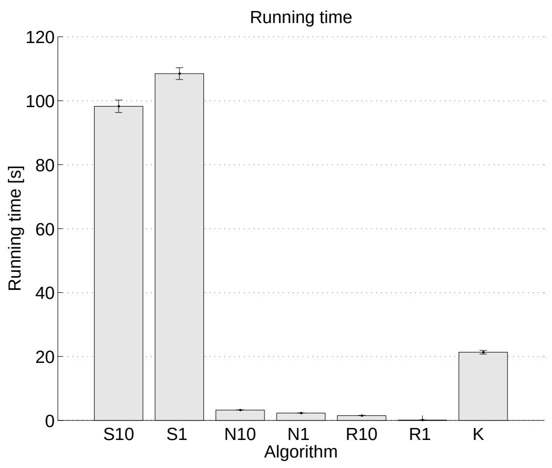

Figure 6: The graph shows the average time taken by each method to perform one clustering of the KDD dataset. Error bars show standard errors.

formed on the k_2 clusters. This can be done with $O(k_2^2 d)$ time and space complexity [7].

The steps described above must be repeated r times to scan through the whole dataset. Typically $r > 1/b$ since the whole buffer cannot be filled at each fill. So, the algorithm of Bradley *et al.* has a total time complexity of

$$O\left(nbr(kdm' + d + \log nb + k_2 dm_2 + \frac{k_2^2 d}{nb})\right).$$

In general $k_2 > k$ and $m_2 > m'$, so the total time complexity is $O(nbrk_2dm_2)$. An assumption here is that the clustering is not stopped until the whole dataset has been processed. This assumption is true in all our experiments.

The time, memory, and disk I/O complexities of the three algorithms are summarized in Table 3. The simple single pass algorithm is superior asymptotically in both time and space complexity to the algorithm of Bradley *et al.*

6. DISCUSSION

The main positive result of this paper is that a simple single pass k-means algorithm, with a buffer of size 1% of the input dataset, can produce clusterings of almost the same quality as the standard multiple pass k-means method, while being several times faster.

Being faster than the standard k-means algorithm is not a trivial accomplishment, because the standard algorithm is already quite scalable. Its running time is close to linear in the size of the input dataset, since the number of passes required is empirically almost independent of the size of the dataset. In addition, at each pass the dataset is scanned sequentially, so a good operating system and disk array can easily provide access to the dataset with high bandwidth.

Although it is called scalable, the algorithm of Bradley *et al.* is much slower in our experiments than the standard k-means method. Bradley *et al.* did not report this fact because their paper contains no comparisons with standard k-means, and no running times. Moreover, the paper gives no measures of statistical significance for differences in clustering quality between algorithms, and the largest dataset used in the paper whose size can be computed from information in the paper occupies only 10 megabytes when stored as a floating point binary file. The operating system of any modern workstation can cache a dataset of this size in main memory.

We may not have found optimal settings for the parameters of the algorithm of Bradley *et al.* However, we have searched informally for good parameter settings. In general, algorithms that have many parameters with few guidelines about how to choose values for them are difficult to use effectively.

Compared to the standard k-means algorithm, the method of Bradley *et al.* is slower on the KDD dataset than on the synthetic datasets. The opposite is true for the simple single pass method: it is relatively faster on the KDD dataset. The reason is that the KDD dataset has clusters that are separated less well, and the method of Bradley *et al.* is sensitive to clusters not being separated well. The standard algorithm requires 13 passes on average to converge on the KDD dataset, but only 3.3 passes on the synthetic datasets. As explained in Section 5, the number m' of iterations per refill of the buffer tends to 1 for the simple algorithm for all datasets. But for the method of Bradley *et al.*, the number m_2 of iterations in the secondary clustering for each refill of the buffer may remain high.

With all k-means algorithms, both single pass and multiple pass, it is possible to update several clusterings in parallel, where each clustering starts from different initial conditions. We did not do so for our experiments. If we did so, the average running time per clustering of all methods would presumably decrease. There is no reason to think that the relative speeds of the methods would change.

If a dataset to be clustered does not already exist as a single table in a relational database or as a flat file, then materializing it can be expensive. Materializing a dataset may be especially expensive if it consists of a join of tables in a distributed or heterogeneous data warehouse. In this case, all single pass clustering methods can be faster than the standard k-means algorithm. However, the ranking of different single pass methods according to speed is likely to be still the same.

The results of this paper are complementary to those of Pelleg and Moore [8], who show how to use a sophisticated data structure to increase the speed of k-means clustering for datasets of low dimensionality ($d < 8$). Our simple single pass method is effective regardless of dimensionality. The results here are also complementary to those of Guha, Mishra, Motwani, and O'Callaghan [5], who present single pass clustering algorithms that are guaranteed to achieve clusterings with quality within a constant factor of optimal.

We have not tested other single pass clustering algorithms, notably the BIRCH method [9]. The authors of BIRCH have shown convincingly that it is faster than k-means on large datasets. A comparison of the simple single pass method of this paper with BIRCH would be interesting. Also, all the single pass methods discussed in this paper can be extended to apply to other iterative clustering approaches, and in particular to expectation maximization (EM) [2]. It would be interesting to repeat the experiments of this paper in the EM context.

Acknowledgments: The authors are grateful to Nina Mishra and Bin Zhang of Hewlett Packard Laboratories and to the anonymous referees for valuable comments.

7. REFERENCES

[1] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 9–15. AAAI Press, 1998.

[2] P. Bradley, U. Fayyad, and C. Reina. Scaling EM (expectation maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research, Redmond, WA, November 1998.

[3] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

[4] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. *Computer*, 32(8):38–45, 1999.

[5] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the Annual Symposium on Foundations of Computer Science*. IEEE, November 2000. To appear.

[6] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84. ACM, 1998.

[7] M. Meila and D. Heckerman. An experimental comparison of several clustering and initialization methods. Technical Report MSR-TR-98-06, Microsoft Research, Redmond, WA, February 1998.

[8] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1999.

[9] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114. ACM, 1996.