Introduction

The k-means clustering algorithm and its variants are widely used in multivariate data analysis [5], pattern classification [11] and digital image processing [4]. Given a certain metric, it partitions multivariate data by finding a local minimum in the mean squared error. The algorithm is generally considered to be one of the faster methods of clustering. However, its computational complexity is still far too high for large clustering problems.

This paper proposes a new, fast k-means type clustering method, called the *adaptive mean-split* algorithm. Using a divide-and-conquer strategy, it produces K clusters from N multivariate data points in only $O(N\log K)$ time and $O(N)$ space. Experiments on color image quantization, a large clustering problem which presents real challenges to clustering algorithms (see [6]), show that the new method can partition data in the order of minutes when the k-means algorithm requires a whole day to converge. The clustering error shows little deterioration in the resulting clusters. Moreover, if further optimization is sought, the outcome of the new method can be used to create initial cluster centers for the k-means algorithm, gaining a very substantial head-start on the usual technique of beginning from randomly-chosen or roughly-selected initial values.

The structure of the paper is as follows. The first section defines the problem and briefly reviews the k-means algorithm. Then the adaptive mean-split algorithm is introduced and developed in a way which emphasizes the compromise it achieves between computational complexity and optimization. Finally the performance of the new algorithm is analyzed by both a probability model and experiments on color image data, where it is also compared with other algorithms.

Data partitioning and the k-means algorithm

Given N m-variate data q_j $(j=0,1,\cdots N-1)$, there is a corresponding point set $S=\{q_0,q_1,\cdots q_{N-1}\}\subset R^m$, the m-dimensional Euclidean space. Let $P(S,K)$ denote a partition obtained by dividing S into K subsets S_i, $S_i\neq\Phi$, $\bigcap_{i=0}^{K-1}S_i=\Phi$, $\bigcup_{i=0}^{K-1}S_i=S$ $(2\leq K<N)$. $P(S,K)$ can

be viewed as a map $P: S \rightarrow I^+$ (the positive integers), where $P(q_j)=i$ whenever $q_j \in S_i$. P takes each point of S into the subset to which it belongs.

A good partition will localize the data points by suitable choice of subsets. Throughout this paper Euclidean distance is used to measure locality. Each subset is represented by a suitable point in space; denote the representative of S_i by $\overline{S(i)}$. Then the total squared error for the partition $P(S,K)$ is

$$e[P(S,K)] = \sum_{i=0}^{N-1} \| q_i - \overline{S(P(q_i))} \|^2 \qquad Eq\text{-}1$$

where $\| \ \|$ represents Euclidean distance. It is obvious that, whatever the partition, the best choice of point to represent each subset is the centroid of all data points which belong to it. This choice will minimize the squared error for that subset, and consequently for the partition as a whole.

The aim of multivariate data partitioning is to discover or approximate that partition which minimizes $e[P(S,K)]$. The optimal partition is denoted by $P_{opt}(S,K)$. Since there are approaching K^N different partitions (in fact the number can be expressed exactly in closed form as a Stirling number of the second kind [1]), it is clear that exhaustive searching for P_{opt} is generally out of the question.

The k-means algorithm [8] is a commonly-used method for finding a partitioning $P(S,K)$ that has a low error. Starting from initial guesses for the K desired cluster centers (perhaps chosen as the first K data points to be clustered), it assigns each data point to the cluster center closest to it. This produces K initial clusters. Now a new center is assigned to each cluster by computing the centroid of all points in it. If all centers remain at the same place, the algorithm terminates; otherwise the data points are reassigned to cluster centers and the procedure is repeated.

While the k-means method is widely used in practice, a general proof of convergence appeared only recently [9]. Moreover, a moment's reflection reveals it does not necessarily converge to the best possible partition, but merely to a locally optimal configuration. This does not appear to have been a significant problem in practice; indeed according to [5, p.103] there are some asymptotic results which suggest that for large sample sizes there will be only a few local optima, differing little from each other.

Unfortunately the k-means algorithm is far too time-consuming for large clustering problems. Its computational complexity is $O(TmNK)$ where T represents the number of iterations required for the algorithm to converge. T will depend on the distribution of the data, and generally increases with the size of the clustering problem (measured by m, N, and K). For large problems where millions of multidimensional data points must be structured into dozens of clusters, the k-means algorithm becomes totally infeasible. For example, although in principle quite suitable for color image quantization problems (where $m=3$, $N\approx 2^{18}$, $K\approx 256$), its prohibitively high cost forces the use of inferior, heuristic, approaches [6].

The adaptive mean-split algorithm

In this paper we propose a new algorithm for clustering problems. This aims for fast execution, while sacrificing only a small increase in the total error $e[P(S,K)]$ of the partitions it discovers. It adopts the basic divide-and-conquer approach which has proven successful for the k-d tree and other multivariate data partitioning problems [2]. The division is performed on both S and K; these are called "spatial" and "quota" divisions respectively. Spatial division is accomplished quickly and approximately. Any inaccuracy in this is offset by a compensating quota division, by apportioning the number of clusters between the two data subsets according to their population and spread. This section describes and motivates the components of the algorithm, and then gives its implementation.

Divide-and-conquer methodology. The algorithm is basically recursive. Instead of trying to find K clusters in a large data set S at once, it divides the $P(S,K)$ problem into two smaller ones, $P(S_1,K_1)$ and $P(S_2,K_2)$. This is done by first splitting the data set S into S_1 and S_2, and then splitting K into K_1 and K_2 according to the populations and spreads of the two parts. This procedure is then repeated for the partitions $P(S_1,K_1)$ and $P(S_2,K_2)$. K_1 and K_2 act as quotas of partitions allocated to S_1 and S_2. A subdivision terminates whenever its quota is 1, or whenever it decreases beyond a certain minimal size, whichever happens first. In the latter case the excess quota is redistributed between the remaining partitions. The algorithm terminates with K partitions $P(S_i,1)$, $i=0,1,...,K-1$, whose combination is a

solution of $P(S,K)$.

At each stage the subdivision could be accomplished by the conventional k-means (in this case, 2-means) clustering algorithm. This would involve selecting two initial cluster centers; constructing the discriminating hyperplane; segregating the data points x of S into two clusters using a test like

$$\sum_{i=1}^{m} a_i x_i + a_0 \gtrless 0; \qquad Eq\text{-}2$$

calculating the centroid of each cluster; and iterating until insignificant change took place in the cluster centers. The discriminating hyperplane, which constitutes the Voronoi diagram of the two tentative cluster centers, is defined by the coefficients a_i. The total cost of this divide-and-conquer 2-means partitioning is $O(tmN \log K)$, where t is the average number of iterations to convergence over all stages.

This strategy achieves a great reduction in computational complexity over the conventional k-means algorithm. Firstly, computation time depends logarithmically rather than linearly on K. Secondly, the smaller values of N and K which are involved in each subdivision imply that each 2-means iteration converges much more quickly on average than an overall k-means one would (ie $t \ll T$). Thirdly, the geometric calculation for 2-means division is much simpler than in the general case because only a single hyperplane decision is required instead of a full Euclidean distance calculation. Against this, the overall error $e[P(S,K)]$ associated with the new algorithm would be greater than for k-means since only two neighbors are considered at each stage, and it is not clear how much this would increase $e[P(S,K)]$.

In order to perform as well as k-means, the new algorithm would have to use a complex hypersurface instead of a hyperplane to perform the binary discrimination at each stage. This is because the clustering produced by k-means can be viewed as an m-dimensional Voronoi diagram on the cluster centers – in geometric terms, as a combination of K convex polyhedra. It is clearly impractical to extend the divide-and-conquer methodology in this way.

Instead, it is extended in a different direction, by compensating for inaccurate spatial division by appropriate allocation of the quota to the two parts. This self-adapting strategy is sufficiently powerful to

allow further simplification of the subdivision from the general hyperplane discriminant described above, with concomitant increase in execution speed. The next subsection describes the spatial subdivision policy; the following one covers the method of quota allocation.

Spatial division. In common with the k-d tree method of data partioning [3], the problem is simplified by always aligning the separating hyperplane parallel to the coordinate axis along which the data points extend furthest. This eliminates the dependency of execution time on the dimension m of the data space by reducing the test of *Eq*-2 above to $x_d \gtrless \xi$, where d is the appropriate coordinate axis.

Now consider the key question: how to determine the cutting point ξ for a subset $\Omega \subset S$ of data points. Let the optimal cutting position (namely the one which minimizes $e[P(\Omega,2)]$) be ξ_{opt}. This cutting position could be obtained by iterative 2-means clustering. However, this is deemed to be too expensive, and instead it is estimated as follows. Rather than minimizing the overall error, we seek to minimize the error of the d th marginal distribution. This crude approximation is acceptable because that marginal distribution is likely to have higher variance than the others, and different cutting axes d will be used at different levels of subdivision. As shown in Appendix A, the mean μ for a continuous univariate distribution lies between the optimal cutting point ξ_{opt} and the median ξ_m. This suggests that ξ_{opt} could be estimated by a linear combination of the mean and median of the points in Ω:

$$\xi_{opt} \approx \mu_d + \lambda(\mu_d - \xi_{m,d}) \qquad \textit{Eq}\text{-3a}$$

(subscript d stands for that coordinate axis along which the cut takes place). λ is a positive constant which could be optimized for any specific marginal distribution function. Unfortunately, the computational effort required to find the median would increase the execution time of the adaptive mean-split algorithm very significantly. A similar and much cheaper estimator of ξ_{opt} is

$$\xi_{opt} \approx \mu_d + \eta(c_d - \mu_d), \qquad \textit{Eq}\text{-3b}$$

where c is the geometric center of the hyperbox tightly bounding Ω. η is an empirically-determined constant in the range [0.05,0.25] (the reason for this range is explained later). *Eq*-3b can be understood

intuitively as saying that the best cutting point is close to the mean; very close in the case when the mean is near the geometric center of the tightest bounding hyperbox. However, when the mean strays significantly from the geometric center, the cutting point moves slightly towards the center. This is because the outlying point or points at or near the edge of the hyperbox which is opposite the mean will have a dominating effect on the mean-square error criterion.

The accuracy of the approximation of *Eq*-3b has been verified for a number of different data distributions. It will be examined further below in a discussion of the performance of the adaptive mean-split algorithm, and *Eq*-3a will be reconsidered there also.

Quota division. Whenever a data set S is split into parts S_1 and S_2, the quota K allocated to S is split into K_1 and K_2. The quota subdivision strategy allows the algorithm to adapt to the result of the current cutting operation and feed this information back into further divisions. This produces remarkably good performance without the need for expensive geometric calculations.

The populations $N(S_1)$, $N(S_2)$ and the volumes $V(S_1)$, $V(S_2)$ of the tightest bounding hyperboxes of sets S_1 and S_2 comprise the information on which the quota subdivision is based. It would be preferable to use variances instead of volumes, but this information is expensive to calculate. At first sight it may seem sufficient to distribute K according to the populations alone; but on reflection it is clear that the variances (approximated by the hyperbox volumes) will have an effect on the total squared error of *Eq*-1. Both effects can be incorporated by choosing

$$K_i = \left[\rho \frac{N(s_i)}{N(s_1) + N(s_2)} + (1 - \rho) \frac{V(s_i)}{V(s_1) + V(s_2)} \right] . K \qquad (i = 1, 2). \qquad \textit{Eq-4}$$

$\rho \in [0,1]$ is an empirically-determined constant, usually chosen between 0.5 and 0.7.

During earlier levels of division, the quota is relatively large and the quota-distribution procedure can effectively compensate for poor choice of cutting point. Little is gained by a more extensive computation of a better approximation for ξ_{opt}. For example, cutting data sets at the geometric center of their bounding hyperboxes is usually quite adequate for these early divisions. Later, when the quota is

small, it may be worth expending more effort on choosing better hyperplanes for subdivision; this does not impact computation time significantly because the number of data points involved is small. In particular, when K is down to two it may be worthwhile to use iterative 2-means spatial division. At this level it will have a small cost and will guarantee a reduction in $e\,[P(S,K)]$.

The divide-and-conquer methodology distributes the effort appropriately between spatial division and quota division at different levels of the tree. This ensures good performance with a small computational investment.

Implementation of the algorithm. The main parts of the adaptive mean-split algorithm are shown in the Pascal pseudo-code given in Figure 1. After briefly explaining this implementation, it will be shown how it can be made more efficient using a more sophisticated data structure.

The procedure *partition(k, n, tag, dataset)* divides the data set into k partitions. It works by splitting the data set in two and calling itself recursively for each part. The centroids of the clusters are stored in a global array and calculated as soon as each new cluster is created. *tag* is a number identifying the current cluster (starting from 0); the next unused cluster number is kept in the global variable *newtag*.

The body of procedure *partition* first checks the quota k allocated to this partition, and returns if it is one. Next it determines if the current partition is less than a pre-specified minimum size. If so, it does not attempt to further subdivide, but returns the unused quota to a pool for use by other, more needy, partitions (this part of the code is not shown). Otherwise, it finds which dimension of the data set has the greatest extent, using procedure *widest_dimension(hyperbox)*. (The details of how the hyperbox dimensions are calculated and stored are not shown.)

The result of this operation is used as a parameter to the procedure *divide(cutaxis)*. This estimates the optimal position to cut along the chosen axis, using *Eq*-3b. Then it performs the division operation, segregating the resulting points into arrays *subset1* and *subset2*. Finally it calculates the centroids and places them in the global array. Note that given the centroid before the partition was divided, and the centroid of just one part afterwards, the centroid of the second part can be determined directly. This is a

useful shortcut. *divide* also places the populations of the two subsets into local variables *n1* and *n2* for later use by other procedures.

Returning to the main body of *partition*, function *quota1* is called next to determine what part of k should be allocated to the first subset, according to *Eq*-4. Finally, *partition* calls itself twice recursively, once for each subset.

Some technical improvements can easily be made to this basic algorithm. The copying operation into *subset1* and *subset2* is unnecessary, and these arrays consume a large amount of space unless more sophisticated dynamic memory management is used. Instead, the data points can be shuffled around within a single array. The basic idea is to repeatedly swap elements of the array until each cluster's points are adjacent. Instead of manipulating the actual data points, however, it is better to define a global vector *labels[1..N]*, containing initially the numbers 1..N and subsequently permutations of that set, to track which partition each data point belongs to. These numbers index the actual data points, and during the *divide* procedure only the labels need be swapped.

Figure 2a shows the *label* array at some intermediate point during clustering. Several clusters have by now been formed, some of which will be split (and, perhaps, split again) before the procedure is finished. Each cluster occupies a contiguous sequence of locations in the *label* array. Auxiliary arrays *ptr[0..K-1]* and *no[0..K-1]* tell more about the clusters. *ptr[17]*, for example, points to the beginning of the contiguous sequence representing cluster 17, and *no[17]* gives the size of that cluster. Suppose that the next step involves splitting cluster 17 into a new cluster 17 and a further cluster, to be numbered 33 (this supposes that 32 clusters are represented all told in Figure 2a). The appropriate *cutaxis* and *cutplane* will be determined as before. But instead of copying points into *subset1* and *subset2*, the procedure will shuffle that part of the *label* array containing cluster 17 by exchanging elements as appropriate, to build the data structure of Figure 2b. *ptr[17]* remains the same but *no[17]* is updated, and *ptr[33]* and *no[33]* are created as shown.

The improved *divide* procedure is shown in Figure 3. The new data structures are *label[1..N]*, *ptr[1..K]*, and *no[1..K]*, and they should be initialized with *label[i]=i*, *ptr[0]=1*, and *no[0]=N*.

Performance of the adaptive mean-split algorithm

This section discusses the performance of the new algorithm and compares it with others. A number of experiments on large data sets demonstrate that it occupies a new and important place in the computation/optimality tradeoff: it is exceedingly speedy in comparison with all other methods we know of for large problems; and its results are close to the local optima discovered by the k-means algorithm. In addition to these experimental findings, a study of the linear estimator of ξ_{opt} (Eq-3b) on a particular family of probability distributions provides some further encouraging evidence.

Experimental results. The adaptive mean-split algorithm has been tested extensively on a set of large clustering problems. These problems involve color image quantization, in which a good approximation to an original image is sought within the limitation of K available colors. The chief advantage of this domain is that the results can be inspected visually.

Pictures with a spatial resolution of 256×256 pixels are used, each having a color resolution of 8 bits in each of the three primaries, red, green, and blue. Given a picture, the problem involves selecting a palette of K colors from the 2^{24} possible ones, and mapping each pixel on to its nearest representative in the palette. Clearly what is required is a clustering $P(S,K)$, where S is the set of pixels in the picture and contains 2^{16} 3-dimensional data points. Typical values for K are 16, 64, and 256. This corresponds to a data reduction from 24 bit/pixel to between 4 and 8 bit/pixel.

Recognized clustering algorithms such as k-means are impractical for such a large problem. A survey paper [6] discusses the use of various simple and heuristic methods. Later, a new approximate method was proposed based on recursive space-filling curves [10]. Among all these techniques, median cutting in the style of the k-d tree method [3] performs best. It and the k-means algorithm will be compared with the new adaptive mean-split clustering technique developed above.

Table 1 shows the results of these algorithms when clustering each of six test pictures into 64 and 256 clusters. Both the standard deviation of error per pixel and the CPU time are shown. It is clear that the mean-split algorithm is by far the fastest method. Moreover, it always produces better clusters than

the k-d tree method — sometimes with less than half the standard deviation of error of that method. Its error performance is around 30% worse than the k-means algorithm on the smaller clustering problems, and around 25% worse on the larger ones. However, it is two to three orders of magnitude faster.

It is clear that the k-means method, because of its iterative nature and the complexity of each iteration, will be much slower than the others. Let us consider why mean-split is faster than the k-d tree method. Finding means is an $O(N)$ procedure, while finding medians requires sorting and is hence (for most implementations) $O(N \log N)$. The kind of operations, addition versus comparison, will usually take similar amounts of computer time. The expected complexity of bi-partitioning a set of N m-variate data is $O(mN)$ for the new algorithm and $O(N \log N)$ for median cutting (k-d tree). Since $\log N >> m$ for most applications, the mean-split algorithm has overwhelming superiority over median cutting in speed, and, as Table 1 shows, in performance as well. Moreover, a factor of two speed-up is achieved by calculating the mean of only one subset of a bi-partition directly, and combining it with the overall mean to find that of the other subset. One caveat should be noted here. If the data type is integer, radix sorting techniques can be used [7]. This gives median cutting $O(N)$ complexity at the price of larger working space. Even though the color image data are integers, quicksort was used for the results of Table 1, for generality.

Another feature of the mean-split algorithm, which can be seen in Table 1, is that its speed is almost independent of the input data. Neither median cutting nor the k-means algorithm share this advantage; the former because the $O(N \log N)$ for sorting is only an average figure, and the latter because its convergence time depends heavily on both the initially-chosen cluster centers and the data distribution.

To give some idea of the effect of the differing performances in practice, test image 1 of Table 1 was reproduced using several clustering methods. Figure 4(a) shows the original digitized picture with 256×256 pixels, although it is drawn on a 512×512 grid by replicating each pixel four times. (This has the effect of exaggerating deficiencies in the rendering — jaggies are more prominent since the 2×2 blocks of pixels are nearly square.) The picture was digitized with 8-bit resolution in each of red, green, and blue, giving a potential palette of 2^{24} different colors. However, since there are only 2^{16} pixels, at most

this number of colors can be represented in the picture (in fact it happens to contain 60224 different colors). The other four parts of the figure show versions with only 64 different colors, produced by partitioning the 2^{16} points in 3-dimensional color space into 64 clusters. (b) was generated by the *k*-means method; (c) with the adaptive mean-split technique developed in this paper; (d) using the *k-d* tree as recommended by Heckbert [6]; and (e) by naively retaining only the top two bits of each color coordinate. (b) is very similar to the original (a), although close inspection shows some contouring in the flesh tones under the chin. The skin tones in (c) are somewhat inferior; visible contouring is increased and has spread to the nose and highlighted cheek. There is significant further deterioration in (d); now the skin tones are noticeably artificial. Finally, (e) shows how essential clustering is to get an acceptable rendition. Considering that version (c) takes less than one minute to calculate, while (d) takes two hours and (b) half a day, the new algorithm appears to attain a convincing compromise between quality and speed.

Theoretical analysis. To provide further evidence to support the adaptive mean-split algorithm, the quality of the approximation to ξ_{opt} of *Eq*-3b is studied for a particular class of probability distributions.

During the recursive spatial division, if the current data subset has an obvious distribution into two clusters, where the separation between the cluster centers exceeds the clusters' diameters, it is clear that a split at the mean is adequate. Otherwise, when the distribution appears fairly continuous, perhaps with one major peak, the choice of cutting position is much more difficult. Figure 5, which shows the three marginal distributions of test image 3, gives some idea of the problem faced in practice.

In order to model this situation, the Weibull distribution

$$f(x) = \alpha\beta\, x^{\beta-1}\exp(-\alpha x^{\beta}), \qquad x>0,\ \alpha,\ \beta>0,$$

is used. It is shown in Figure 6 for different values of α and β, and varying these parameters provides a good approximation to the general shape of the marginal data distributions encountered in practice.

The mean μ, median ξ_m and cumulative distribution $F(x)$ of the Weibull distribution are given by

$$\mu = \alpha^{-1/\beta}\Gamma\left[\frac{1}{\beta}+1\right] \qquad \xi_m = \left[\frac{\ln 2}{\alpha}\right]^{1/\beta} \qquad F(x) = 1 - \exp(-\alpha x^{\beta}).$$

To make the distribution finite, we calculate the value of x_0 for which $F(x_0)=0.999$ and use only that part of the distribution between 0 and x_0, normalizing it to [0, 1]. ξ_{opt} can be found from *Eq*-A1 in Appendix A. Figure 7 shows the position of μ, ξ_m and ξ_{opt}, plotted against β. Surprisingly enough the graph turns out to be virtually independent of the value of α, although the actual error $e[P(S,K)]$ decreases as α increases and the data set thereby becomes more clustered.

Figure 7 supports all the conclusions of Appendix A. The relative position of the three curves indicates that μ is a better estimator of ξ_{opt} than ξ_m. Increasing β drags the peak of the distribution toward the center (see Figure 6) so that it becomes more symmetric. Then μ and ξ_m approach ξ_{opt} and eventually become equal. The greater the departure from symmetry, the further from ξ_{opt} are both μ and ξ_m; however μ remains a better estimator than ξ_m.

Next, consider the coefficient η in *Eq*-3b and, for comparison, the corresponding coefficient λ of *Eq*-3a. While the implementation of the algorithm treats η as if it were constant, it can be expressed as a function of the parameters of the distribution to make *Eq*-3b exact. $\eta(\beta)=\frac{\xi_{opt}-\mu}{c-\mu}$, and also $\lambda(\beta)=\frac{\xi_{opt}-\mu}{\mu-\xi_m}$, are plotted against β in Figure 8. λ varies by only 17% over the entire family of Weibull distributions and should be set to $\lambda\approx 1.7$ if *Eq*-3a were used. Although (as suggested above) η is more sensitive to the individual distribution, it has a fairly limited range, from 0.05 to 0.25. Experiments show that choosing η within the above range for *Eq*-3b gives a noticeable improvement in $e[P(S,K)]$ over using $\eta=0$ and $\xi_{opt}=\mu$.

Conclusions

A fast algorithm for clustering has been derived, implemented, and analyzed. It provides a tool for large clustering problems which occupies a new and important place in the optimality/complexity tradeoff, thereby opening up new areas of application -- particularly in image processing. The key to its success is

the use of subdivision in both the spatial and quota domains. Spatial subdivision is achieved rapidly with a good approximation, but deficiencies arise partly from the use of hyperplanes parallel to a coordinate axis and partly from approximating the optimal cutting point. These are corrected by appropriate allocation of the quota between the two parts. While it does not guarantee to find a local optimum, the new algorithm's performance in practice is very good indeed. In situations where optimality guarantees are essential, it provides an excellent and inexpensive starting-point for further iteration.

There is another potential advantage of the adaptive mean-split clustering algorithm. It is simple to record the successive subdivisions it makes in the form of a tree structure (similar to the k-d tree). However, a more informative record can be kept by storing with each subdivision a list of neighboring clusters. This can be done during the clustering process without any calculation of Euclidean distances. Combined with the technique of m-dimensional Voronoi diagrams, such lists can be used to expedite nearest-neighbor queries. For example, to implement the k-means algorithm, the new method not only provides excellent starting cluster centers but also produces a data structure which greatly assists the nearest-neighbor calculations required in further iterations. This is the subject of current research.

References

[1] M.R. Anderberg, *Cluster analysis for applications.* New York: Academic Press, p. 3, 1973.

[2] J.L. Bentley, "Multidimensional divide-and-conquer," *Communications of the Association for Computing Machinery,* vol. 23, no. 4, pp. 214-229, April 1980.

[3] J.H. Friedman, J.L. Bentley and R.A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans Mathematical Software,* vol. 3, no. 3, pp. 209-226, Sept. 1977.

[4] E.L. Hall, *Computer image processing and recognition.* New York: Academic Press, 1979.

[5] J.A. Hartigan, *Clustering algorithms.* New York: Wiley, 1975.

[6] P. Heckbert, "Color image quantization for frame buffer display," *ACM Transactions on Computer Graphics,* vol. 6, no. 3, pp. 297-307, July 1982.

[7] D.E. Knuth, *The art of computer programming 3: Sorting and searching.* Reading, Massachusetts:

Addison Wesley, 1973.

[8] J.B. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proc Fifth Berkeley Symposium on Mathematical Statistics and Probability,* vol. 1, pp. 281-297, 1967.

[9] S.Z. Selim, and M.A. Ismail, "K-means-type algorithms: a generalized convergence theorem and characterization of local optimality," *IEEE Trans on Pattern Analysis and Machine Intelligence,* vol. PAMI-6, no. 1, pp. 81-87, Jan. 1984.

[10] R.J. Stevens, A.F. Lehar and R.H. Preston, "Manipulation and presentation of multi-dimensional image data using the Peano scan," *IEEE Trans Pattern Analysis and Machine Intelligence,* vol. 5, no. 5, pp. 520-, Sept. 1983.

[11] I.T. Tou and R.C. Gonzalez, *Pattern recognition principles.* Reading, Massachusetts: Addison-Wesley, 1974.

Captions for figures

List of tables

Appendix A: Rationale for the approximation to the optimal cutting point

Suppose a one-dimensional data set with continuous probability density $f(x)$ is split at point $x=\xi$. The centroids (means) of the two parts are

$$E_1(\xi) = \frac{\int_{-\infty}^{\xi} xf(x)dx}{\int_{-\infty}^{\xi} f(x)dx} \quad \text{and} \quad E_2(\xi) = \frac{\int_{\xi}^{\infty} x\ f(x)dx}{\int_{\xi}^{\infty} f(x)dx} .$$

The mean squared error e over both partitions is

$$e = \int_{-\infty}^{\xi} [x-E_1(\xi)]^2 f(x)dx + \int_{\xi}^{\infty} [x-E_2(\xi)]^2 f(x)dx .$$

The optimal cutting position $\xi=\xi_{opt}$ is the value which minimizes this expression. To find this, consider the derivative

$$\frac{\partial e}{\partial \xi} = 2 f(\xi) [E_1(\xi) - E_2(\xi)] \left[\frac{E_1(\xi)+E_2(\xi)}{2} - \xi \right] .$$

Since $f(x)\geq 0$ and $E_1(\xi)\leq E_2(\xi)$, the optimal cutting point ξ_{opt} must satisfy

$$\xi_{opt} = \frac{E_1(\xi_{opt}) + E_2(\xi_{opt})}{2} . \qquad Eq\text{-}A1$$

It is simple to check that this equation has a unique solution, provided only that the second derivative of $f(x)$ exists. Notice that $x=\xi_{opt}$ represents the half-plane of the one-dimensional Voronoi diagram of $E_1(\xi_{opt})$ and $E_2(\xi_{opt})$.

The proximity of different approximations to ξ_{opt} can be measured by the function $B(\xi)$:

$$B(\xi) = \frac{E_1(\xi)+E_2(\xi)}{2} - \xi .$$

Because $B(\xi_{opt})=0$ and $B(\xi)$ is monotonically decreasing, its sign indicates on which side of ξ the optimal cutting position ξ_{opt} lies (if $B(\xi)\geq 0$ then $\xi\geq\xi_{opt}$ and *vice versa*). It can be re-expressed as

$$B(\xi) = \frac{\mu-E_1(\xi)}{2(1-F(\xi))} + E_1(\xi) - \xi,$$

where F is the cumulative distribution of f and μ is its mean. For a symmetric density, $F(\mu)=1/2$ and so $\xi_{opt}=\mu$ because $B(\mu)=0$.

Denote by ξ_m the median of f, so that $F(\xi_m)=1/2$ and

$$B(\xi_m) = \mu - \xi_m .$$

In the case of a symmetric distribution, the mean and median are identical and both are the optimal cutting point. The main result is that for non-symmetric distributions, the mean is a better estimate of ξ_{opt} than the median.

Theorem: For any continuous $f(x)$, ξ_m cannot be closer to ξ_{opt} than μ.

Proof: Since for any ξ, μ $E_1(\xi) \geq 0$, it follows that

If $\mu > \xi_m$, then $B(\xi_m) > 0$ and $F(\mu) > 1/2$; hence $B(\mu) > 0$ and so $\xi_{opt} > \mu$.

Likewise, if $\mu < \xi_m$, then $\xi_{opt} < \mu$.

Consequently, either $\xi_{opt} \geq \mu \geq \xi_m$ or $\xi_{opt} \leq \mu \leq \xi_m$.

Table 1 Experimental clustering results on test images

image	error (standard deviation)			CPU time (hours:minutes:seconds)		
	mean-split	*k-d* tree	*k*-means	mean-split	*k-d* tree	*k*-means
64 clusters						
1	5.49	8.04	4.24	51	2: 3:52	14:53: 4
2	2.85	6.26	2.29	53	53:36	12:43:41
3	9.03	9.76	7.12	53	20:10	9:19:39
4	8.68	10.87	6.37	1: 5	23:28	15:53:34
5	10.11	12.96	7.71	49	28:43	13:39:33
6	10.68	12.63	8.59	52	19:22	8:46: 2
256 clusters						
1	2.78	4.47	2.27	1:11	2:10:30	25:45:34
2	1.32	2.71	1.18	1: 8	1: 3: 2	12:33: 0
3	5.33	6.12	4.24	1:19	21:13	23: 4:17
4	4.78	6.21	3.71	1:22	26:46	24:30:32
5	6.16	7.35	4.86	1: 1	30:53	34:18:38
6	6.75	7.73	5.5	1:10	20:21	27:34:38

```
const K = the number of partitions desired;
      N = the number of points in the data set S;
      M = the dimensionality of the data set S;
type  dimension = 1..M;
      vector = array[dimension] of real;
var   centroid: array[1..K] of vector;                    { centroid vectors for K clusters }
      newtag:  integer;                                   { the partition number for the next data subset }

{ initialize newtag to 0 and then call partition(K, N, 0, S) with centroid[0] set to the centroid of S }

procedure partition(k, n, tag: integer; dataset: array[1..N] of vector);
                                  { k, n, tag are the quota, population and partition number of the dataset }
  var n1, n2: integer;                                    { populations of subset1, subset2 }
      subset1, subset2: array[1..N] of vector;

  procedure divide(cutaxis: dimension);                   { cutaxis is the dimension along which to cut }
    var i: integer;
    begin
        cutplane := centroid[tag] + η*(ξ_c - centroid[tag]);
                                                          { determine where to cut, using Eq-3b }
        newtag := newtag+1;                               { allocate a new partition }
        centroid[newtag] := 0;
        n1 := n2 := 0;
        for i := 1 to n do                                { perform the division into two subsets }
            if dataset[i, cutaxis] < cutplane then begin
                n1 := n1+1;
                subset1[n1] := dataset[i];
                centroid[newtag] := centroid[newtag]+dataset[i];
            end
            else begin
                n2 := n2+1;
                subset2[n2] := dataset[i];
            end
        centroid[newtag] := centroid[newtag]/n1;          { calculate new centroid }
        centroid[tag] := (n*centroid[tag] - n1*centroid[newtag])/n2;
    end

  procedure hyperbox(dataset);                { finds the smallest bounding hyperbox of the dataset }
  function widest_dimension(hyperbox): dimension;         { finds which dimension is widest }
  function quota1: integer;
                                  { computes the quota for subset1, based on n1, n2, s1, s2, k, β, using Eq-4 }

  begin
      if k=1 then return;
      else if smaller_than_minimum_size(hyperbox(dataset)) then
          save extra rations for large subset and return;
      else begin
          divide(widest_dimension(hyperbox));
          q1 := quota1; q2 := k - q1;
          partition(q1, n1, newtag, subset1);
          partition(q2, n2, tag, subset2);
      end
  end
```

Figure 1

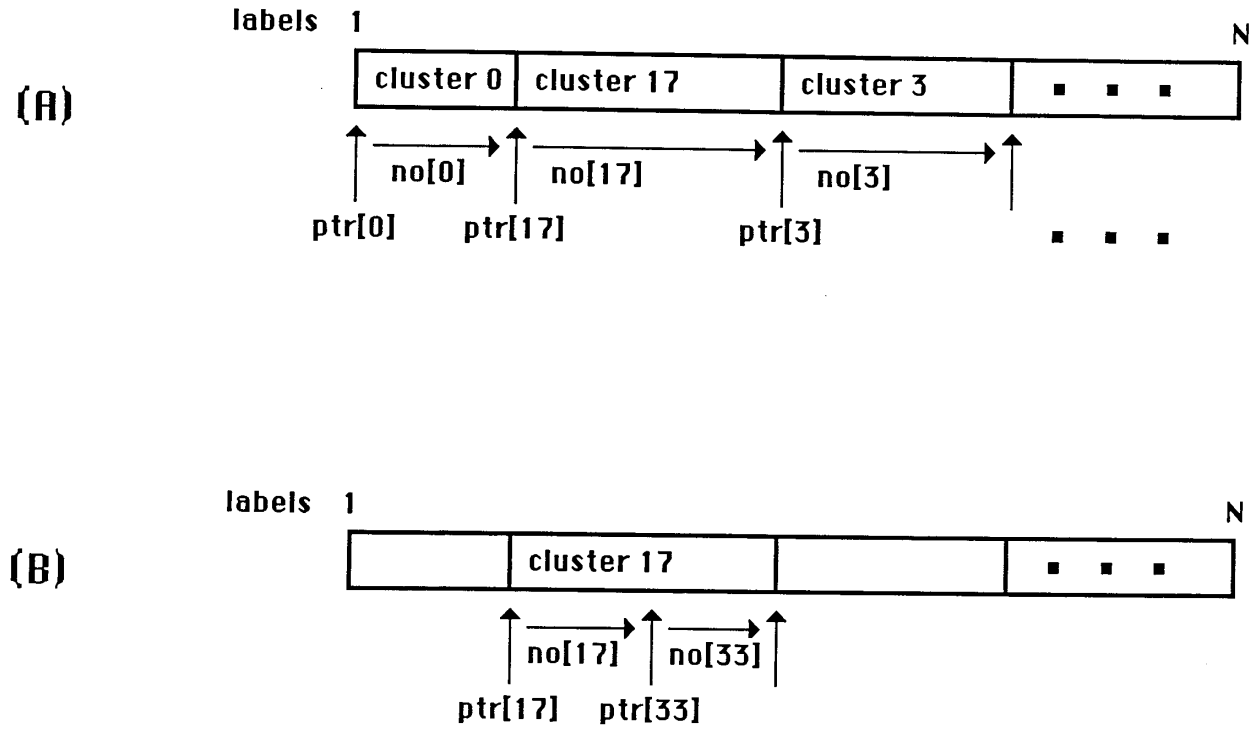
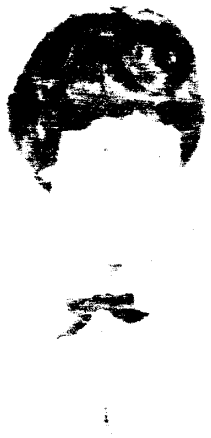

(A)
labels 1
N
cluster 0
cluster 17
cluster 3
no[0]
no[17]
no[3]
ptr[0]
ptr[17]
ptr[3]
(B)
labels 1
N
cluster 17
no[17]
no[33]
ptr[17]
ptr[33]

Figure 2

```
procedure divide(cutaxis: dimension);                { cutaxis is the dimension along which to cut }
  var p1, p2, t: integer;
  begin
      cutplane := centroid[tag] + η*(ξ_c - centroid[tag]);
                                                     { determine where to cut, using Eq-3b }
      newtag := newtag+1;                            { allocate a new partition }
      centroid[newtag] := 0;
      p1 := ptr[tag];
      p2 := p1+no[tag]-1;
      while p1 < p2 do begin                         { perform the shuffle into two subsets }
          while data[label[p1], cutaxis] < cutplane do p1 := p1+1;
          while data[label[p2], cutaxis] >= cutplane do begin
              centroid[newtag] := centroid[newtag] + data[label[p2]];
              p2 := p2-1;
          end
          if p1 < p2 then begin
              centroid[newtag] := centroid[newtag] + data[label[p1]];
              swap(label[p1], label[p2]);
              p1 := p1+1;
              p2 := p2-1;
          end
      end
      ptr[newtag] := p1;
      no[newtag] := no[tag] - (p1 - ptr[tag]);
      no[tag] := no[tag] - no[newtag];
      centroid[newtag] := centroid[newtag]/no[newtag];
      centroid[tag] := (n*centroid[tag] - no[newtag]*centroid[newtag])/no[tag];
  end
```

Figure 3

Figure 4 (a)



(b)

(c)

(d)

(e)

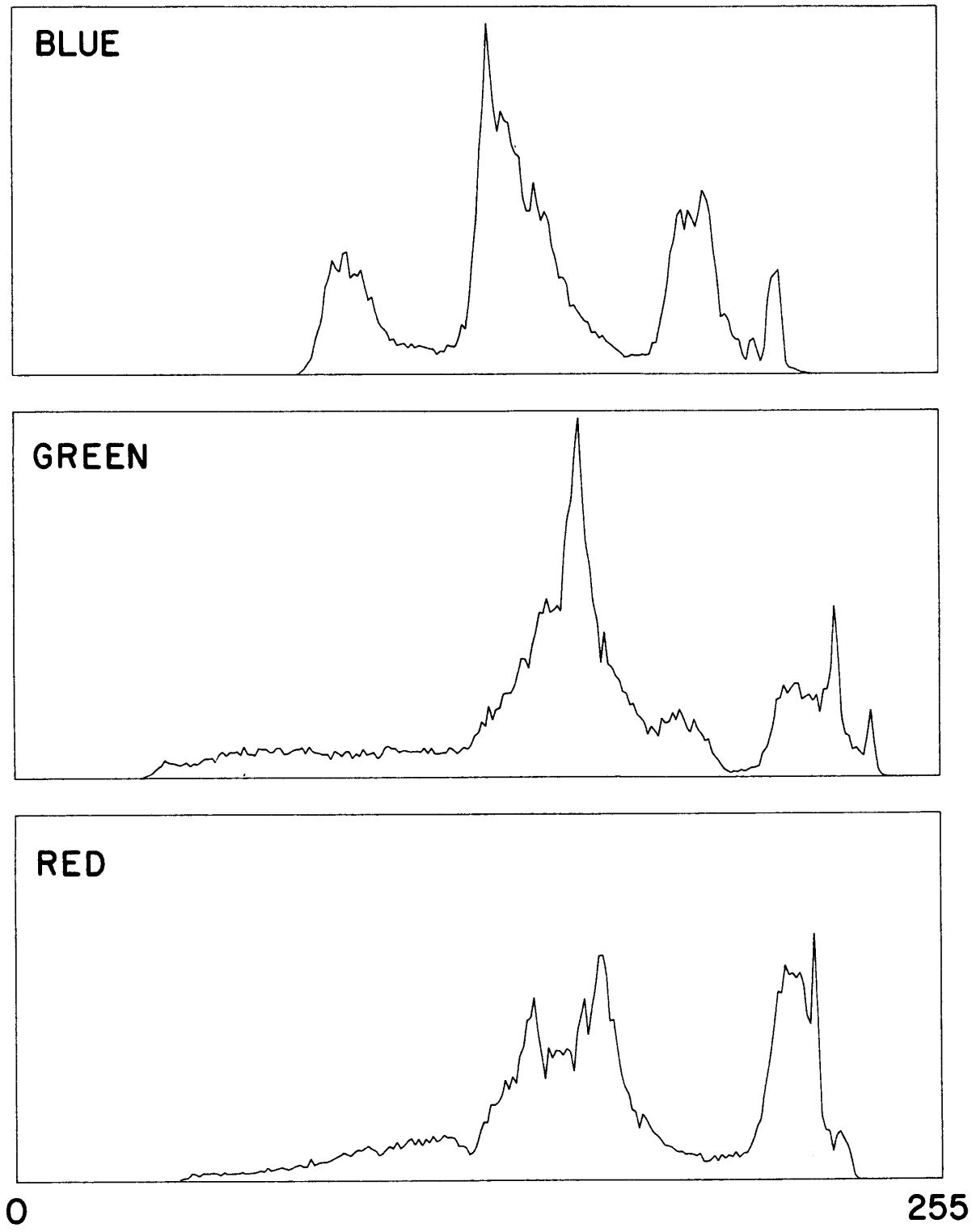


Figure 5

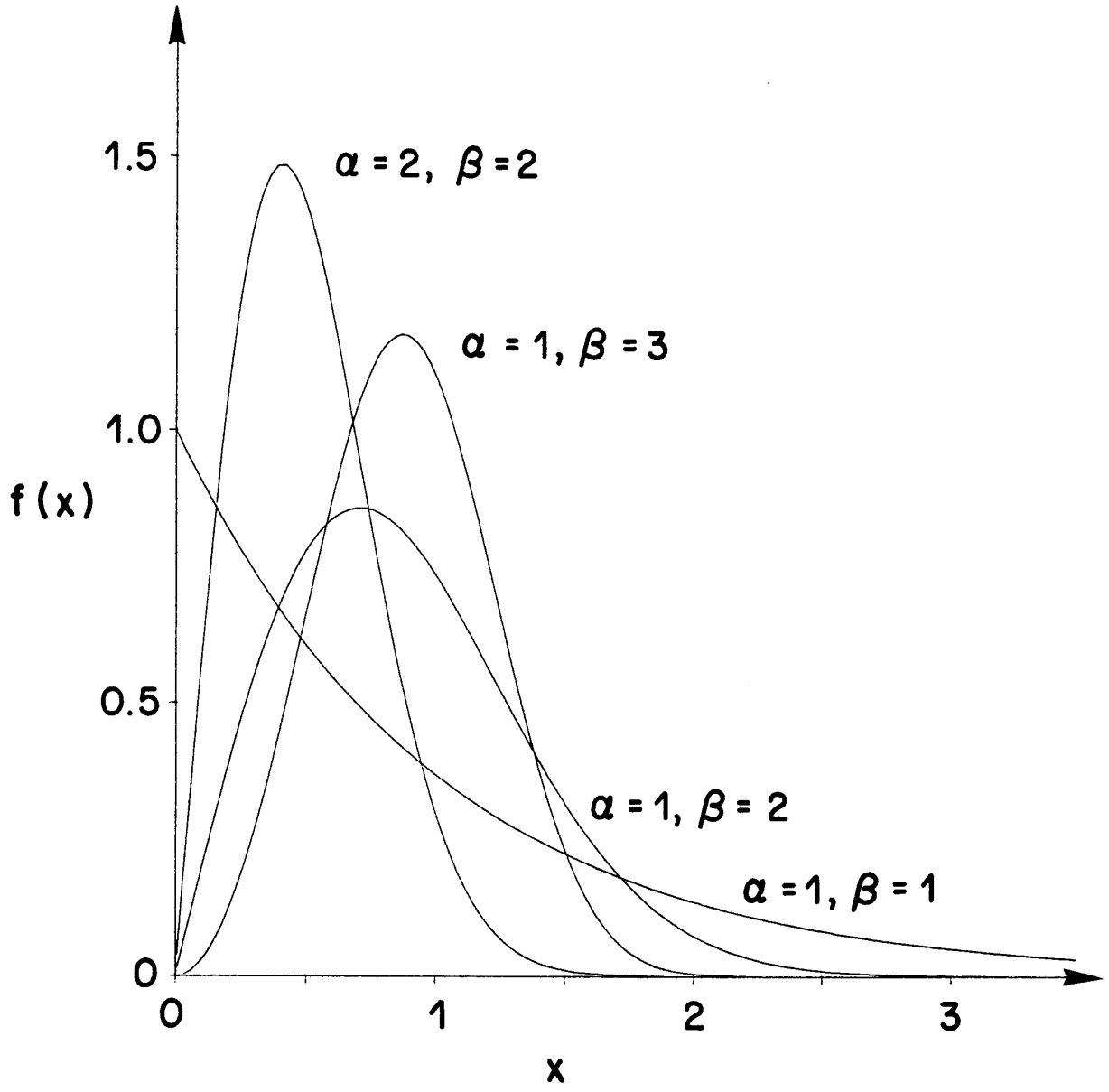


Figure 6

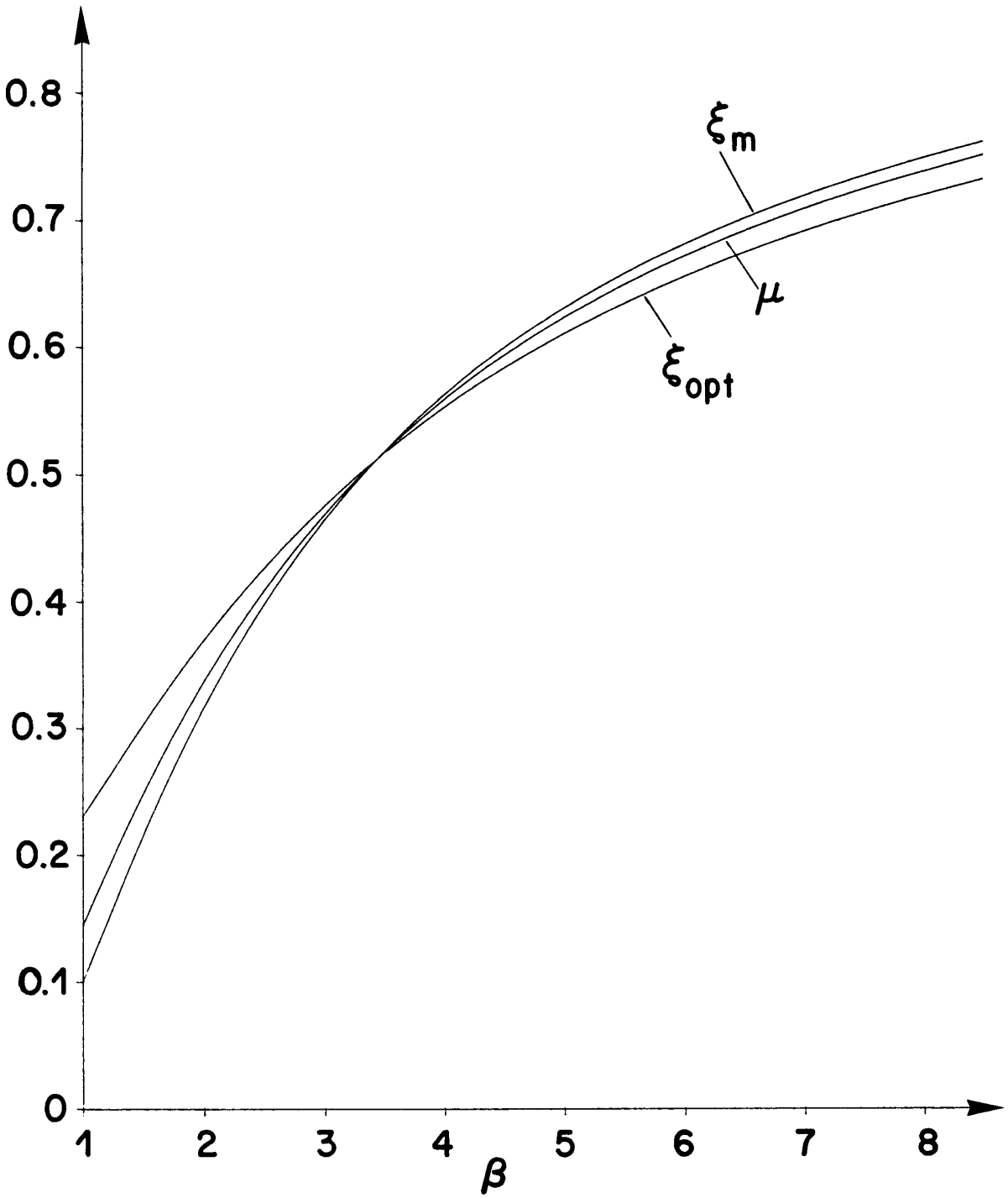


Figure 7

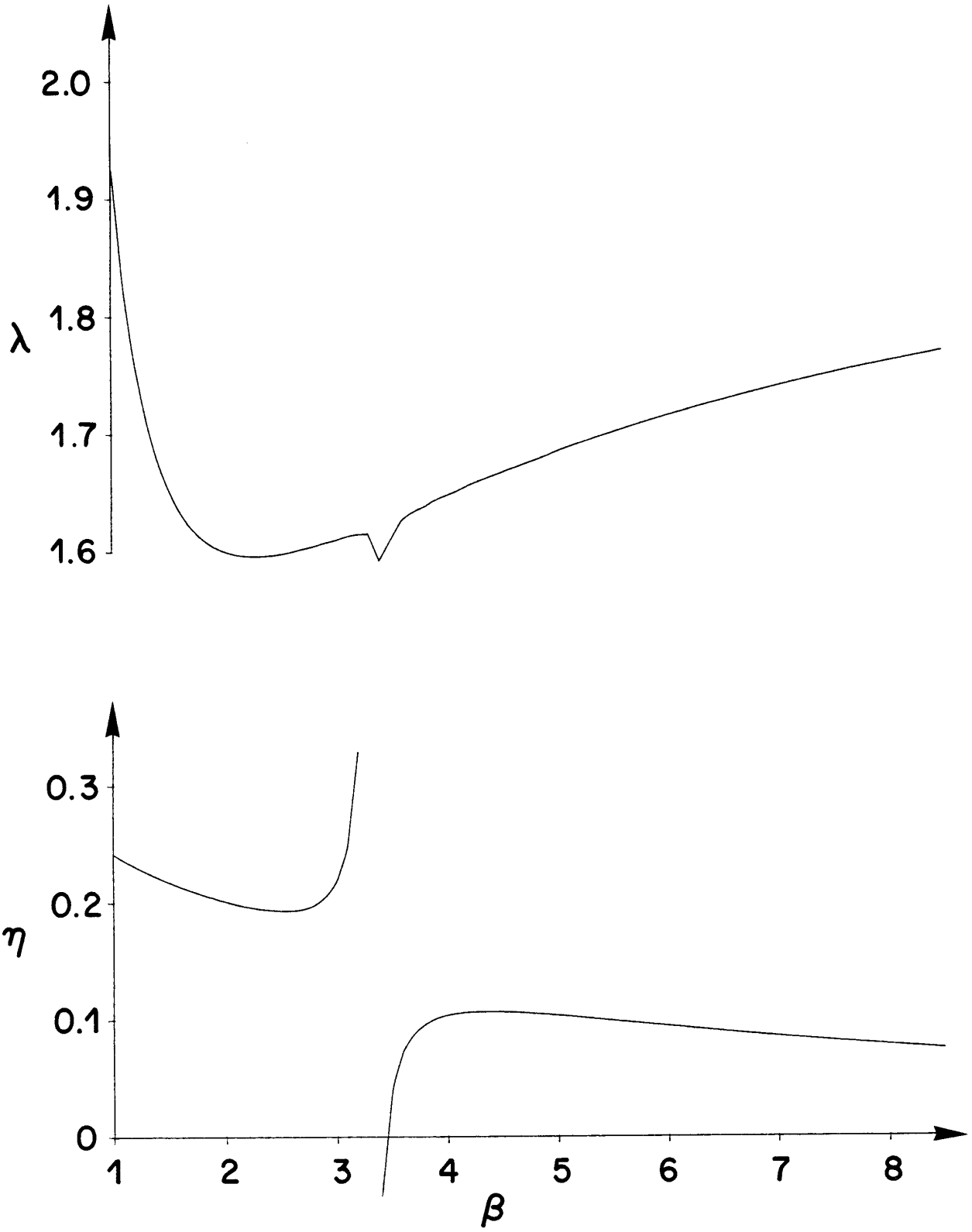


Figure 8