EPCC-SS97-03

Cluster Analysis using Genetic Algorithms

Daniel Chen

Abstract

Existing methods for cluster analysis typically define membership, the actual clustering of a data set, and granularity, the number of clusters, as two separate problems. The typical approach to clustering is to group data points based on their similarity. A genetic algorithm is proposed as an efficient method of optimizing both membership and granularity in cluster analysis. Furthermore, the clustering problem is approached not as a grouping problem but as an optimization of the arrangement of cluster centers.

1 Introduction

Cluster analysis is a statistical technique that finds inherent structure within data sets. Among the many methods that can find such clustering of data, optimization methods stand out because of their flexibility and versatility relative to other methods. Given some numerical criterion, an optimization method such as a genetic algorithm can maximize or minimize the criterion to find the optimum solution. This numerical criterion is some evaluation of the data set and its clustering. Differences in numerical criterion give rise to the many possible methods.

More specifically, optimization methods for cluster analysis deal with two issues — membership and granularity. Membership is the problem of determining the best locations for cluster centers. Another way to think about it is deciding which data points are similar enough to group together and treating cluster centers as group means. Granularity is the problem of discovering the proper number of clusters. Fine-grained granularity is a preference for a large number of small clusters (figure 2). Coarse-grained granularity is the opposite, a preference for a small number of large clusters (figure 3).

Figure 1: Unclustered household population data

Historically, optimization methods deal with these two issues separately. First, they cluster the data based on some clustering criterion which is maximized or minimized. Then, they use another criterion to evaluate each solution and determine the best number of clusters. A method utilizing genetic algorithms is proposed as a way to deal with both issues at once. Given some preference for granularity, various techniques were investigated and synthesized to discover both the optimal clustering and cluster number. Another unique characteristic of the proposed approach is that it focuses on finding cluster centers rather than the clustering of data points.

Figure 2: Coarse-grained granularity

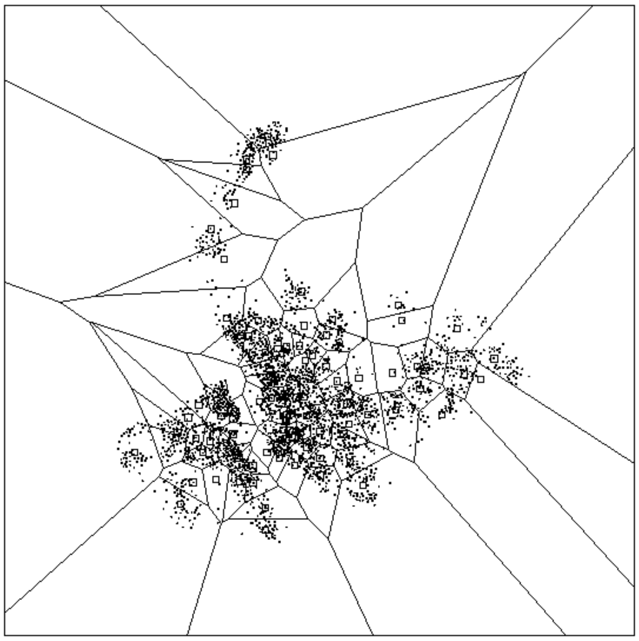

Figure 3: Fine-grained granularity

2 Background

2.1 Cluster Analysis

Clustering and the need for clustering is perhaps inherent in human nature and society. Probably the earliest examples of clustering are found in taxonomy, where one classifies plants and animals based on salient characteristics. Clustering did not start out as a statistically rigorous technique, rather its main purpose was for survey data. This still applies today, as most applications of clustering are in the social sciences and medicine. The technique commonly used in those areas are called hierarchical methods. This is usually done in one of two ways. One can either start with each individual data point being its own cluster and aggregate clusters one by one. Or, one can begin with a large cluster incorporating everything and then subdividing the cluster into smaller clusters.

Another technique for cluster analysis involves optimization. Rather than imposing a hierarchical structure on the data, these methods optimize some numerical criterion that measures the quality of a given data set and its clustering. Often, the clustering criterion is based on a measure of distance between data points and between cluster centers. The most common criterion is the sum of the squares of the Euclidean distances between each data point and its group mean or cluster center. This however has drawbacks as it imposes a spherical structure on the clustering and looks for clusters of equal size. It is also scale dependent. There exists several other criterion that attempt to remedy the scaling, size, and shape problems but require significantly more computational and mathematical exposition than the scope of this report.

The typical application of cluster analysis involves visualization and quantification of data characteristics. Any pre-processing information obtained from visualization can only help the actual clustering process, often accomplished by existing software packages. The primary purpose of clustering, however, is not to test hypotheses, but to help form new ones. Because the clustering methods outlined above are not statistically rigorous, cluster analysis is usually a starting point to investigating data and answering questions [1].

2.2 Genetic Algorithms

Function optimization with many dimensions is a diffcult problem. A robust technique must be able to both converge on a global optima as well as search the entire space for a possible solution. Doing both simultaneously is a nontrivial task. Optimization becomes computationally expensive when the derivative of the objective function is not analytical or is expensive to compute. Methods for numerical optimization range from the very simple, such as steepest descent, to much more complex algorithms. Two methods (besides the genetic algorithm) which do not need derivatives are investigated in this project and are mentioned briefly below.

The first method is called the simplex method. Here, the idea is to keep track of a set of points and at each iteration, replace the worst element with a new element created from all the elements of the previous set. The new element is usually created via a reflection of the worst point through the centroid of all the other points, or contraction around the best point. This process continues until the simplex hopefully finds the global optimum [2].

Another recently popularized method is simulated annealing. Its analogy is of heating up metal to a high temperature and then gradually cooling, or 'annealing', the metal to ensure the lowest amount of internal energy. General simulated annealing is an algorithm that keeps track of one point. With each iteration, a candidate is chosen at random from the previous point's neighborhood. If the new fitness evaluation is better, then the candidate is kept and the previous point replaced. If it is worse, the candidate replaces the previous point with a certain probability. The higher the temperature, the higher the probability of taking these poor steps. It is these poor steps that allow the algorithm to climb out of local optima and find the global one [2].

The idea behind genetic algorithms comes from biological evolution. Most organisms evolve through natural selection or sexual reproduction. Natural selection works by selecting the fittest individuals for survival and reproduction. The fittest individuals can then propagate their own genes to future generations through reproduction. Sexual reproduction between two parents occurs through genetic crossover, so each parent passes on some genetic information to the child. Over many generations, a population of individuals will evolve and adapt to a particular fitness landscape. Similarly, a genetic algorithm will find a numerical optimum by evolving a population of individuals, characterized by their genomes. The genomes represent all possible solutions. Genomes with better fitness values are selected for reproduction, and child genomes retain good characteristics of their parent genomes. Furthermore, diversity of the population is maintained through random mutations. The population should converge over many generations to some optimum value for the fitness function as fitter characteristics are selected for.

The pseudo-code for a generic genetic algorithm [3]:

```
BEGIN /* genetic algorithm */
  generate initial population
  compute fitness of each individual

  WHILE NOT finished DO
  BEGIN /* produce new generation */

    FOR population_size / 2 DO
    BEGIN /* reproductive cycle */
      select two individuals from old generation for mating
        /* biased in favor of the fitter ones */
      recombine the two individuals to give two offspring
      compute fitness of the two offspring
      insert offspring in new generation
    END

    IF population has converged THEN
      finished := TRUE

  END
END
```

To implement genetic algorithms, this project used RPL2, the Reproductive Planning Language [4]. It comes with several built-in representations and operators and is designed to run on a variety of computer architectures in serial or parallel. Moreover, the user can create new representations and operators by creating additional C libraries. The most crucial aspect of applying genetic algorithms is the choice of representation and operators.

Although there have been many applications of simulated annealing to cluster analysis (see for example, Brown *et. al.* in [5]), surprisingly, the application of genetic algorithms to cluster analysis is relatively rare. Genetic algorithms have been applied to fuzzy clustering [6] and the K-means algorithm [7]. K-means is a popular simple algorithm that at every iteration, looks to reassign individual data points and then recalculates the appropriate cluster means. As previously noted, one of the initial motivations for the project was to see how genetic algorithms could be used for cluster analysis.

The question the project attempted to answer was how genetic algorithms can be used for cluster analysis. More specifically, can the genetic algorithm find not only the optimum cluster center distribution but also the best number of clusters given some preference for granularity. The project's aim was to create objective functions that would allow one to control the number of cluster centers indirectly. The work involved investigating, modifying, and synthesizing several existing methods for cluster analysis.

3 Approach

Membership is defined here as the grouping of data points into clusters. In this project, optimizing the membership function is equivalent to finding the best arrangement of cluster centers. Granularity refers to the density of clusters. Fine-grained granularity indicates a preference for a large number of small clusters. Coarse-grained granularity indicates a preference for a few number of large clusters.

3.1 Membership

The approach taken is to minimize the sum of distances from each data point to its cluster center. This can be expressed as:

$$E = \sum_{i=1}^{n} d_{i,c(i)}^{p}$$

where $d_{i,c(i)}$ is the Euclidean distance from individual i, to its cluster mean or center $c(i)$. If each individual has a corresponding weight, the distance is multiplied by this weight. This allows us to take account of "density"-based data.

The most commonly used criteria is $p = 2$ [1]. However, it was hypothesized and shown that this criteria would pull cluster centers away from the main group of points and give outliers too much influence. It was noted that smaller exponents p would alleviate this potential problem by giving less weight to the outliers. Indeed, with $p = 1$, the outliers were ignored and the data points themselves were frequently chosen as cluster centers (see figure 4), which is not necessarily desirable.

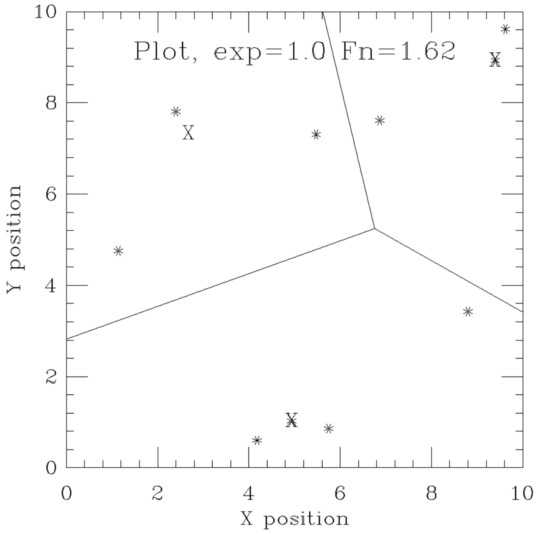

Figure 4: Two different membership functions, 10 data points, 3 cluster centers

With larger data sets, the exponent seemed to matter much less in finding good clustering. However, with larger numbers of cluster centers, $p = 1$ tends to subdivide dense areas too much. The data set shown in figure 5 is household data for Merseyside. Each dot represents about 500 households. The number of households is the weight that multiplies the corresponding distances. The left shows $p = 1$ and the right shows $p = 2$. It was noted how $p = 1$ seemed to subdivide dense areas more than $p = 2$.

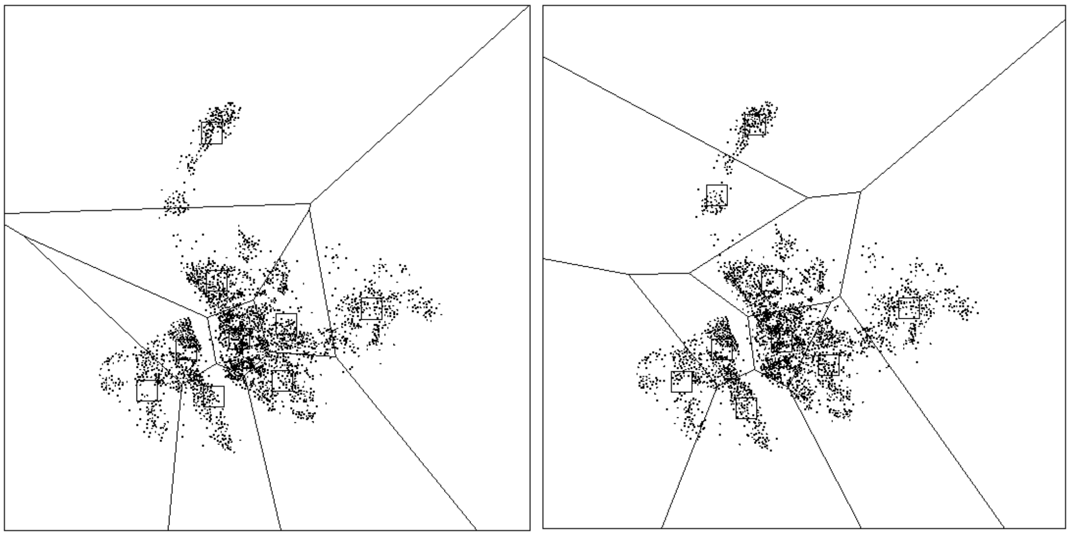

Figure 5: Two different membership functions, 2814 data points, 9 cluster centers

Initially, an exhaustive search was used to find the optimum. Starting with this naive algorithm allowed a more thorough testing of the membership function for various random and non-random distribution of data points. As this is only feasible for very small number of cluster centers and data points, simulated annealing was used instead for testing larger data sets. Simulated annealing was implemented with the typical simplex algorithm by adding noise to each function evaluation. At each iteration, the higher the corresponding temperature, the more random noise was added to each point of the simplex. Over time, as the temperature decreased, the algorithm approached a true simplex algorithm, as described above. The algorithm was taken from Numerical Recipes [2].

Exhaustive search algorithm:

```
DO i=0, (GridSize**NumClusters-1)
  DO j=0, (GridSize**NumClusters-1)
    Cluster = Cluster_of(i, j, NumClusters,
    &  MinX, MaxX, MinY, MaxY, GridSize)
    Grid(i,j) = Fitness_of(Point,Cluster,NumPoints,
    &  NumClusters,exp)
  END DO
END DO

Position = MINLOC(Grid)
Fitness = MINVAL(Grid)

Cluster_of converts i and j, array indices of 'Grid', into clusters,
using NumClusters, mins and maxes, and GridSize.
Fitness_of returns the evaluation of the membership function based on
Point, the array of points, Cluster, the array of clusters, and
NumPoints, and NumClusters.
```

Simulated annealing algorithm:

```
DO for all 2*NumClusters number of points in the simplex
  select random points from the data set to be initial points in the simplex
  keep track of function evaluations
END DO

initialize temperature and number of iterations per temperature

DO twice
  initialize annealing schedule
  DO for number of iterations per temperature
    decrease temperature
    call function which executes one iteration of simplex algorithm
    keep track of historically best solution
  END DO
  IF this is first annealing
    set up second annealing simplex with historically best solution
  END IF
END DO
```

Simplex algorithm, one iteration:

```
/* Whenever function evaluation is needed, random noise is added,
proportional to the current annealing temperature */

Determine highest (worst), second highest, and lowest (best) points.
Compute vector average of all points except the highest, i.e. the
  center of the face of the simplex across from the high point.
  Subsequent explorations occur along the ray from the high point to
  the center.
First try reflection of high point through center.
If result is better than best point, move further half the previous move.
If current attempt is worse than the second highest, contract along
  the ray to half the distance.
If current attempt is worse than the highest point,
  contract simplex around the lowest point.
Else
  Replace old high point and exit.
End if
```

In an effort to normalize the membership function with respect to the scale of the data points, the coordinates of the data set were scaled to a range limited by 0 and 1. This made it possible to use the same annealing schedule for any data set. The membership function was already normalized with respect to the number of data points and the exponent of distance.

3.2 Granularity

Most literature addresses granularity by focusing on within-cluster distance measures and between-cluster distance measures. Within-cluster distance is frequently the sum of squares of the distances within a cluster, such as the function used for membership above. Between-cluster distance means the sum of squares of the distances between cluster centers or means. The method used was a minimization of within-cluster to between-cluster distance ratio:

Given a partition of n individuals into K partitions,

$$DB(K) = (1/K) \sum_{k=1}^{K} R_k \text{ for } K > 1$$

$$\text{where } R_k = \max_{(j \neq k)} \{R_{j,k}\} \text{ and } R_{j,k} = \frac{e_j + e_k}{m_{j,k}}$$

e_j is the average within-cluster distance for the jth cluster and $m_{j,k}$ is the Euclidean distance between cluster center j and k. The smaller DB(K) (Davies-Bouldin index), the better the clustering [8].

For the household data set, this function found a global minimum at five with a close second best at four cluster centers. Nine and twenty were local minimums. See figure 6 below:

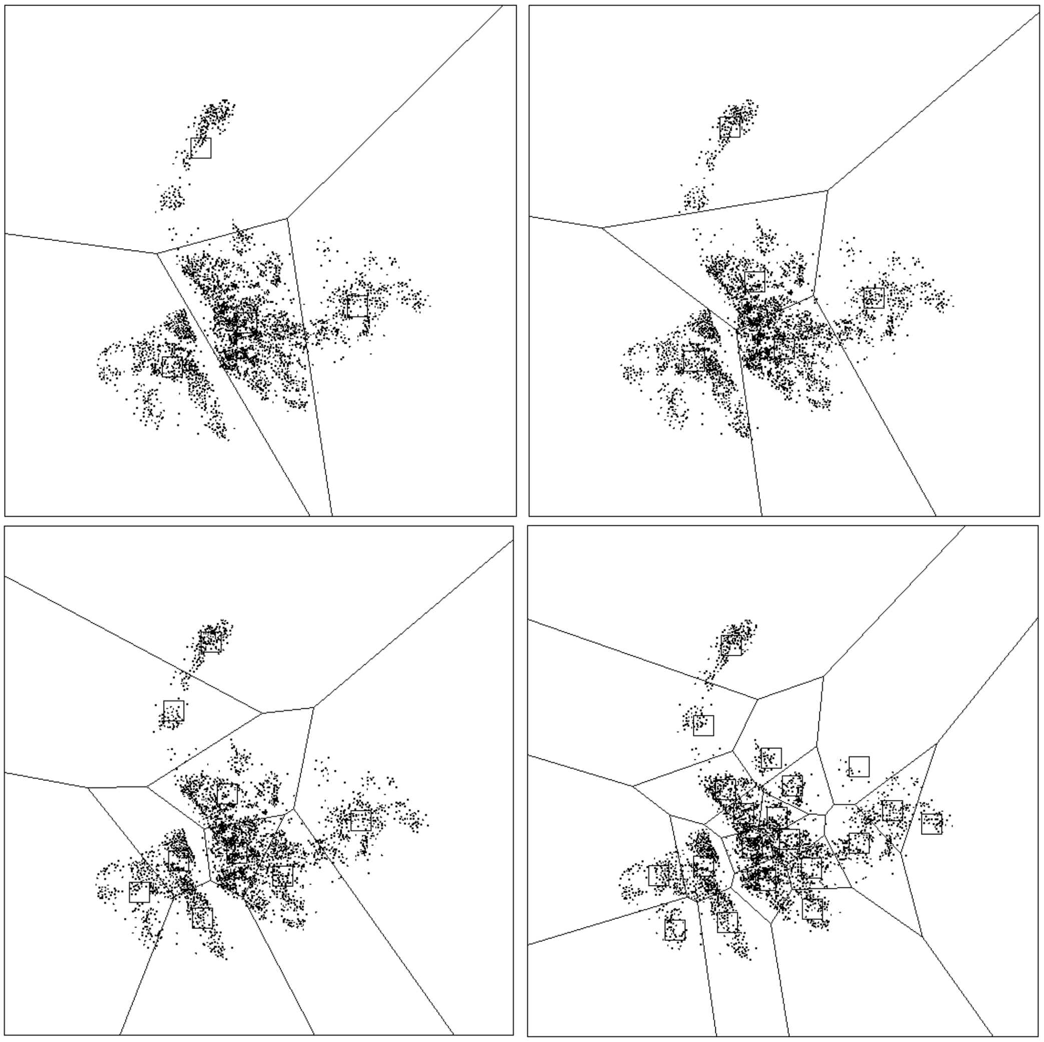

Figure 6: Four possible granularities. 4, 5, 9, and 20 cluster centers.

The algorithm used for the above began by using the membership function to optimize clustering for a large range of cluster numbers. Then, applying the granularity function, one can pick out the global minima as the optimal number of clusters. To implement preferences for granularity, the granularity function, which was normalized by the number of cluster centers, can actually be divided by some power greater than one of the cluster number. If the function is plotted on an x-y coordinate system where the x-axis is the number of cluster centers and the y-axis is the granularity function evaluation, then this division basically tilts the function downwards on the right. Thus, larger numbers of clusters are more likely to become the global minimum.

The granularity function was also tested on homogenous data to see if the function would indeed choose a distribution that evenly divides up the data set. A 50 by 50 grid of points was created, all with equal weight. The membership function was then used to optimize the clustering for cluster numbers ranging from 1 to 30. Since the granularity function must be run on cluster numbers greater than 1, the best distributions should be cluster numbers that are squares. This was so, as one can see from the following Voronoi diagram (figure 7):

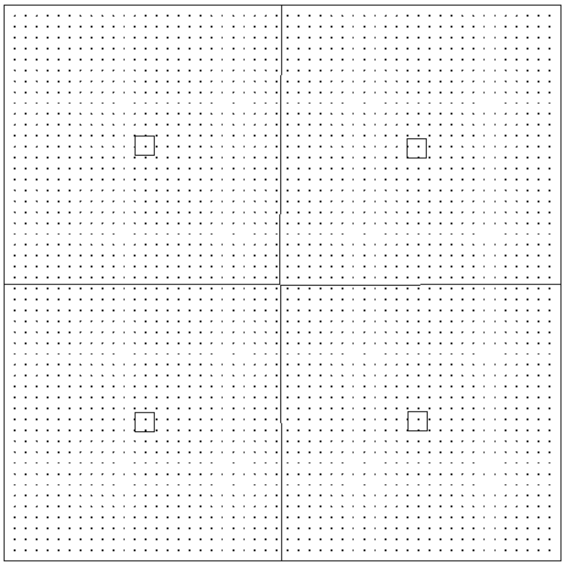

Figure 7: Four cluster centers for a 50 by 50 point grid.

3.3 Implementation of Genetic Algorithm

As mentioned earlier, the genomic representation of solutions is the most crucial aspect of using a genetic algorithm. The representation used here was the set representation. The elements of the set were array indices representing the cluster centers. The set representation had the advantages that solutions were not duplicated through ordering of cluster centers. Also, operators that modify the number of cluster centers were easily implemented. The drawback was that the set representation was limited to a finite search space. Each element of the universal set was a predefined possible cluster center. The universal set was also the set of data points to be clustered. This forced approximation, however, should be mitigated by a large number of data points.

The genetic algorithm used tournament-style parental selection. This mean for each parent, two individuals were selected at random from the population. Out of these two individuals, the fitter individual was selected (with a certain probabilty) to be a parent. A child was created from the parents by taking the union of the two parent sets. From this union, each element had a probability of being in the child set. The result was then adjusted by random additions or deletions to fit the appropriate set range. Mutation was achieved by randomly swapping an included element with an excluded element or through random additions or deletions of elements. At the end of the iteration, the new child would replace the worst element of the existing population.

The algorithm was as follows:

```
for p := 1 to popSize
  child := RandomGenome();
  FuncEval2(child);
  Push(child, current);
endfor
for g := 1 to numGen
    for p := 1 to popSize
      mother := SelectRawTournament(current, minmax, 2, .6, TRUE);
      father := SelectRawTournament(current, minmax, 2, .6, TRUE);
      child := RRR(mother, father);
      MutateSwap(child, mutateRate);
      MutateCreep(child, mutateRate);
      FuncEval2(child);
      ReplaceRawWorst(current, child, minmax);
    endfor

    StatsPrint(g, 10, current, filename);
    if (g/10 = g/10.0)
      child := SelectRawBest (current, minmax);
      PrintGenome (child, filename);
    endif
endfor
```

Preliminary results show that the genetic algorithm is an acceptable substitute for simulated annealing. It is too early to say whether it is a better substitute, but the promise of parallelization makes it possible to say so. Optimization based on the granularity function itself, rather than membership and then granularity, is currently being investigated. The advantage of using the granularity function by itself is that it is normalized to the number of cluster centers, whereas the membership function is not. Thus, the potential is there to directly optimize the clustering and the cluster number simultaneously. The granularity function described above tends to push its cluster centers far apart because the between-cluster distance is in the denominator (see section 3.2 and figure 8). The membership function described above tends to subdivide dense areas of the data set rather than leaving them in a larger contiguous cluster group. In any case, this proposed alternative appears to work well with small numbers of cluster centers, but more intensive testing is definitely needed.

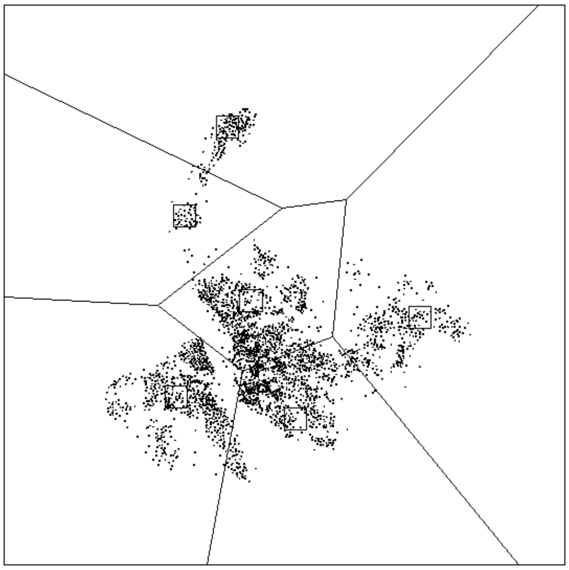

Figure 8: A better algorithm?

4 Conclusions and Further Work

The problem of membership, granularity, and optimization in cluster analysis was approached by addressing each one in turn and synthesizing the results. First, the membership function was chosen and tested with an exhaustive search. To investigate larger data sets, a switch to a simplex and simulated annealing algorithm was made. After testing for the granularity function, a genetic algorithm was implemented to deal with very large data sets. Work was also done to create the Voronoi diagrams in this paper. This involved using freely available software from the net and writing some Fortran code to make the results drawable by axx, a graphical software package.

To answer the original question, yes, the goal of creating a self-contained algorithm that searches over a given range of granularities, incorporates a preference for granularity, and outputs the optimal clustering and cluster number in a reasonable amount of time was attained. It is possible to utilize genetic algorithms for cluster analysis. And, yes, it is probably possible to maximize a genetic algorithm's potential for cluster analysis through parallelization. However, due to limited time, data, and computer resources, a robust implementation of the proposed algorithms could not be thoroughly tested and polished.

Several possibilities exist for further work. First, an empirical comparison of different optimization techniques, membership and granularity functions, and algorithms can be done. As mentioned above, there exist more computationally expensive membership and granularity functions that avoid the assumptions of similar sizes and shapes. This would thus avoid subdividing of dense areas. Another idea is to implement a penalty function that prevents such subdividing by incorporating some measure of cluster radius. If a particular arrangement of cluster centers results in two adjacent clusters where the distance between the centers is significantly less than twice the sum of the cluster radii, then there should be a penalty to force these two clusters into one. Perhaps there should be no arrangement of cluster centers where the distance between the cluster centers is less than the sum of the two relevant cluster radii. For the penalty function to work, the definition of cluster radius must be chosen carefully. Simulated annealing and genetic algorithms can be more rigorously compared. More work can also be done on achieving the goal of a self-contained algorithm that optimizes membership and granularity simultaneously.

Another interesting possibility is an application of genetic algorithms to mixture models. Most cluster analysis techniques are not statistically rigorous, but mixture models are different in that they attempt to fit distributions onto the data. Furthermore, the data used in this paper were about population in the UK (figure 9). An investigation and comparison with other computational geography methods for population clustering might prove interesting. Finally, demographics typically have much promise of applications to economics and national policy. For example, electoral districts need to be redefined over time. Businesses and governments may need to know the optimal locations for their institutions. Once the proposed method is made robust, exploring various applications may be quite useful.

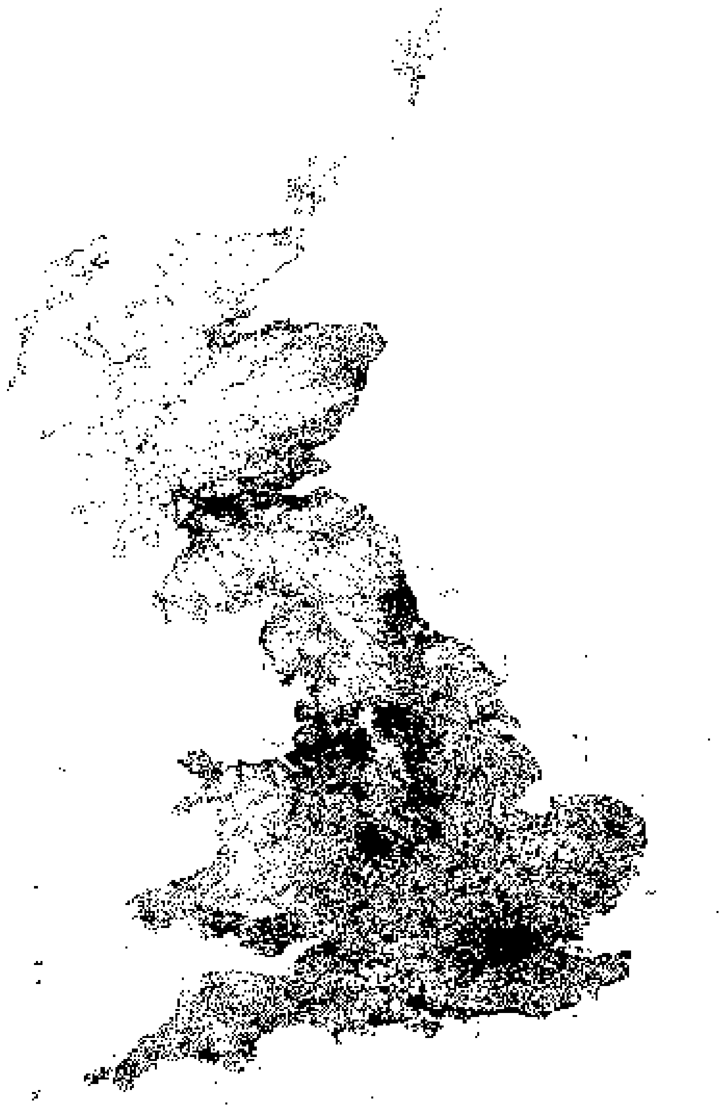

Figure 9: 50 million people, each dot represents 300-400 people. Due to data entry errors, there are some dots over water.

References

[1] Brian S. Everitt. *Cluster Analysis*. Edward Arnold, London, 1993.

[2] William H. Press, Brian P. Flannery, Saul A. Taukolsky, and William T. Vetterling. *Numerical Recipies in FORTRAN: The Art of Scientific Computing*. Cambridge University Press, Cambridge, second edition, 1992.

[3] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):1–16, 1993.

[4] Anna Hondroudakis, Joel Malard, and Gregory V. Wilson. *An Introduction to Genetic Algorithms Using RPL2*, 2 edition, May 1996.

[5] Donald E. Brown and Christopher L. Huntley. A practical application of simulated annealing to clustering. *Pattern Recognition*, 25(4):401–412, 1992.

[6] R. Srikanth, R. George, N. Warsi, D. Prabhu, F. E. Petry, and B. P. Buckles. A variable-length genetic algorithm for clustering and classifcation. *Pattern Recognition Letters*, 16:789–800, 1995.

[7] C.A. Murthy and Nirmalya Chowdhury. In search of optimal clusters using genetic algorithms. *Pattern Recognition Letters*, 17:825–832, 1996.

[8] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1:224–227, 1979.

Daniel Chen will obtain his applied mathematics bachelor's and master's degrees at the end of his fourth year, Spring 1999, at Harvard University. He plans to pursue a PhD in economics at MIT and work towards an academic position at a major research university. He hopes to make an impact on policy in the future.

This project was supervised by Dr. Harvey Richardson and Dr. David Henty of EPCC. Many thanks to both and several other EPCC staff, particularly Mario Antonioletti, for their time in dealing with my many queries. The Merseyside household and UK population data sets were obtained from the source: The 1991 Census, Crown Copyright. ESRC purchase.