CLUSTERING WITH GENETIC ALGORITHMS

This thesis is
presented to the
Department of Computer Science
for the degree of
Master of Science
of the
University of Western Australia

By
Rowena Marie Cole
January 1998

Abstract

Clustering is the search for those partitions that reflect the structure of an object set. Traditional clustering algorithms search only a small sub-set of all possible clusterings (the solution space) and consequently, there is no guarantee that the solution found will be optimal. We report here on the application of *Genetic Algorithms* (GAs) — stochastic search algorithms touted as effective search methods for large and complex spaces — to the problem of clustering. GAs which have been made applicable to the problem of clustering (by adapting the representation, fitness function, and developing suitable evolutionary operators) are known as *Genetic Clustering Algorithms* (GCAs).

There are two parts to our investigation of GCAs: first we look at clustering into a given number of clusters. The performance of GCAs on three generated data sets, analysed using 4320 differing combinations of adaptions, establishes their efficacy. Choice of adaptions and parameter settings is data set dependent, but comparison between results using generated and real data sets indicate that performance is consistent for similar data sets with the same number of objects, clusters, attributes, and a similar distribution of objects. Generally, group-number representations are better suited to the clustering problem, as are dynamic scaling, elite selection and high mutation rates. Independent generalised models fitted to the correctness and timing results for each of the generated data sets produced accurate predictions of the performance of GCAs on similar real data sets.

While GCAs can be successfully adapted to clustering, and the method produces results as accurate and correct as traditional methods, our findings indicate that, given a criterion based on simple distance metrics, GCAs provide no advantages over traditional methods.

Second, we investigate the potential of genetic algorithms for the more general clustering problem, where the number of clusters is unknown. We show that only simple modifications to the adapted GCAs are needed. We have developed a merging operator, which with elite selection, is employed to evolve an initial population with a large number of clusters toward better clusterings. With regards to accuracy and correctness, these GCAs are more successful than optimisation methods such as simulated annealing. However, such GCAs can become trapped in local minima in the same manner as traditional hierarchical methods. Such trapping is characterised by the situation where good (k-1)-clusterings do not result from our merge operator acting on good k-clusterings. A marked improvement in the algorithm is observed with the addition of a local heuristic.

Acknowledgements

I am grateful to the many people who have contributed to this thesis. I thank Dr. Nick Spadaccini for his wisdom and patience. His belief in my research and pertinent advice has been a guiding influence on the evolution of this thesis.

I was fortunate enough to participate in an exchange program with The University of British Columbia (UBC) during 1995, and I thank the Departments of Computer Science and Statistics at UBC, for the use of their facilities and their hospitality. I also thank Dr. David Lowe for his advice and support. The time spent at UBC greatly enhanced the content of this thesis.

I thank both The University of Western Australia (UWA) and UBC for allowing me the opportunity to study at UBC. I also acknowledge UWA's support of this research in the form of a HECS scholarship.

Professor Adrian Baddeley, Professor of Statistics at UWA, was kind enough to allow me the use of statistical software on the Mathematics Department's computers, without which the statistical analysis presented in the thesis would not have been possible. I also thank Dr. Katia Stefanova for her initial advice on this analysis.

I thank the members of the Computer Science Department, here at UWA, who by example (or by direct advice) have contributed ideas to the research behind this thesis. In particular, I wish to thank all who have extended extra effort to help in the production of this thesis.

Finally, I thank my friends and family for their continual support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Clustering, or grouping, is an exploratory procedure that searches for "natural" structure within a data set. This process involves sorting the data cases, or *objects*, into groups, or *clusters*, so that objects in the same cluster are more like one another than they are like objects in other clusters. Sorting occurs on the basis of *similarities* calculated from the data; no assumptions about the structure of the data are made. Clustering is useful for data reduction (reducing a large amount of data to a number of characterising sub-groups), developing classification schemes (also known as taxonomies), and suggesting or supporting hypotheses about the structure of the data.

Clustering techniques have been used in a wide range of disciplines. In psychiatry, Pilowski, Levine and Boulton [32] used clustering to develop a classification of depression. In market research, Green, Frank, and Robinson [27] used a clustering algorithm to identify homogeneous sets of test markets. In archaeology, Hodson [30] applied clustering to the problem of classifying British Hand-axes. In pattern recognition, Levrat et al. [45] used fuzzy clustering to segment an image. In engineering, Reich and Fenves [54] used a clustering algorithm to create a hierarchy of specifications and associated designs for existing bridges, and in medicine, Funk et al. [21] used clustering as a method of knowledge acquisition for expert system assisted diagnosis.

1.1 Background

This section contains a review of clustering, including a more formal definition of the clustering problem and an introduction to traditional clustering techniques.

1.1.1 The Clustering Problem

We shall define the clustering problem as follows:

> The set of n objects $X = \{X_1, X_2, ..., X_n\}$ is to be clustered. Each $X_i \in \Re^p$ is an attribute vector consisting of p real measurements describing the object. The objects are to be clustered into non-overlapping groups $C = \{C_1, C_2, ..., C_k\}$ (C is known as a *clustering*), where k is the number of clusters, $C_1 \cup C_2 \cup ... \cup C_k = X$, $C_i \neq \emptyset$, and $C_i \cap C_j = \emptyset$ for $i \neq j$. The objects within each group should be more similar to each

other than to objects in any other group, and the value of k may be unknown. If k is known, the problem is referred to as the k-clustering problem.

1.1.2 Measures of Similarity

To cluster objects according to their similarity, one must define a measure of how close two objects are, or how well their values compare. A small distance between the objects should indicate a high similarity. Thus a *distance measure* can be used to quantify *dissimilarity*.

Several distance measures are employed for clustering [35, 15]. The most commonly used is the Euclidean distance,

$$d(X_i, X_j) = \sqrt{(X_i - X_j)'(X_i - X_j)} = [\sum_{l=1}^{p}(x_{il} - x_{jl})^2]^{\frac{1}{2}},$$

the straight line distance between the two points representing the objects. An alternate measure is the "city-block" distance which sums the difference between all the attributes,

$$d(X_i, X_j) = \sum_{l=1}^{p} |x_{il} - x_{jl}|.$$

This measure is problematic if the attributes are correlated. Both of these distance measures are special cases of the more general Minkowski distance,

$$d(X_i, X_j) = [\sum_{l=1}^{p}(x_{il} - x_{jl})^m]^{\frac{1}{m}}.$$

The Mahalanobis distance is a standardised form of the Euclidean distance

$$d(X_i, X_j) = \sqrt{(X_i - X_j)'\Sigma^{-1}(X_i - X_j)},$$

where Σ is the correlation matrix. This measure scales the data in terms of standard deviations and adjusts for inter-correlations between the variables. However, it is not as commonly used as the Euclidean as prior knowledge of the clusters are required to compute Σ.

1.1.3 Traditional Clustering Algorithms

Clustering algorithms can be catergorised as either *hierarchical* or *optimisation*.

Hierarchical Techniques

Hierarchical clustering techniques proceed by either a series of successive mergers or a series of successive divisions. The result is the construction of a tree–like structure or hierarchy of clusterings which can be displayed as a diagram known as a *dendogram* (Figure 1). *Agglomerative hierarchical methods* begin with the each observation in a separate cluster. These clusters are then merged, according to their similarity (the most similar clusters are merged at each stage), until only one cluster remains.

Divisive hierarchical methods work in the opposite way. An initial cluster containing all the objects is divided into sub-groups (based on dissimilarity) until each object has its own group. Agglomerative methods are more popular than divisive methods.

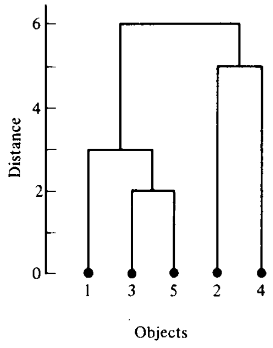


Figure 1: Dendogram for hierarchical clustering (from [35]).

For both methods, the number of clusters is needed to select a clustering from the hierarchy. However the difference between the levels of the hierarchy may be an indication of the correct number of clusters.

The following are the the steps in an agglomerative hierarchical clustering algorithm for grouping n objects. Methods differ in how the distance between clusters is calculated.

1. Begin with n clusters, each containing one object.
2. Calculate the distance between each pair of clusters. These distances are usually stored in a symmetric distance matrix, $D_{n \times n} = \{d_{ij}\}$.
3. Merge the two clusters with the minimum distance.
4. Update the distance matrix.
5. Repeat Steps 3 and 4 until a single cluster remains.

There are four important agglomerative clustering algorithms: *single-linkage*, *complete-linkage*, *average-linkage*, and *Ward's minimum variance method.* For single-linkage, or *nearest neighbour* clustering [20], the distance between two clusters is the distance between the two nearest objects in those clusters. Problems occur when the clusters are poorly delineated; this method can result in long chains with dissimilar objects at the ends (Figure 2).

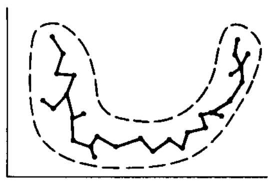


Figure 2: Chaining in single-linkage clustering (from [2]).

Complete-linkage or *furthest neighbour* clustering [59] joins the two clusters with the minimum distance between their two furthest objects, thus eliminating the chaining problem experienced with single-linkage clustering.

Average-linkage or *group-average* clustering [58] defines the distance between two clusters to be the average distance from all objects in one cluster to all objects in the other cluster. This approach tends to combine clusters with small variances, and the method is biased towards producing clusters with approximately equal variance. In a variation of this method, called *centroid clustering* [44], the distance between two clusters is defined as the distance between their centroids. This method can produce messy and confusing results, since the centroids move as clusters are combined. Thus the distance between two clusters may be less than the distance between the centroids of clusters merged at an earlier stage (Figure 3).

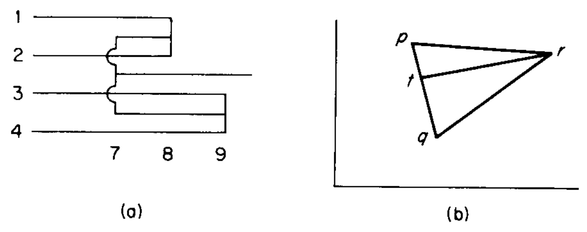


Figure 3: Reversal in centroid clustering: (a) dendogram showing reversal; (b) migration of centroids — if clusters with centroids p and q are merged, the centroid of the resulting cluster, t, is closer to r than either p or q (from [2]).

Unlike the above clustering methods, *Ward's minimum variance method* [67] optimises an objective statistic – the sum of the squared distances between each object and its cluster centre. At each step the algorithm merges the clusters that will minimise the increase of this statistic. This method tends to join clusters with small numbers of objects, and is biased towards producing clusters of approximately the same size.

Further detail on these clustering methods can be found in [2, 65, 15].

Comparison studies show that the performance of hierarchical methods vary according to the type of data — there is no one method that is best in all circumstances [53, 15]. However, Ward's minimum variance method and the average-linkage method cluster relatively accurately over a wide range of data types.

Hierarchical clustering methods suffer from several disadvantages:

(i) they are restricted to smaller data sets due to the need to store similarity matrices;

(ii) there is no provision for reallocation of objects that have been incorrectly grouped at an early stage; and

(iii) the results reflect the degree to which the data conforms to the structural forms embedded in the algorithm.

Optimisation Techniques

Unlike hierarchical techniques, which produce a series of related clusterings, optimisation techniques produce a single clustering which optimises a pre-defined criterion or objective function. The number of clusters in this clustering, is either specified *a priori* or is determined as part of the clustering method.

Optimisation methods start with an initial partition of objects into a specified number of groups. Objects are then reassigned to clusters according to the objective function until some terminating criterion is met. These methods differ with respect to the starting partitions, the objective functions, the reassignment processes, and the terminating criteria.

Optimisation methods may use a random initial partition or one generated from seed points [2]. In the latter these seed points may be random, or may be selected using a method that attempts to ensure that they span the data. There are several methods for building partitions from seed points.

Objective functions which are commonly used as clustering criteria [15] include:

- Minimisation of trace (W)
- Minimisation of the determinant of (W)
- Maximisation of trace (BW^{-1})

where:

$$W = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(X_{ij} - \overline{X}_j)(X_{ij} - \overline{X}_i)'$$

is the pooled within-cluster covariance matrix, and

$$B = \sum_{i=1}^{k} n_i(\overline{X}_i - \overline{X})(\overline{X}_i - \overline{X})'$$

is the between cluster covariance matrix. Here n_i is the number of objects in cluster i, X_{ij} is the jth object of the ith cluster, $\overline{X}_i = \frac{1}{n_i}\sum_{l=1}^{n_i} X_{il}$ is the centroid of cluster i, and $\overline{X} = \frac{1}{n}\sum_{l=1}^{n} X_l$ is the grand mean.

The minimisation of trace (W) is equivalent to minimising the sum of square Euclidean distances between individuals and their cluster centroids (Ward's minimum variance method uses this statistic to determine which clusters to merge at each step). This clustering criterion favours spherical clusters, since the correlation between the attributes is not considered. The second criterion is scale invariant, and is suitable for clusters which do not have a spherical distribution. The third criterion, trace (BW^{-1}), is a generalisation of the Mahalanobis distance to more than two groups. For a more detailed review of these and other criterion refer to [26, 15].

Two types of reassignment are generally employed. The first loops through all the objects, reassigning each to the cluster whose centroid is the closest. The second searches a local *neighbourhood* of clusterings for one which improves the objective function value. Although the majority of optimisation methods require the number of clusters *a priori*, some reassignment processes have been designed to allow the number of clusters to evolve during clustering [2].

1. Start with an arbitrary partition into k initial clusters.
2. Take each object in sequence and move to the cluster which reduces the numerical criterion the most. Transfer the object, compute new centroids.
3. Repeat Step 2 until a full cycle through the objects cannot reduce the criterion value.

Figure 4: The k-means algorithm.

Optimisation algorithms terminate when there are no reassignments that will reduce the criterion value. This occurs when all objects are in the cluster whose centroid is closest to them, or the current clustering is a local minima.

Two popular optimisation methods are k-means and hill-climbing. Algorithms for these methods are given in Figures 4 and 5. Further discussion of optimisation methods can be found in [2, 29, 53, 15]. A more recent approach to the problem, using simulated annealing, is introduced in [41].

The k-means technique performs well in comparison to hill-climbing and hierarchical methods, although it is sensitive to its initial partition [53]. The k-means method is also less affected by outliers, the choice of distance measure, and the presence of irrelevant attributes or dimensions.

Unlike hierarchical clustering techniques, optimisation methods do not store similarity matrices. Thus the size of the data is not limited by storage space. However, there are a number of disadvantages affecting optimisation methods:

(i) some methods require the number of clusters *a priori*, and will divide the data into this number of clusters regardless of the data structure;

(ii) certain clustering criterion are biased toward particular cluster shapes, and will impose these shapes on the data; and

(iii) the performance of optimisation techniques is highly dependent on the initial partition.

1. Start with an arbitrary partition into k initial clusters.
2. Search the local neighbourhood for the clustering which which reduces the numerical criterion the most.
3. Repeat Step 2 until no clusterings in the neighbourhood reduce the clustering criterion.

Figure 5: The hill-climbing algorithm.

1.2 Motivation

The number of ways of sorting n objects into k groups is given by Liu [46]:

$$N(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i \begin{pmatrix} k \\ i \end{pmatrix} (k-i)^n$$

For example, there are $N(25,5) = 2,436,684,974,110,751$ ways of sorting 25 objects into five groups [2]. If the number of clusters is unknown the objects can be sorted $\sum_{i=1}^{n} N(n,k)$ ways. For our 25 objects this is over 4×10^{18} clusterings. Clearly, it is impractical for an algorithm to exhaustively search the solution space to find the optimal solution.

Furthermore, traditional clustering algorithms search a relatively small subset of the solution space (these subsets are defined by the number of clusters, the clustering criteria, and the clustering method). Consequently, the probability of success of these methods is small. Algorithms such as single-linkage are deterministic and will repeatedly find the same solution for a given data set, whereas algorithms such as k-means conduct a local search starting from an initial partition. In each case, the solution may be a *local* optimum, which is not necessarily the *global* solution. This is exacerbated when the solution space is very large.

Clearly, we need an algorithm with the potential to search large solution spaces effectively. Recently, *genetic algorithms* have been widely employed for optimisation problems in several domains. Their success lies in their ability to span a large subset of the search space. The application of genetic algorithms to clustering is the focus of this work. Below we introduce the fundamentals of genetic algorithms.

1.3 Genetic Algorithms

A Genetic Algorithm (GA) is a computational abstraction of biological evolution that can be used to solve some optimisation problems [31, 24]. In its simplest form, a GA is an iterative process applying a series of genetic operators such as selection, crossover and mutation to a population of elements (Figure 6). These elements, called *chromosomes*, or *individuals* represent possible solutions to the problem; the initial chromosomes are selected randomly from the solution space. Genetic operators combine the genetic information of the elements to form new *generations* of the population; this process is known as *reproduction*. Each chromosome has an associated fitness value which quantifies its value as a solution to the problem — a chromosome representing a better solution will have a higher fitness value. The chromosomes compete to reproduce based on their fitness values, thus the chromosomes representing better solutions have a higher chance of survival.

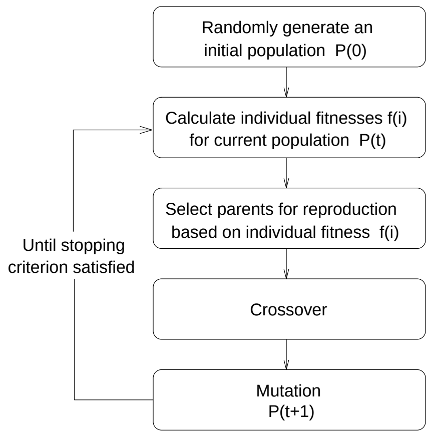

Randomly generate an initial population P(0)
Calculate individual fitnesses f(i) for current population P(t)
Select parents for reproduction based on individual fitness f(i)
Crossover
Mutation P(t+1)
Until stopping criterion satisfied

Figure 6: Outline of a simple GA.

Selection according to fitness combined with crossover gives the GA its evolutionary power. The underlying assumption is that the recombination of short sequences of genetic material, or *building blocks*, from fit parents will lead to children of higher fitness. This is known as the *building block hypothesis*, and violation of this assumption may lead to poor performance [24].

GAs have been used to solve a variety of optimisation problems. Goldberg [24] reviews the application of GAs to problems including natural gas pipeline control, structural optimisation, and image registration. Michalewicz [48] discusses the application of evolutionary techniques to problems such as job scheduling, path planning, and the travelling salesman problem.

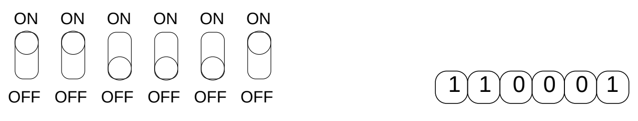
To successfully apply a GA to solve a problem one needs to determine the following:

1. how to represent possible solutions, or the chromosomal encoding;
2. what to use as the fitness function which accurately represents the value of the solution;
3. which genetic operators to employ; and
4. the parameter values (population size, probability of applying operators, etc.) which are suitable.

1.3.1 Representation

The representation should be *complete*, that is one should be able to encode all possible solutions to the problem. Clearly, if the GA cannot represent the solution, it can never find it. A secondary consideration is *validity*, that is all possible encodings should correspond to points inside the solution space. Invalid representations can be used, but it may be necessary to adapt the GA to avoid invalid encodings. The representation may also have a number of different chromosomes that represent the same solution, this is known as *redundancy*. Since a GA is a search over its representation space, not the solution space, high redundancy may present problems for convergence [36].

As a simple example of an encoding scheme let us consider a bank of six input switches. We can create a code by using a string of six 1's and 0's where each switch is represented by a 1 if the switch is on and a 0 if the switch is off (Figure 7). This representation is complete, valid and has no redundancy.



(a) (b)

Figure 7: Encoding scheme for six on-off switches: (a) the six switches; (b) a representation of the given switch positions.

Goldberg [24] offers two basic principles for choosing a GA coding. First, the coding should contain meaningful building blocks (related information should be contained within a short substring). In our example the position of each switch is contained in a single bit which is the smallest sub-string possible. Secondly, the alphabet should be the smallest that permits a natural expression of the problem. The binary encoding we have selected is a natural representation for the switching problem.

In accordance with the above principles, the majority of traditional GAs have used the binary alphabet. Antonisse [3] argues that larger encoding alphabets are suitable for GAs, and empirical studies [34, 48] show that GAs with floating point representations are faster and more precise than binary encodings on continuous domains. Davis [13, 14] and Michalewicz [48] further emphasise the need for an encoding scheme that gives a natural representation of the problem irrespective of the alphabet size. For example, a permutation is perhaps the most natural representation of a tour for the travelling salesman problem (Figure 8).

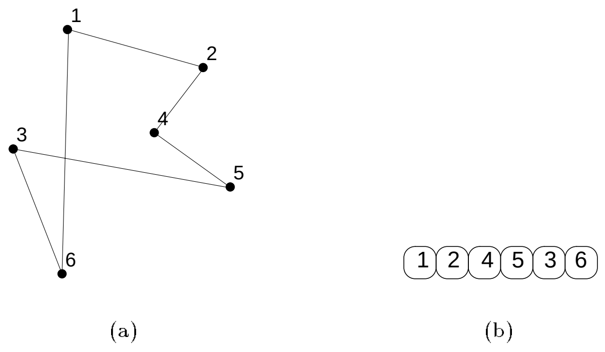


Figure 8: Encoding scheme for the travelling salesman problem: (a) a tour of six cities; (b) a representation of the given tour.

1.3.2 Fitness Function

The fitness function quantifies the suitability of each chromosome as a solution and is used as a basis for selecting chromosomes for reproduction. Chromosomes with high fitness have more chance of being selected, and thus, passing their genetic material (recombined via crossover) to the next generation. The fitness function provides the pressure for the GA to evolve its population toward chromosomes of higher fitness, and clearly, the success of the GA for the problem is dependent on the choice of the fitness function.

The chromosome representing the optimal solution should have the maximum fitness value for the solution space; near optimal solutions should have high fitness values. Since GAs tend to retain genetic material found in chromosomes of high fitness, appropriate choice of fitness function will increase the probability of retaining genetic material associated with optimal or near optimal solutions.

Consider the six switches from Figure 7. The switches belong to a black box optimisation problem that has an associated payoff measure. A selection of payoff values are given in Table 1. An appropriate fitness function to find the switch settings with the highest payoff would be the payoff values themselves. However, if we wanted to find the switch settings that would give the payoff closest to \$20, we would use a fitness function based around the difference between the payoff and \$20 (for example, $f = -|20 - \text{Payoff}|$).

Switch						Payoff
1	2	3	4	5	6	$
OFF	OFF	OFF	OFF	OFF	OFF	0.9
OFF	ON	OFF	OFF	ON	ON	15.1
OFF	ON	ON	ON	ON	OFF	1.7
ON	OFF	OFF	OFF	OFF	OFF	21.0
ON	ON	OFF	OFF	OFF	ON	39.4
ON	ON	ON	ON	ON	ON	23.3

Table 1: Some payoff values for the black box optimisation problem with six switches.

A further issue when considering the fitness function is *scaling*. When the population fitness values are diverse, the fitter individuals will have high selection probabilities compared to average individuals. However, as the fitness values converge the fitter individuals will have similar probabilities to the average individuals. Thus the probability of selecting the fitter individuals (the driving force behind the GA's evolution) can be significantly reduced.

Selection pressure can be maintained by scaling the fitness values. *Linear scaling* calculates the scaled fitness f' from the raw fitness f using a linear transformation, $f' = af + b$ [24]. The constants a and b are chosen so that the average scaled fitness is equal to the average raw fitness, and the maximum scaled fitness equals the maximum raw fitness multiplied by the number of expected copies desired for the best population member. Linear scaling may produce negative scaling values, although Goldberg [24] provides an algorithm to avoid this. *Power law scaling* calculates the scaled fitness as some specified power of the raw fitness, $f' = f^c$ [24]. The value of c is problem dependent.

1.3.3 Genetic Operators

There are four major operators used for GAs: initialisation, selection, crossover and mutation.

The Initialisation Operator

This operator is used to generate the initial population for the GA. The initial population should contain chromosomes which are scattered throughout the solution space, thus providing the GA with a variety of genetic material. The easiest way to achieve this is to randomly select chromosomes from the representation space. For our switches example (Section 1.3.1) we could generate a population of random binary strings of length six. Alternatively, a population of random permutations of the integers 1 to 6 is appropriate for our travelling salesman representation.

The Selection Operator

Selection chooses individuals for reproduction based on their fitness values. *Fitness proportional selection* or *roulette wheel selection* [24] chooses individuals with a probability proportional to their relative fitness. This corresponds to a spinning wheel where each chromosome has been allocated a section of the wheel according to its relative fitness value — the higher the fitness the larger the allocated section. An alternate selection scheme is *remainder stochastic sampling* [9]. Here, the number of expected occurrences of each individual in the next population are calculated. Parents are selected according to the integer part of this number, and the chromosomes then compete according the the fractional parts for the remaining places in the population. *Stochastic universal sampling* [4] selects all the parents at the same time. This method is analogous to a single spinning wheel with the number of (equally spaced) pointers representing the desired number of parents. *Tournament selection* [9] is a process which selects parents according to their rank. The identity of a parent is decided by randomly selecting a predetermined number of individuals, and then finding the fittest individual within this set.

A simple GA will select a sufficient number of parents to reproduce and form the next generation. However, *steady state* GAs will select a single pair of parents to reproduce and then add the resulting offspring back into the population (perhaps deleting the parents to make room for them) before selecting another pair to reproduce. *Elite* selection strategies [37] copy the fittest individuals straight from one generation to the next to prevent them from being lost during reproduction. Further discussion of these and other selection mechanisms can be found in [24, 48].

The Crossover Operator

Crossover combines the genetic material of one or more parents to produce one or more offspring. *Single-point crossover* exchanges the tails of two parent strings. The size of the strings exchanged is determined by a randomly generated *crossover point* (Figure 9a). *Uniform crossover* exchanges each bit with a random probability (Figure 9b).

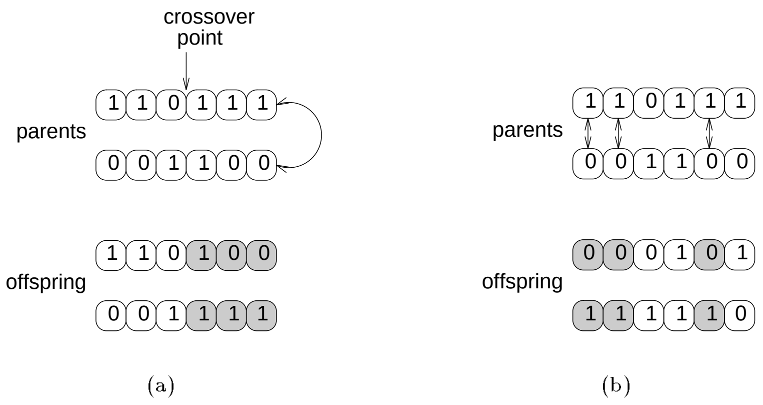


Figure 9: Examples of single-point and uniform crossover: (a) single-point crossover exchanges the tails of two parents; (b) uniform crossover randomly swaps bits between parents.

There are two main problems to consider when deciding which crossover operator to use. Firstly, some crossover operators give invalid solutions when applied to certain representations. For example, if we applied single-point or uniform crossover to the permutation representation we used for the travelling salesman problem (Section 1.3.1), we may produce a tour which visits some cities more than once, and skips one or more cities.

There are two ways to deal with this problem: (1) we can check each offspring after crossover and try again if the offspring is invalid; or (2) we select (or modify/design) the crossover operator so that it always produces valid offspring for that representation.

Numerous crossovers have been designed for permutation representations. *Partially mapped crossover* or PMX [23] chooses a substring of one parent and then preserves the order and position of as many values as possible from the other parent (Figure 10a). PMX starts by swapping a randomly selected substring between the two parents. This substring defines a series of mappings which are applied to produce the offspring.

Order-based crossover or OX [12] builds offspring by choosing a substring from one parent and preserving the relative order of values from the other parent (Figure 10b). OX starts by copying a (randomly selected) substring of the second parent into the first child. Then starting at the end of the substring, the sequence of the remaining values in the first parent is copied into the child.

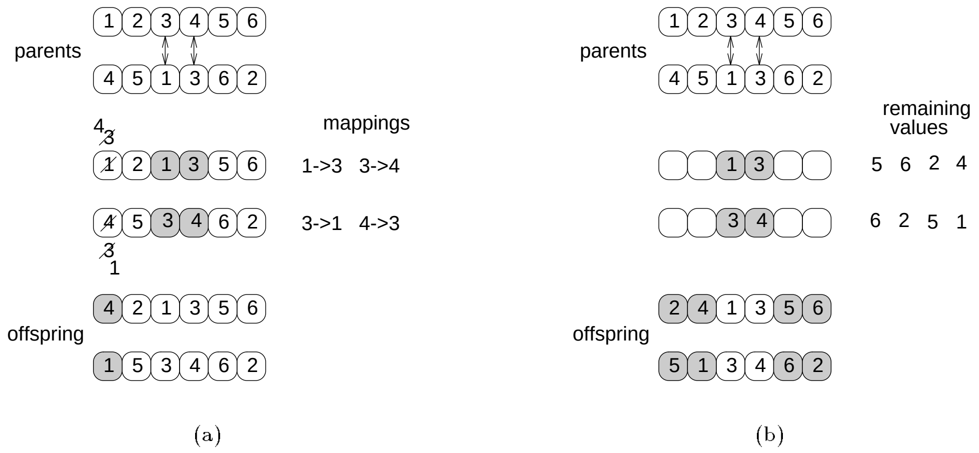


Figure 10: Crossover operators for permutation representations: (a) PMX preserves the order and position of values in the parents; (b) OX preserves the order of values within the parents.

Michalewicz [48] describes suitable crossover operators for a variety of encoding schemes.

The second problem to consider when selecting a crossover operator is one of *context insensitivity*. This occurs when a chromosome is similar to its parent chromosomes, but the solution it represents is not similar to the solutions represented by the parent chromosomes — the crossover operator is insensitive to the solutions that the parents represent. We shall discuss this further in Chapter 2.

The Mutation Operator

The mutation operator introduces new genetic material into the population. For a binary string, a simple mutation would change a 0 to a 1 (or a 1 to a 0) somewhere along the string. For a permutation, we might swap two integer values. For real-valued chromosomes, a mutation operator could replace values by using a Gaussian distribution with a mean equal to the current real value.

As with the crossover operator, it is possible to create invalid chromosomes by applying some mutation operators to certain representations (for example, applying a binary mutation to a permutation encoding), so we need to ensure that the mutation operator is suitable for the representation.

1.3.4 Parameter Values

GA performance is sensitive to certain parameter values, in particular the population size, frequency of operator application, and the termination criterion. Small populations may converge quickly to sub-optimal solutions, while large populations waste computational resources. Low crossover rates slow down the convergence because of the increased time required to explore the existing genetic material. If the mutation rate is too high, the relationship between generations may be too low, possibly decaying into random search. If the rate is too low, unseen and novel traits will appear infrequently.

There are two common termination criteria. The first allows the GA a set number of generations which may not be enough to produce a satisfactory solution. The second requires the convergence of the population. If complete convergence is necessary for termination the GA will only stop when all members of the population are the same chromosome. This may never occur. An alternative is to stop the GA when one chromosome occupies a certain percentage of the population.

A number of studies have been conducted to determine appropriate parameter values. De Jong [37] calculated parameter values from theory, using a suite of five functions to test the values. Grefenstette [28] used a GA to evolve parameter values — a meta-GA with a population of GAs with varying parameter values. Schaffer et al. [56] ran exhaustive tests for a wide range of values. More recently, numerous theoretical studies model GAs as Markov chains in order to determine the optimal parameter values [25, 52, 62] However, all of these studies consider only binary encoding schemes, and only small problems are used in the empirical studies. An alternate approach is to allow the operator probabilities to adapt during the GA's evolution [13, 61].

1.3.5 Final Comments — GAs

GAs have gained popularity for solving optimisation problems. However, De Jong [38] emphasises that GAs are not function optimisers, but can be adapted to work as such. Davis [13, 14] and Michalewicz [48] go even further, stressing that a GA must be adapted to suit the problem — in particular, the representation and operators need to be be designed carefully. In fact Davis [13] states:

> ... a great many real-world optimization problems appear ripe for solution by genetic algorithms, yet the binary representation appears ineffective or inefficient for them. Further, operators other than binary crossover and binary mutation appear to contribute to good performance in those domains. ... [A further problem] then, is that of parameterizing a genetic algorithm that differs from the type studied so thoroughly by researchers in the field.

1.4 Overview

This thesis is an investigation of using genetic algorithms to cluster. In particular, we:

(i) determine the adaptions necessary to enable the GA to cluster successfully;

(ii) find suitable parameter values for k-clustering (these may vary for different data sets);

(iii) compare the performance of genetic clustering with traditional k-clustering algorithms; and

(iv) explore the potential of GAs for the general clustering problem.

The layout of this thesis is as follows: Chapter 2 discusses the process of adapting GAs for the clustering problem when the number of clusters is known, Chapter 3 contains a comparison of genetic clustering with other clustering methods, and Chapter 4 examines the issue of genetic clustering when the number of clusters is unknown. A final discussion is presented in Chapter 5.

Chapter 2

Adapting GAs for k-Clustering

A number of authors have applied GAs to the problem of k-clustering, where the required number of clusters is known [43, 1, 8, 6]. Various adaptions are used to enable the GAs to cluster and to enhance their performance, but there is little or no comparison between the different adaptions. Further, the clustering GAs or *genetic clustering algorithms* (GCAs) are tested on small data sets, or heuristics are added to enable the GAs to cope with a larger number of objects. It is not clear which adaptions are best suited to the clustering problem, or how any adaptions will affect GA performance for differing data sets.

In this chapter we shall compare a number of adaptions appropriate for the k-clustering problem, including some used for more general grouping problems. The aim is to determine which adaptions will enable GAs to find the correct clusterings in the fastest time. We shall also ascertain whether these adaptions should vary with the data set.

2.1 Background

Adaptions for the k-clustering problem fall into the following areas: representation, fitness function, operators, and parameter values.

2.1.1 Representation

Genetic representations for clustering or grouping problems are based on two underlying schemes. The first allocates each object one (or more) integers or bits, known as *genes*, and uses the value of these genes to signify which cluster the object belongs to. The second scheme represents the objects with gene values, and the position of these genes signifies how the objects are divided amongst the clusters. Representations using these schemes differ in how the genes are assigned and how the gene values are interpreted.

Figure 11 contains encodings of the clustering $\{\{X_1, X_3, X_6\}, \{X_2, X_4, X_5\}\}$ for a number of representations that we will discuss in detail shortly. The two clusters are denoted as 1 and 2 respectively, and the six objects are denoted by the numbers 1 to 6.

Group-number encoding [36] is based on the first encoding scheme and represents a clustering of n objects as a string of n integers where the ith integer signifies the group number of the ith

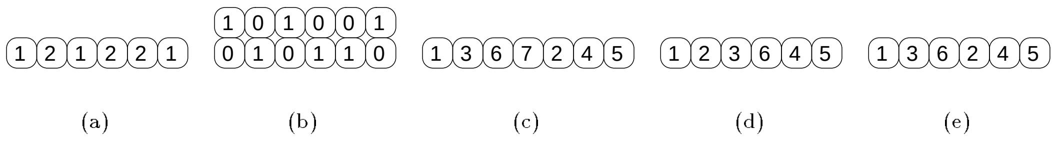


Figure 11: Chromosomes representing the clustering $\{\{X_1, X_3, X_6\}, \{X_2, X_4, X_5\}\}$ for various encoding schemes: (a) group-number; (b) matrix; (c) permutation with the separator character 7; (d) greedy permutation^1; (e) order-based^1. ^{1} assuming the correct clustering is the local minimum for this chromosome.

object. When there are only two clusters this can be reduced to a binary encoding scheme by using 0 and 1 as the group identifiers [43].

Bezdek et al. [6] use an $n \times k$ matrix to represent a clustering, with each row corresponding to a cluster and each column associated with an object. A 1 in row i, column j means that object j is in group i. Each column contains exactly one 1, whereas a row can have many 1's. All other elements are 0's. This representation can be adapted for overlapping clusters or fuzzy clustering [5].

Encoding schemes that signify objects by gene values use permutations of the object numbers to represent clusterings. *Permutation with separators* encoding [36] uses the integers $n+1$ to $n+k-1$ (or other appropriate separators) to indicate where the cluster boundaries are in the permutation.

There are also permutation representations that need a local search to determine which clustering they correspond to — these are known as *greedy* representations. *Greedy permutation* encoding [36] uses the first k objects in the permutation to seed k clusters. The remaining objects are then, in the order they appear in the permutation, added to the cluster which yields the best objective function value (typically the cluster with the closest centroid).

Bhuyan, Raghavan, and Elayavalli [8] also use a greedy encoding scheme. Here a permutation represents all possible clusterings with the correct number of clusters and the objects in that order. An algorithm by Fisher [19, 29] is used to find which of these clusterings gives the best objective function value (Figure 12). This algorithm will find the optimal k-clustering for the permutation, but the clustering is not necessarily unique. This is called *order-based* encoding.

As we saw in Chapter 1, when selecting a genetic representation we need to consider: completeness, validity, and redundancy. An incomplete representation cannot encode all possible solutions as chromosomes, and any GA using such a representation will be searching a proper subset of the solution space. All of the above representations are complete except for the two greedy encoding schemes. Both of these decode clusterings using local search, so only clusterings that are local minima (or maxima) are represented. This can be a problem if the optimal clustering is not a minima (or maxima) of the local search.

For the k-clustering problem, any chromosome that does not represent a clustering with k groups is necessarily invalid: a group number chromosome that does not include all group numbers as gene values is invalid; a matrix encoding with a row of 0's is invalid; a permutation with separators chromosome with adjacent separators, or a separator as the first or last gene, is invalid. A matrix encoding is also invalid if there is more than one 1 in any column. Chromosomes with group values that do not correspond to a group or object, and permutations with repeated or missing object identifiers are invalid.

1. Compute the diameter $D(I,J)$ for the cluster $(I, I+1, ..., J)$, for all I, J such that $1 \leq I < J \leq M$. $D(I,J) = \sum\{I \leq L \leq J\}[X(L) - \overline{X}]^2$ and $\overline{X} = \sum\{I \leq L \leq J\}\frac{X(L)}{J-I+1}$
2. Compute the errors of the optimal partitions, $2 \leq I \leq M$, by $e[P(I,2)] = \min[D(1, J-1) + D(J, I)]$ over the range $2 \leq J \leq I$.
3. For each $L(3 \leq L \leq K)$ compute the errors of the optimal partitions $e[P(I,L)](L \leq I \leq M)$ by
$$e[P(I,L)] = \min\{e[P(J-1, L-1)] + D(J,I)\}$$
over the range $L \leq J \leq I$.
4. The optimal partition $P(M,K)$ is discovered from the table of errors $e[P(I,L)](1 \leq L \leq K, 1 \leq I \leq M)$ by first finding J so that
$$e[P(M,K)] = \min\{e[P(J-1, K-1)] + D(J,M)\}.$$
The last cluster is then $(J, J+1, ..., M)$. Now find J^* so that $e[P(J-1,K)] = e[P(J-1, K-1)] + D(J^*, J-1)$. The second-to-last cluster of $P(M,K)$ is $(J^*, J^*+1, ..., J-1)$, and so on.

Figure 12: Fisher's Algorithm for finding the optimal k-clustering for a permutation (from [29]).

All of the above encoding schemes have some level of redundancy (more than one chromosome represents a clustering). We can swap the group numbers (or rows) $k!$ ways, and the redundancy of permutation encoding grows exponentially with the number of objects [17].

A final consideration when selecting a representation is the complexity of the local search for the greedy representations. The local search for greedy permutation is $O(nk)$, while the order of Fisher's algorithm for order-based encoding is $O(n^2k)$.

2.1.2 Fitness Function

Objective functions used for traditional clustering algorithms (see Section 1.1.3) can act as fitness functions for GCAs. However, if the optimal clustering corresponds to the minimal objective function value, we will need to transform the objective function value since GAs work to maximise their fitness values. In addition, fitness values in a GA need to be positive if we are using fitness proportional selection.

Krovi [43] uses the ratio of the between sum of squares and within sum of squares as his fitness function. Since the aim is to maximise this value, no transformation is necessary. Bhuyan et al. [8] and Bhuyan [7] use the sum of squared Euclidean distance of each object from the centroid of its cluster. This value is then transformed ($f' = C_{max} - f$, where f is the raw fitness, f' is the scaled fitness, and C_{max} is the value of the poorest string in the population — this is known as a *local* transformation) and linearly scaled (see Section 1.3.2) to get the fitness value. Alippi and Cucchiara [1] also use the same criterion, but use a GA that has been adapted to minimise fitness values. Bezdek et al.'s [6] clustering criterion is also based around minimising the sum of squared distances of objects from their cluster centres, but they use three different distance metrics (Euclidean, diagonal, and Mahalonobis) to allow for different cluster shapes. Bezdek et al. have adapted their selection operator to avoid the need to transform the criterion values (see

Section 2.1.3).

2.1.3 Genetic Operators

The operators pass genetic information between subsequent generations of the population. As a result, operators need to be matched with or designed for the representation, so that the offspring are valid and inherit characteristics from their parents. Operators used for genetic clustering or grouping include some of the selection, crossover and mutation methods we looked at in the previous chapter, some adaptions of these, and some totally novel operators.

Initialisation

The initial population of a GCA should provide a wide variety of genetic material, necessary for a thorough search of the problem space. Care also needs to be taken to ensure that the population contains only valid chromosomes unless the GCA can handle invalid ones. Randomly generating the initial population is one method of selecting a spread of genetic material. Heuristic initialisation operators can be used to select fit chromosomes in an effort to reduce the time required for the GCA to converge on a solution.

- **Group-number**
  An initial group-number population can be created from chromosomes with each object's group number a random number between 1 and k inclusive (Krovi [43] uses 0 or 1). This method may produce invalid chromosomes (with less than k groups) so Jones and Beltramo [36] check to see that all groups are included. If not, the chromosome is rejected.

- **Matrix**
  Bezdek, et al. [6] use a partially random initialisation process. k cluster centres are produced by selecting feature values from random objects (the jth attribute of the ith centre is the jth attribute of a randomly selected object). A matrix chromosome is then created from the cluster centres — the exact method is not explained in detail. Alippi and Cucchiara [1] do not elaborate on their initialisation process.

- **Permutation with separators**
  Jones and Beltramo [36] use a three step process to produce valid chromosomes for this representation:

  1. Generate a string of random group numbers and divide the objects into groups accordingly. Reject the chromosome if there is an empty group.
  2. Randomly permute the order of the objects in each group.
  3. Create the chromosome by listing the objects in their permuted order and adding the separators (in random order).

- **Greedy permutation**
  Jones and Beltramo [36] use random permutations as the initial population for this representation.

- **Order-based**
  Bhuyan et al. [8] compare three different initial population constructors, namely A, B, and C. The first constructor, A, randomly places the objects into a random list. The second, constructor B, chooses an initial object randomly, and then selects (from the remaining objects) the object that is closest to the last selected object. This is repeated until all the objects have been selected. The complexity of this constructor is $O(n^2)$. Constructor C is the same as constructor B except that instead of searching all remaining objects to find the closest one, only c (a constant defined by the user) are searched. Bhuyan [7] uses the same constructor C.

Selection

Chromosomes are selected for reproduction based on their relative fitness. Thus the representation is not a factor when choosing an appropriate selection operator, but the fitness function is. If all fitness values are positive, and the maximum fitness value corresponds to the optimal clustering, then fitness proportional selection may be appropriate. Otherwise, a ranking selection method may be used. In addition, elite selection will ensure that the fittest chromosomes are passed from one generation to the next.

Krovi [43] uses the fitness proportional selection provided in Goldberg's book [24]. The selection operator used by both Bhuyan et al. [8] and Bhuyan [7] is an elite version of fitness proportional selection. A new population is formed by picking the x (a parameter provided by the user) best strings from the combination of the old population and the offspring. The remaining chromosomes in the new population are selected from the offspring.

Jones and Beltramo [36] and Bezdek et al. [6] use ranking selection methods. Jones and Beltramo use a steady-state GCA where only two parents are selected for reproduction during each iteration. The selection probability depends linearly on rank — the best population member is selected with probability b/P and the worst member with probability $(2 - b)/P$. Here b is the bias and P is the population size. The parents are crossed to form a single child which is inserted into the population and the worst population member is deleted so the population size remains constant.

The selection operator used by Bezdek et al. works as follows. First, the population is sorted by fitness value and a set of R chromosomes with the lowest values are chosen to reproduce. The reproduction pairs are randomly chosen from this set. After reproduction the R child matrices are added to the population and those with the greatest fitness values are dropped. Bezdek et al. actually state that $P - R$ matrices are dropped but this would mean that the population size would change unless R is half the value of P (which it is in all their examples). If instead R matrices are dropped the population size will remain constant for all values of P and R. Since Bezdek et al. use a ranking selection method, the fitness function values do not need to be transformed. Hence the lower fitness values correspond to better solutions in this case, which is why the chromosomes with low fitness values are selected for reproduction.

Alippi and Cucchiara [1] do not report their selection method.

Crossover

The crossover operator is designed to transfer genetic material from one generation to the next. The major concerns with this operator are validity and context insensitivity. It may be necessary to check that offspring produced by a certain operator are valid, and reject any invalid chromosomes.

Context insensitivity occurs when the crossover operator used in a redundant representation acts on the chromosomal level instead of the clustering level. In this case the child chromosome may resemble the parent chromosomes, but the child clustering does not resemble the parent clusterings — the operator is insensitive to the context of the chromosomes. Figure 13 shows

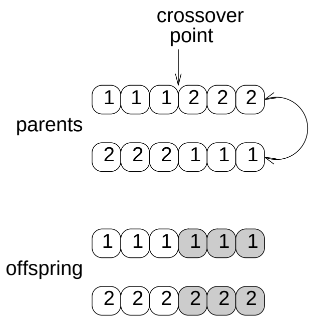


Figure 13: Context insensitivity of single-point crossover.

that single-point crossover is context insensitive for the group-number representation. Here both parents represent the same clustering, $\{\{X_1, X_2, X_3\}, \{X_4, X_5, X_6\}\}$ although the group numbers are different. Given that the parents represent the same solution, we would expect the children to also represent this solution. Instead both children are the clustering $\{\{X_1, X_2, X_3, X_4, X_5, X_6\}\}$ which does not resemble either parents (it is also invalid, but that is a separate issue).

Falkenauer [18] demonstrates the context insensitivity of the PMX operator for permutation encodings of grouping problems.

- **Group-number**

  Single-point and uniform crossover can be used for group-number chromosomes. However both operators may produce invalid chromosomes and are context insensitive.

  Krovi [43] uses single-point crossover as implemented by Goldberg [24].

  Jones and Beltramo [36] compare three types of cross-over for the group-number representation: single-point, uniform, and edge-based. They use two versions of the former operators: one with rejection, and one with rejection and renumbering. Rejection is introduced to counter the problem of validity — a child is only accepted if it is valid. Renumbering is an attempt to cope with context insensitivity — the parents are canonically renumbered before crossover. For example the first parent in Figure 13 would be renumbered to match the second.

  For single-point crossover all possible children are generated and then one valid child is selected from these. Uniform crossover is repeated until the child has k groups or a limit on the number of attempted crosses is reached (if this occurs the child is set to one of the parents at random).

The edge-based crossover operator constructs a child chromosome by combining the edges of the parent chromosomes (two objects are considered to be connected by an edge if they are in the same group). This proceeds as follows:

1. Initialise the child to the set of non-empty intersections of the clusters of the two parents. Let L denote the number of non-empty intersections.
2. If $L = K$, stop. Otherwise go to step 3.
3. Select the pair of groups with the minimum number of non-inherited edges (between-group edges not present in either parent), breaking ties at random. Join this pair of groups, set $L = L - 1$, and go to step 2.

Since edge-based crossover manipulates the parent clusterings rather than their chromosomes, this operator is context sensitive for *all* representations. Further, if both parents are valid, edge-based crossover will always produce a valid child. However, this crossover has $O(k^4)$ complexity.

For example, consider the following parent chromosomes:

$$\boxed{1}\boxed{3}\boxed{2}\boxed{2}\boxed{2}\boxed{3}$$

$$\boxed{3}\boxed{2}\boxed{1}\boxed{2}\boxed{3}\boxed{2}$$

These parents encode the clusterings:

$$\{\{X_1\}, \{X_3, X_4, X_5\}, \{X_2, X_6\}\}$$
$$\{\{X_3\}, \{X_2, X_4, X_6\}, \{X_1, X_5\}\},$$

and the non-empty intersections of these clusterings are:

$$\{\{X_1\}, \{X_3\}, \{X_4\}, \{X_5\}, \{X_2, X_6\}\}$$

We initialise the child to the set of intersections and then merge clusters until the correct number of clusters is reached.

One possible child is:

$$\boxed{1}\boxed{3}\boxed{2}\boxed{2}\boxed{1}\boxed{3}$$

which inherits $\{X_3, X_4\}$ from parent 1, $\{X_1, X_5\}$ from parent 2, and $\{X_2, X_6\}$ from both parents.

Von Laszewski [66] and Mühlenbein [50] also describe crossover operators (for the graph partitioning problem) that work with partitionings rather than chromosomes. Both copy a random group from one parent to the other and then rearrange the result to form a valid partitioning (Von Laszewski and Mühlenbein have the constraint of equal group sizes). In order to deal with the problem of context insensitivity, Mühlenbein [50] rearranges the group numbers in the second parent so that the group number of the partition being copied is the same as the most similar partition in the second parent.

- **Matrix**

  Alippi and Cucchiara [1] use a single-point asexual crossover to avoid the problem of

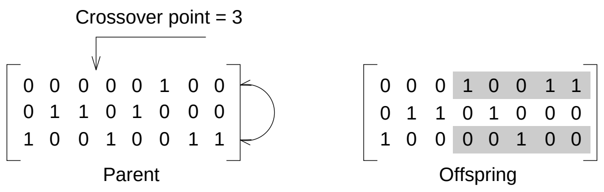


Figure 14: Alippi and Cucchiara's asexual crossover (adapted from [1]).

redundancy (Figure 14). The tails of two rows of the matrix are swapped, starting from a randomly selected crossover point. Clusterings with less than k groups may be produced by this operator.

Bezdek et al. [6] use a sexual 2-point crossover (Figure 15). A crossover point and a distance (the number of columns to be swapped) are randomly selected — these determine which columns are swapped between the parents. This operator is context insensitive, and may produce children with less than k groups.

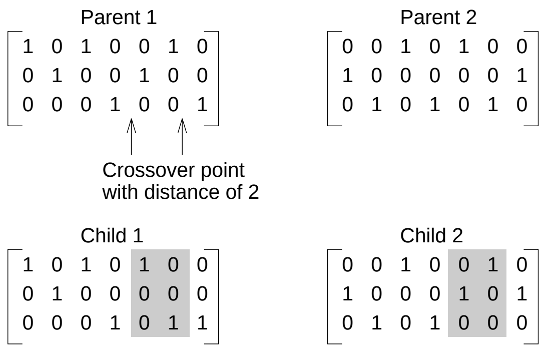


Figure 15: Bezdek et al.'s 2-point matrix crossover (adapted from [6]).

- **Permutation with separators**

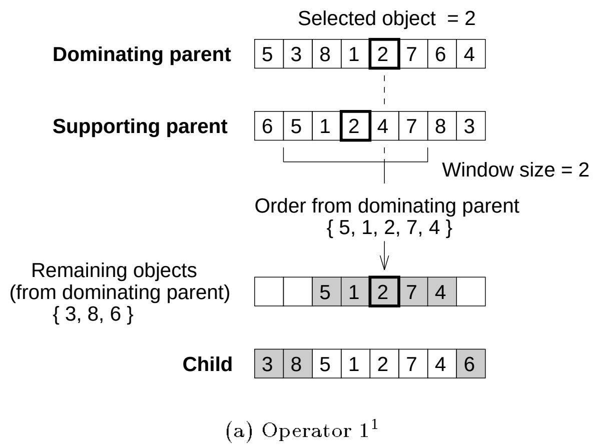
  Jones and Beltramo [36] use two crossovers for permutation encodings, PMX and OX (see Section 1.3.3). In both cases parents are repeatedly crossed until the child decodes into a clustering with k groups.

- **Greedy permutation**

  Jones and Beltramo [36] also use PMX and OX crossover for the greedy permutation representation.

- **Order-based**

  Bhuyan et al. [8] describe two operators for their order-based representation. Operator 1 (Figure 16a) randomly selects a dominating parent; the remaining parent becomes the supporting parent. Next an object and a distance or window size are selected at random. These define the substring which is copied from the supporting parent to the child — the substring is the selected object plus any other objects within the window size (either side of the selected object). The substring is copied into the child so that the selected object is in the same position as in the dominating parent, and the order of the objects in the dominating



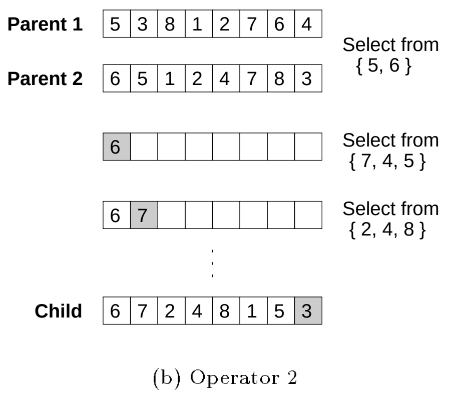


Figure 16: Bhuyan et al.'s crossover operators (adapted from [8]). [1]placement order differs slightly from that shown.

parent is maintained. The remaining objects are placed into the child in the same order they appear in the dominating parent.

Operator 2 (Figure 16b) starts by selecting the object in the first position of one of the parents, and placing this in the first position of the child. We then select an object from all the objects that are next to this object in either parent. The selected object becomes the second object in the child. This process continues until all of the objects are represented in the child.

A third operator is mentioned in the results section but is never described.

Bhuyan's [7] crossover operator is similar to Operator 1 above, in that a substring from a supporting parent is copied to the dominating parent (Figure 17). However, in this case we copy complete clusters (one or more) rather than random substrings. First, the "borrowed"

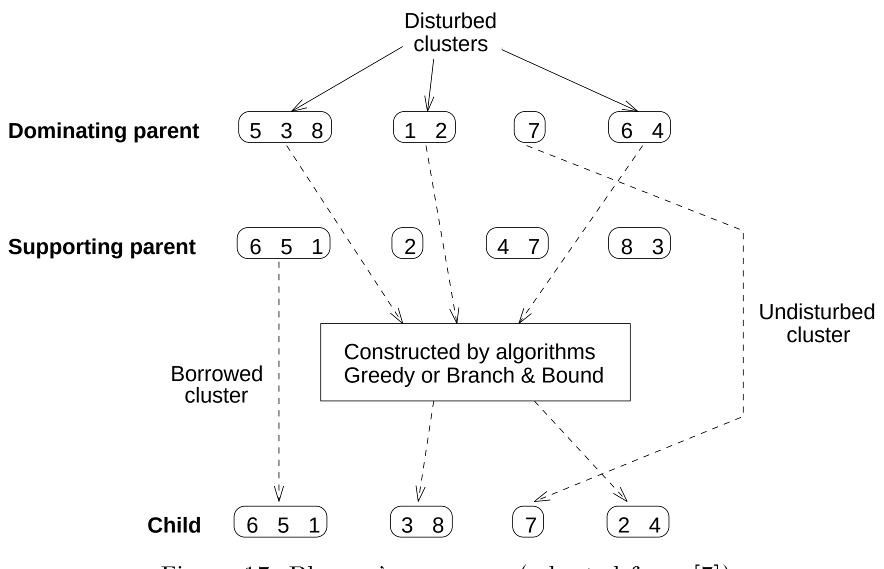


Figure 17: Bhuyan's crossover (adapted from [7]).

clusters are added to the dominating parent. Then we rearrange the remaining clusters in the dominating parent to make it a valid clustering. This is done by removing any clusters that contain objects also in the borrowed clusters, and using a local search to add any missing objects to the clustering. This is essentially the same operator Von Laszewski [66] used for group-number encoding.

Mutation

Mutation introduces new genetic material into the population. In a clustering context this corresponds to moving an object from one cluster to another. How this is done is dependent on the representation.

- **Group-number**

  Krovi [43] uses the mutation function implemented by Goldberg [24]. Here each bit of the chromosome is inverted with a probability equal to the mutation rate, P_{mut}. Jones and Beltramo [36] change each group number (provided it is not the only object left in that group) with probability, $P_{mut} = \frac{1}{n}$ where n is the number of objects .

- **Matrix**

  Alippi and Cucchiara [1] use a column mutation. An element is selected from the matrix at random and set to 1. All other elements in the column are set to 0. If the selected element is already 1 this operator has no effect.

  Bezdek et al. [6] also use a column matrix, but they choose an element that is currently 0 and set it to 1 (Figure 18). The element that is 1 is set to 0.

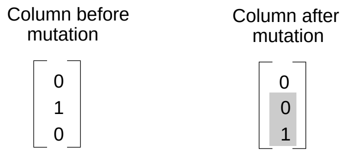


Figure 18: Bezdek et al.'s mutation (adapted from [6]).

- **Permutation with separators**

  Jones and Beltramo [36] randomly select two objects and swap them. To ensure that the resulting chromosome is valid, group separators cannot be swapped.

- **Greedy permutation**

  Two objects are randomly selected and swapped [36].

- **Order-based**

  The mutation operator used by Bhuyan, et al. [8] is the same as that used for the greedy permutation representation. Bhuyan [7] compares two mutation operators. The first moves a randomly selected object to a randomly selected cluster; the second moves the object only if the move results in a decrease in the objective function value.

Other Operators

- **Order-based**
  Bhuyan et al. [8] apply a local-tuning operator after a specified number of generations. Each object is considered in turn, and moved to the first cluster that reduces the value of the objective function.

2.1.4 Parameter Values

The parameter values used for the genetic clustering implementations vary considerably, and there is little or no documented justification for the selection of these values. Some parameters values are even omitted. Population sizes range from 40 [8, 7] to 1000 [36]; the number of generations varies from 40 [8] to 200 [6] or to complete convergence; crossover probabilities are high (0.70, 1.0); and mutation rates are low (0.1, 0.3) — although these values are high compared to typical mutation rates for genetic algorithms.

2.2 Methodology

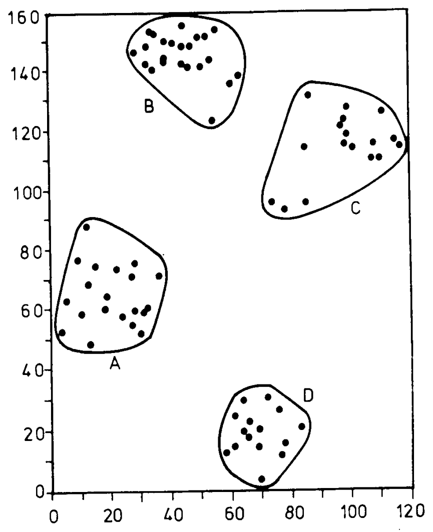
In order to determine the best adaptions for the clustering problem, and the relationship between the data and these adaptions, we compared the performance of a range of adaptions over a number of data sets. This was achieved as follows:

1. We selected three data sets with differing numbers of objects and clusters. For each of these we generated a similar data set (same number of objects, attributes, and clusters; similar cluster shape and proximity).
2. Exhaustive tests with different combinations of adaptions were performed on the generated data sets.
3. Using the test results from Step 2, suitable adaptions were selected for each of the generated data sets. We then compared the performance of these adaptions on the original and generated data.

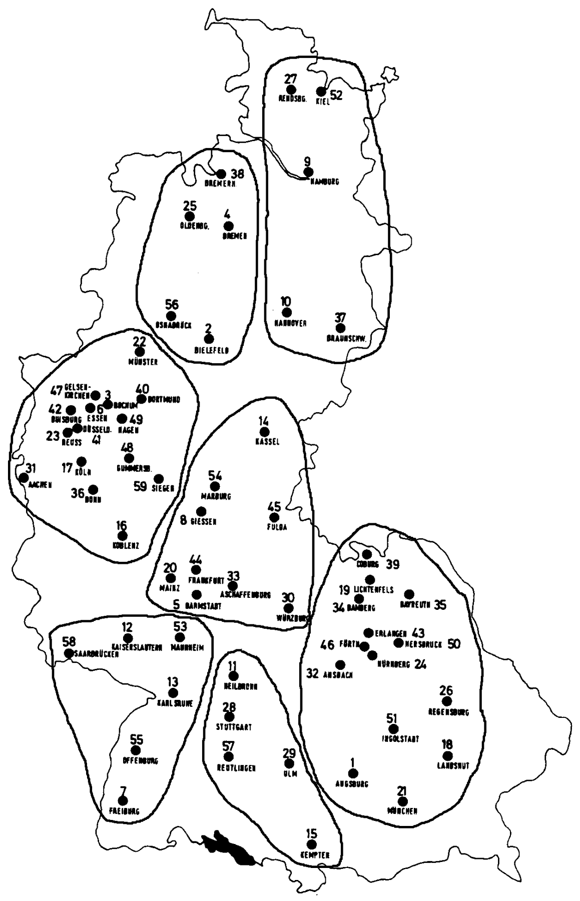
2.2.1 Data Sets

The selected data sets were Ruspini [55, 39], German Towns [60] and Iris [51] (Figure 19); the properties of these data sets are listed in Table 2. The Ruspini data represents a relatively small number of objects clustered into a moderate number of well separated clusters. The German Towns data has a small number of objects, but there is no “correct” clustering for this data set. The data can be “successfully” divided into differing numbers of clusters (although five, seven, or nine clusters appear to best suit the data [60]). For this experiment we chose the correct number of clusters to be seven. The Iris data set contains a larger number of objects which are divided amongst three hyperspherical clusters. Two of these clusters overlap slightly.

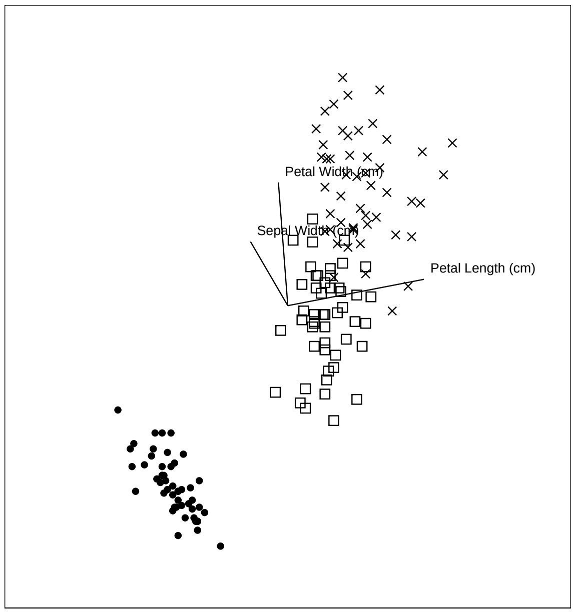
The new data sets were generated with the same number of objects, attributes, and clusters as the original data sets. Further, the shape and proximity of the clusters were based on those found



(a) Ruspini (from [39])



(b) German Towns (adapted from [60])



(c) Iris

Figure 19: Two dimensional plots of the selected data sets.

	Number of objects	Number of attributes	Number of clusters
Ruspini	75	2	4
German Towns	59	2	7
Iris	150	4	3

Table 2: Properties of the experimental data sets.

in the original sets. However, the range of attribute values, the number of objects in the individual clusters, and the relative positioning of the clusters was purposely altered for the generated data sets. Finally, the shape and positioning of the clusters was adjusted so that the correct clustering corresponded to the maximum value of the objective function (or that the clusters were clearly separable).

2.2.2 Objective Function

The objective function for the experiment was trace (W) or the sum of the squared distances between objects and their cluster centres (see section 1.1.3). This was minimised over the solution space.

The attribute values were standardised to minimise the difference in objective function values for different data sets.

2.2.3 Adaptions

The following adaptions were compared:

- **Representation**
  Both the group-number and order-based representations were implemented. The poor performance of the GCAs with order-based representation meant that these trials took longer to complete, and due to time constraints the range of adaptions compared for this representation was reduced.

- **Fitness Function**
  Four different scaling mechanisms were used for the group-number representation: local transformation, adjusted transformation, linear scaling with $C_{Mult} = 2.0$, and linear scaling with $C_{Mult} = 4.0$. Adjusted and linear scaling were also compared for the order-based representation.

- **Selection**
  Fitness proportional selection was compared with three different elite levels: the top 0, 1, or 5% of population size individuals were copied straight from one generation to the next. Elite levels of 0 and 5% were used with order-based encoding.

- **Crossover**
  For the group-number representation, single-point, uniform, and edge-based crossover were compared. For order-based encoding the PMX crossover operator was compared with two new operators. The first, *borrow*, is similar to the operator described by both Bhuyan [7] and

Von Laszewski [66], but the operator has been designed to avoid the need for local search. A single cluster (randomly selected) is copied from the first parent and placed into the child. The remaining objects are added to the child in the order they occur in the second parent.

The second operator is an edge-based operator for order-based encodings. This operator works exactly the same as for the group-number representation, but the objects are placed in the child in the same order they occur in the first parent.

- **Mutation**

  The mutation implemented for the group-number representation randomly changes a group-number with probability $\frac{P_{mut}}{n}$. Two types of mutation were compared for order-based encoding. The first, *uniform*, moves an object with probability $\frac{P_{mut}}{n}$ to a uniformly distributed point along the length of the chromosome. The second, *gaussian*, moves the object according to a Gaussian distribution with a mean of 0 and a standard deviation of $\frac{n}{4}$.

- **Parameters**

  The following parameters were compared for the group-number representation: population size $\in \{50, 100, 200, 400\}$; crossover probability $\in \{0.50, 0.70, 0.90\}$; and mutation rate $\in \{0.01, 0.05, 0.10, 0.20, 0.50, 0.70, 0.90\}$. For order-based these were reduced to: population size $\in \{50, 100, 200\}$; crossover probability $\in \{0.50, 0.70, 0.90\}$; and mutation rate $\in \{0.10, 0.20, 0.70, 0.90\}$.

All of the GCAs used random initialisation, and replaced the entire population during the reproduction phase (except in the case of elite selection). There were no checks to ensure that all chromosomes contained k groups.

Since the operators are dependent on the representation type, the experimental tests were divided into two groups according to the representation type. Each group involved five replications (using different random seeds) of each possible combination of relevant adaptions. There were 3024 combinations of adaptions for the group-number representation and 1296 combinations for order-based encoding. These combinations were tested on each of the three data sets, giving a total of six result sets according to the representation and the data.

Each run of the GCA continued until the correct clustering was found or the execution time exceeded five CPU minutes which was considered a reasonable amount of time to find a solution. The number of correct runs (maximum of five) and the average time to find the solution (calculated from those runs that found the solution) were recorded for each combination.

2.2.4 Testing Conditions

All of the tests were conducted on one of two Sparc Ultras (all runs for a particular representation/data combination were conducted on the same machine). The load average on both machines during testing was approximately 1.0.

2.2.5 Statistical Analysis

Each of the six representation type/data set results were analysed separately. Independent generalised linear models were fitted to the correctness and time results. These were used to determine the factors influencing correctness and speed, and also to predict which adaptions would give the highest probability of a correct clustering in the fastest time. All statistical analysis was completed using S-Plus version 3.4 release 1 for Silicon Graphics Iris, IRIX 5.3: 1996.

Generalised linear models were fitted to both the correctness and time data. The probability of finding the correct clustering was assumed to follow the logit model

$$\pi = \frac{\exp(z'\alpha)}{1+\exp(z'\alpha)},$$

with the design vector z' =(1, PS, Tr, El, Cr, CP, Mu, MR, PS:Tr, PS:El, ...). The symbols in the design vector represent population size, fitness transformation, elite constant, crossover type, crossover probability, mutation type, and mutation rate. Terms such as PS:Tr represent the interaction between these two factors, namely the population size and the fitness transformation.

The time to find the solution was assumed to follow the Poisson model (with z' as above)

$$\mu = \exp(z'\beta).$$

Each model was assumed to be hierarchical, that is higher-order terms were only included if lower-order related terms were also included. The final models were the best possible models with no higher than third-order terms. All of the variables (including those continuous) were treated as factors as the purpose of the analysis was to find the best of the selected factors levels. Analysis of deviance was used to judge the significance of terms, with insignificant terms being dropped from the models. Information on generalised linear models can be found in [64, 47, 16].

The mean time to solution on the real and generated data sets was compared with the predicted values using two-sided z tests (or t tests when there were less than 30 correct runs). Each test used the null hypothesis that the actual mean time was equal to the predicted time.

2.3 Results

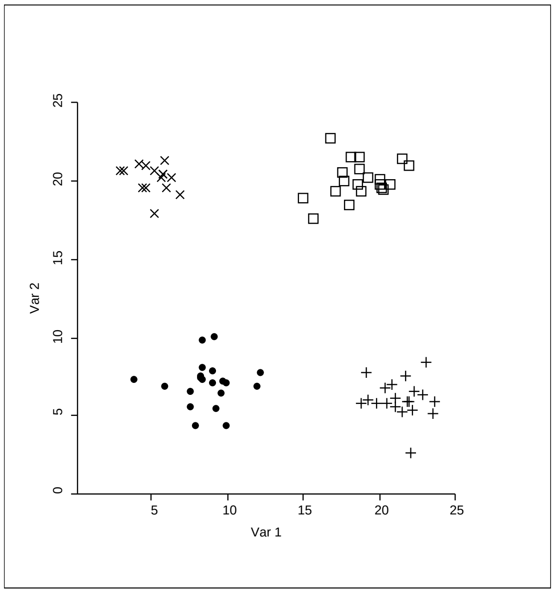
2.3.1 Generated Data

Figure 20 contains the three generated data sets: Ruspini2, Towns2, and Iris2.

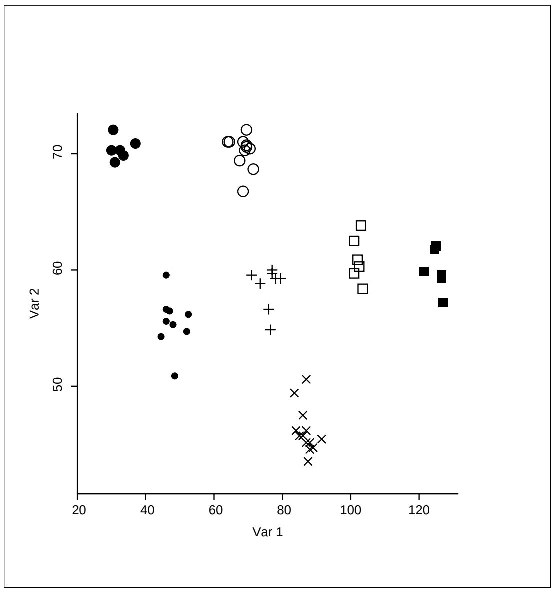
2.3.2 Adaptions

Table 3 contains the correctness results for the group-number GCAs. The majority of the GCAs found the correct clustering for both Ruspini2 and Iris2 in all five runs (79.1% and 73.2 % respectively). However, only 4% found the solution for the Towns2 data in all five runs. 97.8% of the group-number GCAs found the correct clustering for Ruspini2 four or five times — with all GCAs finding the solution in at least two runs for this data set. A considerable proportion of the GCAs did not find the solution in any run for both Towns2 (10.8%) and Iris2 (15.1%).

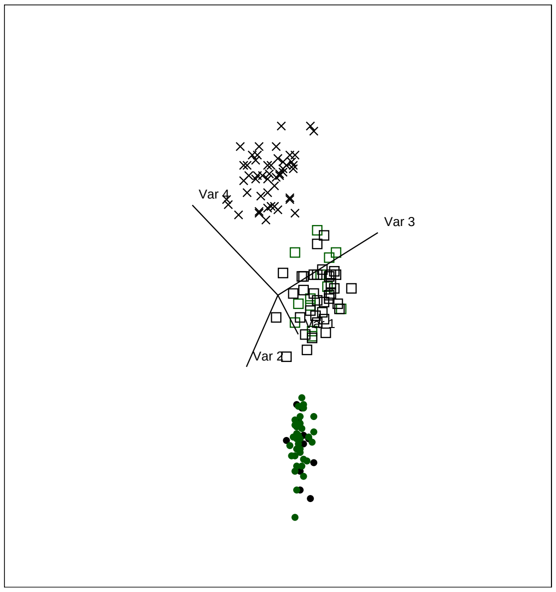
The correctness results for the order-based trials are presented in Table 4. For each data set, a large proportion of the trials did not find the correct clustering in any run. In fact, for the Iris2 data set, the GCAs were unable to find the solution in any run of any trial. However, a reasonable proportion of the GCAs found the correct clustering in all five runs for both Ruspini2 and Towns2.



(a) Ruspini2



(b) Towns2



(c) Iris2

Figure 20: Two dimensional plots of the generated data sets.

		No. of Correct Runs					
		0	1	2	3	4	5
Ruspini2	No. of Trials	0	0	4	64	564	2392
	% of Trials	0.0	0.0	0.1	2.1	18.7	79.1
Towns2	No. of Trials	327	549	849	783	394	122
	% of Trials	10.8	18.2	28.1	25.9	13.0	4.0
Iris2	No. of Trials	456	70	61	64	157	2216
	% of Trials	15.1	2.3	2.0	2.1	5.2	73.2

Table 3: Correctness of group-number GCAs.

		No. of Correct Runs					
		0	1	2	3	4	5
Ruspini2	No. of Trials	547	63	47	78	104	457
	% of Trials	42.2	4.9	3.6	6.0	8.0	35.3
Towns2	No. of Trials	578	56	40	48	66	508
	% of Trials	44.6	4.3	3.1	3.7	5.1	39.2
Iris2	No. of Trials	1296	0	0	0	0	0
	% of Trials	100.0	0.0	0.0	0.0	0.0	0.0

Table 4: Correctness of order-based GCAs.

Table 5 contains the time results for the group-number GCAs. The GCAs were able to find solutions to the Ruspini2 data set in the shortest time, with 95.3% of the trials finding at least one solution in under 120 seconds. The mean time to solution for Iris2 was higher than that for the other two data sets, and the mean time for the Towns2 data set was higher than that for Ruspini2.

The time results for the order-based GCAs can be found in Table 6. The results for the Iris2 data set were not included as no solutions were found. Notice that the recorded times for the Towns2 data set are less than those of Ruspini2. In both cases, the times are greater than those of the corresponding group-number trials.

We now present the results for each of the representation/data set combinations in greater detail. Tables 7 to 10 contain the correctness distributions for the various adaptions, and Figures 21 to 25 provide the corresponding boxplots of the time data. Tables 12 and 13 contain the final models for these results. Further detail on the models can be found in Appendix A. There are no models for the order-based GCAs on Iris2 since no solutions were found.

	Time to solution^1			Percentage of trials^2			
	Minimum	Mean	Median	$<$ 20 sec	$<$ 60 sec	$<$ 120 sec	$<$ 300 sec
Ruspini2	1.98	41.3	27.1	40.1	76.1	95.3	100.0
Towns2	2.50	77.8	58.3	16.6	45.2	69.6	89.2
Iris2	6.41	115.0	89.5	5.2	29.5	51.4	85.0

Table 5: Time to solution for group-number GCAs. ^{1}seconds CPU. ^{2}that found one or more solution(s).

	Time to solution[1]			Percentage of trials[2]			
	Minimum	Mean	Median	< 20 sec	< 60 sec	< 120 sec	< 300 sec
Ruspini2	50.2	183.0	184	0.0	0.5	11.0	57.3
Towns2	20.5	124.0	91.4	0.0	18.0	32.5	55.4

Table 6: Time to solution for order-based GCAs. [1]seconds CPU. [2]that found one or more solution(s).

Group-number and Ruspini2

For each of the distribution tables we are looking for adaptions that give an increased probability of finding the correct solution. In this particular case, the number of trials with less than four correct runs is small and we should concentrate on the last two columns in each table. The adaptions that show marked changes in proportion are population size (200 or 400 appears best), crossover (edge-based out-performs uniform and single-point), and mutation (0.20 – 0.90 are more successful). The final model includes these terms as well as a few higher-order interaction terms.

PS	No. of Correct Runs			
	2	3	4	5
50	25.0	20.3	31.9	23.5
100	25.0	28.1	27.0	24.5
200	0.0	20.3	20.4	26.3
400	50.0	31.3	20.7	25.8

Tr	No. of Correct Runs			
	2	3	4	5
adjust	0.0	25.0	29.6	24.0
local	25.0	28.1	22.5	25.5
scale2.0	75.0	25.0	24.3	25.1
scale4.0	0.0	21.9	23.6	25.5

El	No. of Correct Runs			
	2	3	4	5
0	75.0	34.4	33.9	33.1
1	25.0	31.3	30.9	34.0
5%[1]	0.0	34.4	35.3	32.9

Cr	No. of Correct Runs			
	2	3	4	5
edge	50.0	18.8	21.1	36.6
single	25.0	40.6	38.5	31.9
uniform	25.0	40.6	40.4	31.5

CP	No. of Correct Runs			
	2	3	4	5
0.50	75.0	42.2	33.9	32.9
0.70	0.0	26.6	33.7	33.5
0.90	25.0	31.3	32.4	33.6

MR	No. of Correct Runs			
	2	3	4	5
0.01	0.0	18.8	17.9	13.3
0.05	25.0	20.3	16.5	13.6
0.10	0.0	18.8	17.2	13.5
0.20	0.0	15.6	13.1	14.5
0.50	0.0	9.4	11.7	15.1
0.70	0.0	10.9	13.5	14.6
0.90	75.0	6.3	10.1	15.4

Table 7: Empirical distributions of the adaptions with group-number encoding for Ruspini2 (relative frequencies in percents). [1]of population size.

All of the first-order factors are included in the time model. The larger population sizes took more time to find the correct clustering, plus there was a greater range of times; the adjusted fitness transform produced faster and more consistent times; increasing the elite constant decreased the time to find the solution; single-point crossover was clearly the fastest crossover, with edge-based being the slowest; there was not much difference in the average times between the crossover probabilities, but the 0.50 level appears to be slightly more consistent; mutation rates of 0.50 and 0.70 appear to give the fastest results.

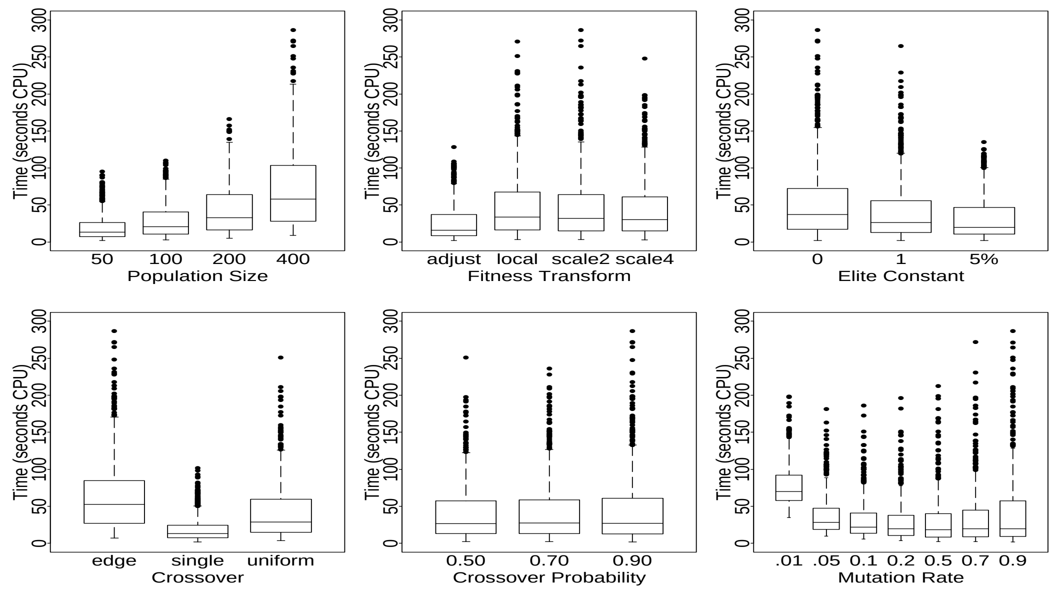


Figure 21: Time results for group-number GCAs on Ruspini2.

PS	No. of Correct Runs					
	0	1	2	3	4	5
50	11.6	26.4	27.4	25.7	29.9	17.2
100	16.5	22.6	27.1	27.2	23.9	33.6
200	26.0	26.8	22.0	25.8	24.6	31.1
400	45.9	24.2	23.4	21.3	21.6	18.0

Tr	No. of Correct Runs					
	0	1	2	3	4	5
adjust	11.0	26.2	28.7	27.3	25.4	14.8
local	27.8	22.8	24.3	25.2	26.1	27.9
scale2.0	29.7	25.9	23.9	23.8	25.1	23.8
scale4.0	31.5	25.1	23.1	23.8	23.4	33.6

El	No. of Correct Runs					
	0	1	2	3	4	5
0	64.2	45.2	32.3	24.5	17.3	13.1
1	23.9	26.0	33.8	35.6	42.9	42.6
5%[1]	11.9	28.8	33.9	39.8	39.8	44.3

Cr	No. of Correct Runs					
	0	1	2	3	4	5
edge	60.9	26.4	28.4	29.5	30.7	58.2
single	11.6	33.0	36.3	39.0	36.8	25.4
uniform	27.5	40.6	35.3	31.5	32.5	16.4

CP	No. of Correct Runs					
	0	1	2	3	4	5
0.50	26.6	38.6	32.7	35.6	29.7	28.7
0.70	35.5	32.2	32.9	33.8	33.0	33.6
0.90	37.9	29.1	34.4	30.5	37.3	37.7

MR	No. of Correct Runs					
	0	1	2	3	4	5
0.01	14.7	16.8	16.7	12.4	11.9	4.9
0.05	8.9	16.6	17.8	13.9	10.4	9.0
0.10	10.1	17.3	16.4	12.4	13.5	12.3
0.20	8.6	13.7	13.9	16.3	14.7	20.5
0.50	13.1	12.8	13.8	15.3	16.2	14.8
0.70	18.0	12.0	10.1	16.1	18.0	19.7
0.90	26.6	10.9	11.3	13.5	15.2	18.9

Table 8: Empirical distributions of the adaptions with group-number encoding for Towns2 (relative frequencies in percents). [1]of population size.

Group-number and Towns2

For the Towns2 data set, lower population sizes (50 – 200) seem to increase the probability of correctness, although the proportion of trials with population size of 50 and five runs correct is relatively small. The treatment data is difficult to interpret, with each method represented in the same proportions in the trials with 0 and 5 correct. Elite selection clearly improved the chances of finding the solution in all five runs, with 64.2% of the GCAs with 0 correct runs lacking elite selection. Edge-based crossover also corresponds to a high proportion of trials with 5 correct runs. However, 60.9% of GCAs with 0 correct runs also have edge-based crossover. The crossover probability did not significantly influence the correctness. Finally, higher mutation rates seem to improve the probability of correctness.

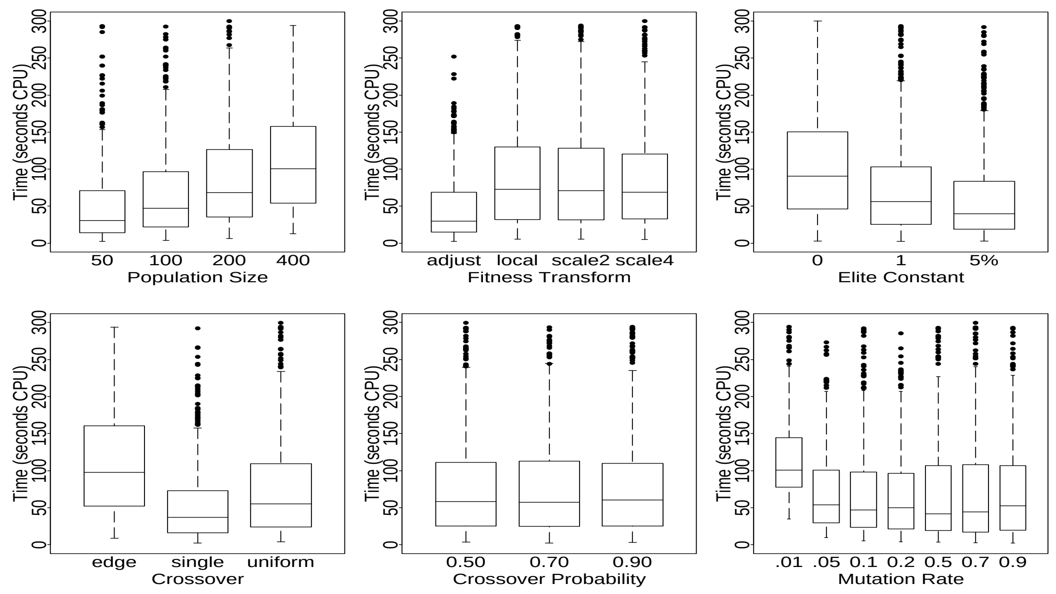


Figure 22: Time results for group-number GCAs on Towns2.

The time data for the Towns2 data set follows the trends of the previous time data although the average times are higher and the deviance of the values is greater. The fitted models are similar, although the model for Towns2 contains more terms.

Group-number and Iris2

Once again, low population size appears to increase the probability of correctness, and this time the population size of 50 appears the best choice. The adjusted fitness transformation clearly increases the chances of finding the solution. Elite selection also represents a higher proportion of the trials with higher correctness. The most successful crossover is single-point, with edge-based crossover the least successful (75.4% of the GCAs with 0 correct runs used edge-based crossover). Lower crossover probabilities seem to slightly increase the probability of correctness, and again the mutation rate significantly influences correctness.

PS	No. of Correct Runs					
	0	1	2	3	4	5
50	3.1	5.7	8.2	7.8	22.9	31.2
100	5.5	24.3	24.6	32.8	34.4	28.2
200	30.7	42.9	42.6	39.1	24.2	22.4
400	60.7	27.1	24.6	20.3	18.5	18.2

Tr	No. of Correct Runs					
	0	1	2	3	4	5
adjust	3.3	7.1	14.8	23.4	24.8	30.4
local	34.0	30.0	29.5	26.6	26.8	22.7
scale2.0	31.8	38.6	31.1	17.2	26.1	23.1
scale4.0	30.9	24.3	24.6	32.8	22.3	23.8

El	No. of Correct Runs					
	0	1	2	3	4	5
0	53.9	57.1	57.4	45.3	41.4	26.8
1	27.6	25.7	29.5	29.7	28.7	35.3
5%[1]	18.4	17.1	13.1	25.0	29.9	38.0

Cr	No. of Correct Runs					
	0	1	2	3	4	5
edge	75.4	45.7	45.9	42.2	49.7	22.5
single	0.4	5.7	11.5	14.1	15.9	43.4
uniform	24.1	48.6	42.6	43.8	34.4	34.1

CP	No. of Correct Runs					
	0	1	2	3	4	5
0.50	31.8	28.6	26.2	29.7	33.1	34.1
0.70	33.1	30.0	31.1	40.6	35.0	33.2
0.90	35.1	41.4	42.6	29.7	31.8	32.7

MR	No. of Correct Runs					
	0	1	2	3	4	5
0.01	21.9	37.1	49.2	51.6	47.8	7.6
0.05	12.1	15.7	11.5	14.1	10.8	15.0
0.10	9.9	7.1	11.5	10.9	9.6	15.9
0.20	9.6	8.6	4.9	9.4	7.6	16.3
0.50	11.4	10.0	9.8	3.1	8.3	15.9
0.70	15.4	14.3	8.2	7.8	7.0	14.9
0.90	19.7	7.1	4.9	3.1	8.9	14.4

Table 9: Empirical distributions of the adaptions with group-number encoding for Iris2 (relative frequencies in percents). [1]of population size.

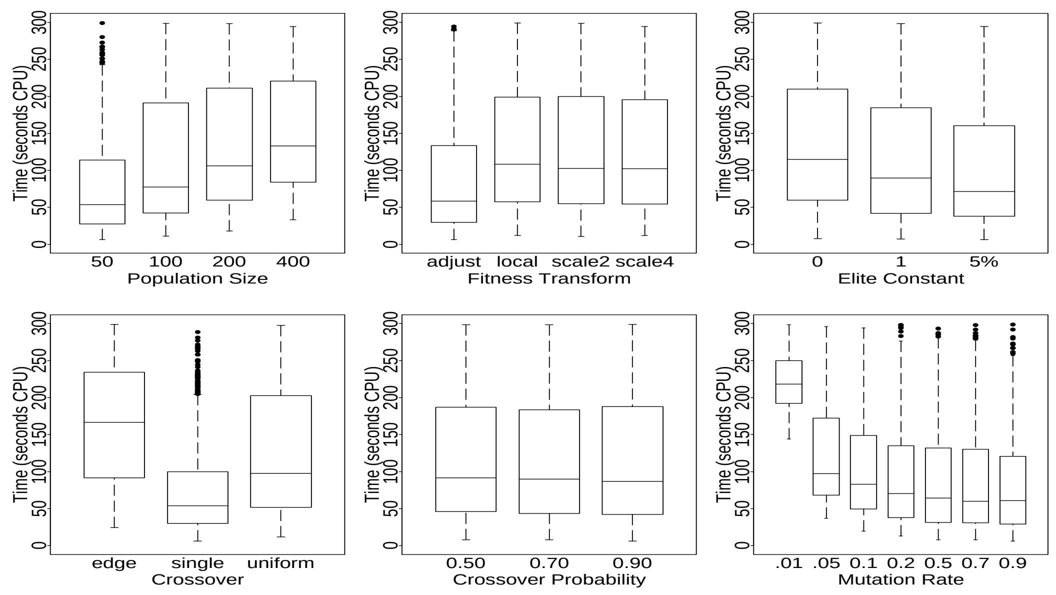


Figure 23: Time results for group-number GCAs on Iris2.

The time results for the Iris2 data set follow the same trends as the two previous models, although the averages and spreads of the values in this case are greater than those of both the previous models. The time models for all three data sets are similar.

Order-based and Ruspini2

Here the GCAs with smaller population sizes (50, 100) achieved a higher level of correctness. The scaling transformations were more likely to produce five correct runs than the adjusted transformation, although the trials with 0 correct were divided fairly evenly between the three. There was a slight advantage to using elite selection. Edge-based crossover clearly out-performed the other crossovers (75% of the trials with 5 correct runs used edge-based crossover). Further, none of the GCAs using PMX crossover found the solution more than three times in any trial, and 77.9% of the trials with 0 correct runs used PMX crossover.

PS	No. of Correct Runs					
	0	1	2	3	4	5
50	28.2	41.3	25.5	38.5	41.3	36.5
100	30.3	22.2	42.6	38.5	35.6	36.1
200	41.4	36.5	31.9	23.1	23.1	27.4

Tr	No. of Correct Runs					
	0	1	2	3	4	5
adjust	32.4	58.7	38.3	42.3	40.4	27.4
scale2.0	34.6	15.9	36.2	29.5	24.0	36.8
scale4.0	33.1	25.4	25.5	28.2	35.6	35.9

El	No. of Correct Runs					
	0	1	2	3	4	5
0	52.3	44.4	53.2	47.4	47.1	48.8
5%[1]	47.7	55.6	46.8	52.6	52.9	51.2

Cr	No. of Correct Runs					
	0	1	2	3	4	5
borrow	21.6	81.0	80.9	69.2	64.4	22.8
edge	0.5	12.7	17.0	29.5	35.6	77.2
PMX	77.9	6.3	2.1	1.3	0.0	0.0

CP	No. of Correct Runs					
	0	1	2	3	4	5
0.50	39.3	36.5	46.8	34.6	28.8	25.2
0.70	31.8	44.4	29.8	33.3	38.5	32.8
0.90	28.9	19.0	23.4	32.1	32.7	42.0

Mu	No. of Correct Runs					
	0	1	2	3	4	5
gaussian	49.5	60.3	57.4	46.2	48.1	49.5
uniform	50.5	39.7	42.6	53.8	51.9	50.5

MR	No. of Correct Runs					
	0	1	2	3	4	5
0.10	26.3	50.8	38.3	44.9	20.2	16.2
0.20	23.4	22.2	36.2	26.9	39.4	22.5
0.70	23.9	14.3	17.0	14.1	24.0	30.6
0.90	26.3	12.7	8.5	14.1	16.3	30.6

Table 10: Empirical distributions of the adaptions with order-based encoding for Ruspini2 (relative frequencies in percents). [1]of population size.

Only six of the GCAs with PMX crossover found the solution in one or more runs, so these GCAs were removed before the time data was analysed. As with the group-number GCAs, larger populations resulted in a longer time to find the solution and increasing the mutation rate tended to decrease this time. However, the adjusted transformation was considerably slower than the scaling transformations, elite selection did not decrease the time to solution (in fact it increased it slightly), edge-based crossover found the solution faster than the simpler borrow operator, and increasing the crossover probability resulted in a marked decrease in the average time to find the solution. Finally, there was a slight time advantage to using uniform rather than gaussian mutation.

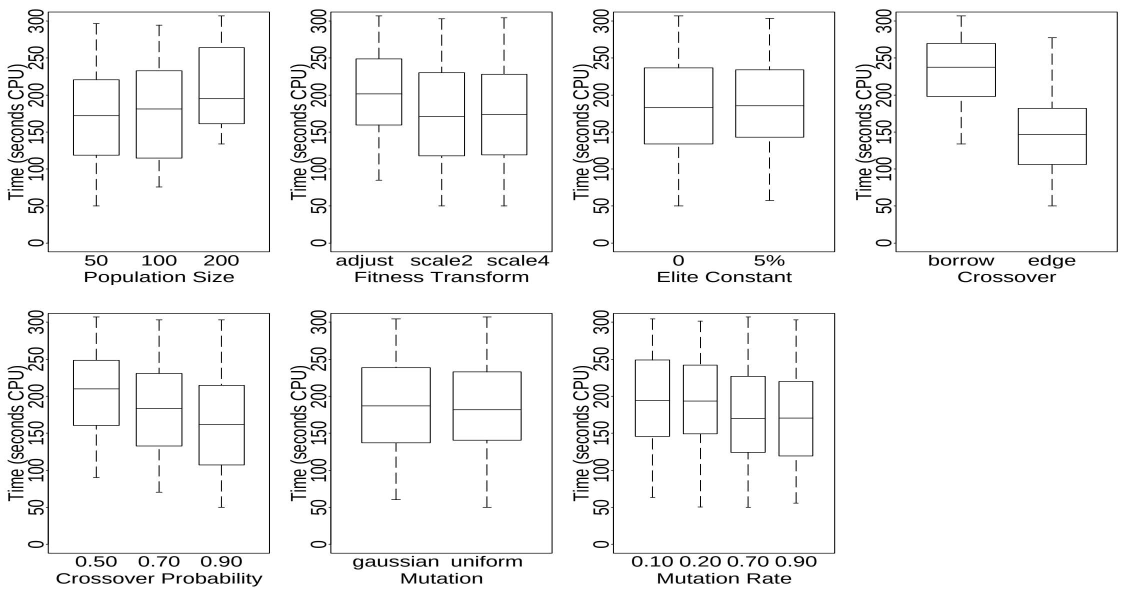


Figure 24: Time results for order-based GCAs on Ruspini2.

PS	No. of Correct Runs					
	0	1	2	3	4	5
50	27.7	33.9	37.5	33.3	47.0	37.6
100	31.1	37.5	30.0	47.9	37.9	33.7
200	41.2	28.6	32.5	18.8	15.2	28.7

Tr	No. of Correct Runs					
	0	1	2	3	4	5
adjust	26.0	42.9	40.0	39.6	37.9	39.0
scale2.0	37.4	23.2	30.0	31.3	27.3	31.1
scale4.0	36.7	33.9	30.0	29.2	34.8	29.9

El	No. of Correct Runs					
	0	1	2	3	4	5
0	55.7	51.8	40.0	37.5	27.3	48.2
5%[1]	44.3	48.2	60.0	62.5	72.7	51.8

Cr	No. of Correct Runs					
	0	1	2	3	4	5
borrow	26.8	87.5	95.0	100.0	95.5	15.6
edge	0.0	0.0	0.0	0.0	4.5	84.4
PMX	73.2	12.5	5.0	0.0	0.0	0.0

CP	No. of Correct Runs					
	0	1	2	3	4	5
0.50	36.3	46.4	42.5	14.6	22.7	30.9
0.70	33.0	32.1	32.5	43.8	25.8	33.9
0.90	30.6	21.4	25.0	41.7	51.5	35.2

Mu	No. of Correct Runs					
	0	1	2	3	4	5
gaussian	49.7	60.7	37.5	47.9	57.6	49.4
uniform	50.3	39.3	62.5	52.1	42.4	50.6

MR	No. of Correct Runs					
	0	1	2	3	4	5
0.10	23.5	30.4	40.0	45.8	37.9	21.3
0.20	23.0	25.0	25.0	29.2	31.8	26.0
0.70	25.8	21.4	17.5	18.8	16.7	26.8
0.90	27.7	23.2	17.5	6.3	13.6	26.0

Table 11: Empirical distributions of the adaptions with order-based encoding for Towns2 (relative frequencies in percents). [1]of population size.

Order-based and Towns2

The distributions for Towns2 are similar to those for Ruspini2, but a higher proportion of GCAs with 5 runs correct used the adjusted fitness transformation. Further, the differences in performance due to crossover probability and mutation rate were not as great, and the dominance of the edge-based crossover was more pronounced (all of the GCAs with edge-based crossover found the solution in four or five runs).

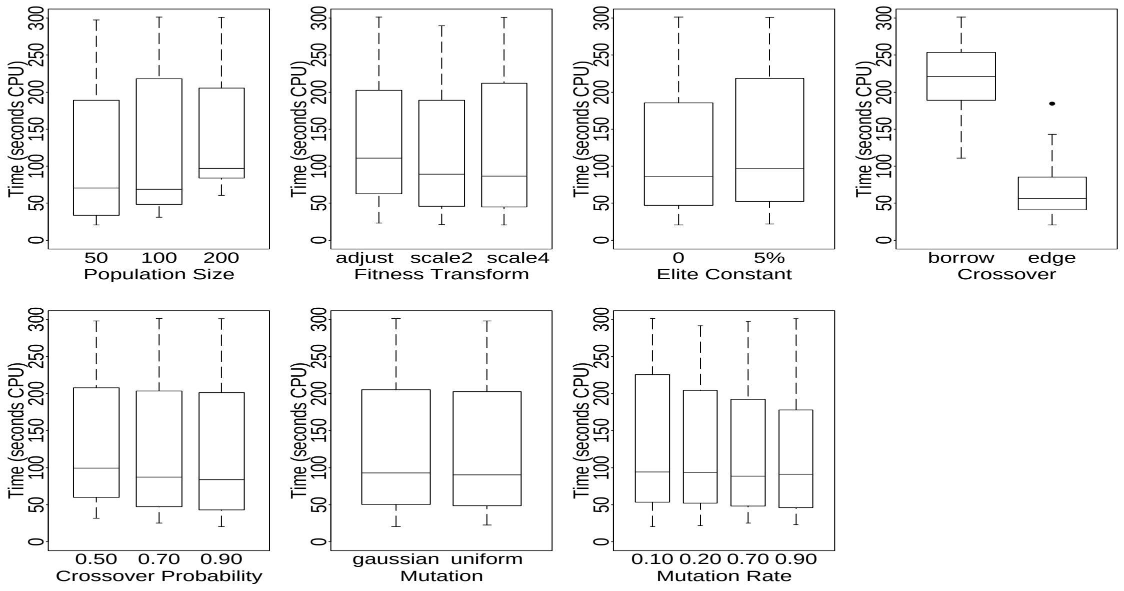


Figure 25: Time results for order-based GCAs on Towns2.

The trends shown in the time results are similar to those of Ruspini2, but the averages for the Towns2 data set are lower and the deviations higher (except for crossover, which has lower averages and smaller deviances). The differences in times for the crossover probability and mutation rate as also not as great. The type of mutation did not significantly influence the time.

The Final Models

The terms which significantly contributed to each model are listed in Tables 12 and 13. A complete listing of the coefficients and discussion of the fit of each model is contained in Appendix A. The group-number correctness models for Ruspini2 and Iris2, and the order-based correctness models for Ruspini2 and Towns2 contain mainly first and second order terms and fit the data well. The correctness model for the group-number GCAs on Towns2 does not fit as well, but is the best third order model for the data. The time models contain a larger number of higher order terms, with the group-number models for Ruspini2 and Iris3 providing a good fit. The remaining time models do not fit as well.

	Ruspini2		Towns2		Iris2	
Factor	Correctness	Time	Correctness	Time	Correctness	Time
PS	**	**	**	**	**	**
Tr	ns	**	*	**	**	**
El		**	**	**	**	**
Cr	**	**	**	**	**	**
CP		**	ns	**	**	**
MR	**	**	**	**	**	**
PS:Tr		**	**	**	**	**
PS:El		**	**	**	**	**
PS:Cr	**	**	**	**	**	**
PS:CP		**	ns	**	**	**
PS:MR	ns	**	**	**	**	**
Tr:El		**	**	**		**
Tr:Cr	**	**	**	**	**	**
Tr:CP			ns	**		**
Tr:MR		**	**	**	**	**
El:Cr		**	**	**		**
El:CP			*	**		**
El:MR		**	**	**	**	**
Cr:CP		**		**	**	**
Cr:MR	ns	**	ns	**	**	**
CP:MR				**		**
PS:Tr:El			**	**		**
PS:Tr:Cr	**	**		**		**
PS:Tr:CP			**	**		**
PS:Tr:MR		**	*	**		
PS:El:Cr			**	**		**
PS:El:CP			*	**		
PS:El:MR		**	*	**		**
PS:Cr:CP				**		
PS:Cr:MR	*	**		**		**
PS:CP:MR				**		**
Tr:El:Cr			**	**		**
Tr:El:CP				**		
Tr:El:MR		**	**	**		**
Tr:Cr:CP				**		
Tr:Cr:MR		**	*	**		**
Tr:CP:MR				**		
El:Cr:CP				**		
El:Cr:MR			**	**		**
El:CP:MR				**		**
Cr:CP:MR				**		**
cont.						

cont.	Ruspini2		Towns2		Iris2	
Factor	Correctness	Time	Correctness	Time	Correctness	Time

Table 12: Factors included in the correctness and time models for the group-number encoding GAs. * = significant at the 5% level; ** = significant at the 1% level; ns = not significant at the 5% level but still included in the model to maintain hierarchical structure; no symbol indicates the factor was not included in the model; : signifies an interaction term.

	Ruspini2		Towns2	
Factor	Correctness	Time	Correctness	Time
PS	**	**	**	**
Tr	**	**	**	**
El	*	**	**	**
Cr	**	**	**	**
CP	**	**	**	**
Mu		**		ns
MR	**	**	**	**
PS:Tr		**		**
PS:El		**	**	**
PS:Cr		**	*	**
PS:CP		**	**	**
PS:Mu		ns		
PS:MR	**	**	*	**
Tr:El		ns	**	**
Tr:Cr	**	**	**	**
Tr:CP		**		**
Tr:Mu		ns		**
Tr:MR		**	**	**
El:Cr	**	**		**
El:CP		*		
El:Mu		ns		
El:MR		**	**	**
Cr:CP	**	**		**
Cr:Mu		ns		
Cr:MR	**	**		**
CP:Mu		ns		
CP:MR	*	**		**
Mu:MR		ns		
PS:Tr:El		*		**
PS:Tr:Cr		**		**
PS:Tr:CP		**		
cont.				

cont.	Ruspini2		Towns2	
Factor	Correctness	Time	Correctness	Time
PS:Tr:MR		**		**
PS:El:Cr				**
PS:El:Mu		*		
PS:El:MR				**
PS:Cr:CP	**			**
PS:Cr:MR		**		**
PS:CP:Mu		*		
PS:CP:MR		**		
PS:Mu:MR		*		
Tr:El:CP		**		
Tr:El:Mu		*		
Tr:El:MR		**		
Tr:Cr:CP		**		
Tr:Cr:MR				*
Tr:CP:Mu		*		
Tr:CP:MR		**		**
El:Cr:MR		**		
El:CP:Mu		**		
Cr:CP:MR		*		
Cr:Mu:MR		**		

Table 13: Factors included in the correctness and time models for the order-based GAs. * = significant at the 5% level; ** = significant at the 1% level; ns = not significant at the 5% level but still included in the model to maintain hierarchical structure; no symbol indicates the factor was not included in the model; : signifies an interaction term. Factors missing from the table are not in either model.

The top five GCAs for each representation/data set combination are presented in Tables 14 to 18. In each case the correctness model was used to *predict* the adaptions that would give the highest probability of finding the correct clustering. These adaptions were then ranked according to the predictions of the appropriate time model.

PS	Tr	El	Cr	CP	MR	Correctness	Time
200	scaled 4.0	5%	edge-based	0.50	0.50	1.000	42.954
200	scaled 4.0	5%	edge-based	0.70	0.50	1.000	50.187
200	scaled 4.0	5%	edge-based	0.90	0.50	1.000	57.205
200	scaled 4.0	1	edge-based	0.50	0.50	1.000	61.510
200	scaled 4.0	0	edge-based	0.50	0.50	1.000	68.034

Table 14: Top five predicted adaption combinations for group-number GCAs on Ruspini2.

PS	Tr	El	Cr	CP	MR	Correctness	Time
100	scaled 2.0	1	edge-based	0.70	0.50	0.826	109.291
50	scaled 2.0	1	single-point	0.50	0.70	0.819	8.615
100	scaled 2.0	1	single-point	0.70	0.90	0.818	37.170
50	scaled 2.0	1	single-point	0.90	0.70	0.806	9.761
100	scaled 2.0	1	edge-based	0.70	0.70	0.804	150.931

Table 15: Top five predicted adaption combinations for group-number GCAs on Towns2.

PS	Tr	El	Cr	CP	MR	Correctness	Time
50	adjusted	5%	single-point	0.90	0.70	1.000	6.678
50	adjusted	0	single-point	0.90	0.70	1.000	7.176
50	adjusted	5%	single-point	0.70	0.70	1.000	7.432
50	adjusted	1	single-point	0.70	0.70	1.000	7.635
50	adjusted	5%	single-point	0.50	0.90	1.000	7.773

Table 16: Top five predicted adaption combinations for group-number GCAs on Iris2.

PS	Tr	El	Cr	CP	Mu	MR	Correctness	Time
50	scaled 2.0	0	edge-based	0.90	uniform	0.70	0.999983	57.317
50	scaled 2.0	0	edge-based	0.90	gaussian	0.70	0.999983	68.455
50	scaled 2.0	5%	edge-based	0.90	uniform	0.70	0.999981	57.4119
50	scaled 2.0	5%	edge-based	0.90	gaussian	0.70	0.999981	66.6432
50	scaled 2.0	0	edge-based	0.90	gaussian	0.70	0.999975	59.171

Table 17: Top five predicted adaption combinations for order-based GCAs on Ruspini2.

PS	Tr	El	Cr	CP	Mu	MR	Correctness	Time
50	scaled 4.0	0	edge-based	0.90	gaussian	0.10	1.000	23.888
50	scaled 4.0	0	edge-based	0.90	uniform	0.10	1.000	24.814
50	scaled 4.0	5%	edge-based	0.90	gaussian	0.10	1.000	27.812
50	scaled 4.0	5%	edge-based	0.90	uniform	0.10	1.000	28.892
50	scaled 4.0	5%	edge-based	0.90	gaussian	0.20	1.000	30.190

Table 18: Top five predicted adaption combinations for order-based GCAs on Towns2.

2.3.3 Performance of Selected Adaptions

Tables 19 and 20 contain summaries of the performance of the top predictions for each model on the generated and real data sets. For the purposes of this experiment, the correct clustering for each of the real data sets was assumed to be the clustering with the minimum objective function value (thus the correct clustering for the Iris data set was not its real clustering, since the real clustering includes overlapping clusters).

The group-number results on both the real and generated Ruspini data match the predictions well, as do the correctness results for the Iris data. However, the group-number GCA was unable to consistently find the correct clustering for the German Towns data set despite good results in the generated data set. Although the mean time to solution for the GCAs on the real and generated Iris data was significantly higher than the predicted value, it was within an order of magnitude in each case.

	Predicted		Generated Data			Real Data		
	Corr.	Time	Corr.	Time		Corr.	Time	
Ruspini	40.000	42.954	39	42.130	(4.523)	38	43.882	(5.942)
German Towns	33.058	109.291	37	120.593	(50.638)	11	187.283*	(28.374)
Iris	40.000	6.678	40	7.827*	(0.9225)	40	11.901*	(2.403)

Table 19: Performance of the top group-number GCAs on generated and real data sets. Corr. is the number of correct runs out of 40. For the real data sets the correct clustering was assumed to correspond to the minimum objective function value. Time is the average time to solution in seconds CPU for the correct runs, the standard deviations are included in brackets. * = significantly different from predicted value at 1% level.

The the correctness of the order-based GCA on the real Ruspini data matched that on the generated data and the prediction, although the time to find the solution was significantly higher in both cases. The order-based GCA for the German Towns data set, matched the prediction on the generated data set but was unable to find a single solution for the real data.

	Predicted		Generated Data			Real Data		
	Corr.	Time	Corr.	Time		Corr.	Time	
Ruspini	39.999	57.317	40	66.482*	(18.770)	40	110.498*	(67.870)
German Towns	40.000	23.888	40	29.010	(27.920)	0	-	

Table 20: Performance of the top order-based GCAs on generated and real data sets. Corr. is the number of correct runs out of 40. For the real data sets the correct clustering was assumed to correspond to the minimum objective function value. Time is the average time to solution in seconds CPU for the correct runs, the standard deviations are included in brackets. * = significantly different from predicted value at 1% level.

Figure 26 shows the evolution of a successful group-number GCA's population over 200 generations for the Ruspini data set. The initial population (Generation 0) has high raw fitness (or objective function) values, which indicate that these clusterings do not suit the Ruspini data. As the number of generations increases, the raw fitness of clusterings within the population decreases, as the GCA is biased towards the survival of genetic material contained within the clusterings with low objective function values. For this data set, the minimum objective function value is 10.088 which corresponds to the correct clustering for the data set. This GCA found the correct clustering

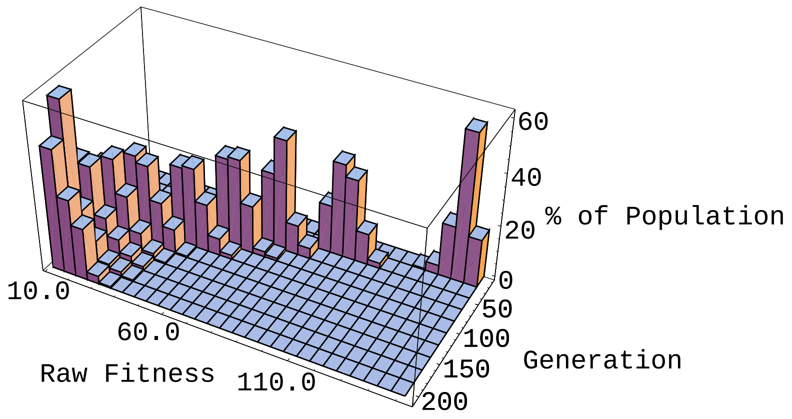


Figure 26: Fitness distribution of 200 Generations[1] of a successful group-number GCA on Ruspini. PS = 200, Tr = scaled 4.0, El = 5%, Cr = edge-based, CP = 0.50, MR = 0.50. [1]sampled every 20 generations.

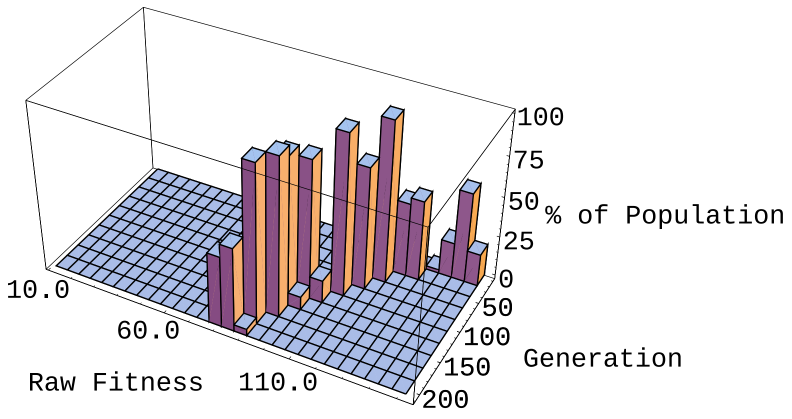


Figure 27: Fitness distribution of 200 Generations[1] of a unsuccessful group-number GCA on Ruspini. PS = 50, Tr = local, El = 0, Cr = single-point, CP = 0.50, MR = 0.05. [1]sampled every 20 generations.

in the 140th generation, and this clustering made up 39% of the final population.

Figure 27 shows the evolution of an unsuccessful GCA for the Ruspini data. Although the fitness of the population is decreasing over the generations, the evolution is not fast enough for the CGA to find the correct clustering within the 200 generations.

2.4 Discussion

The overall correctness results for the generated data (Tables 3 and 4) raise some interesting questions — why did the group-number GCAs perform so well on Ruspini2 and Iris2, yet poorly on Towns2? and why did the order-based GCAs fail to find a single solution for Iris2? The answer to the first question appears to be the high value of k for the Towns2 data set; whereas the large number of objects in the Iris2 data set may provide the answer for the second question.

A group-number representation for a data set with a large number of clusters has a high level of redundancy (each clustering can be represented by $k!$ different chromosomes). This means that the representation space has many more optima than the solution space. This divides the focus of the population, and crosses between chromosomes near differing optima tend to be unproductive and wasteful. Overall, redundancy slows down the convergence and reduces the diversity of the population.

Since edge-based crossover is context sensitive (and therefore not affected by redundancy) we expect the GCAs with this crossover to out-perform the other GCAs on this data set. In fact, a high proportion of the trials with 5 correct runs did use edge-based crossover. However, there were also a large number of edge-based GCAs that did not find a single solution. This is due to the high complexity of the edge-based operator, $O(k^4)$, combined with large population sizes which greatly increases the time required to produce a new generation. Five processor minutes was simply not enough time for these GCAs to find the solution.

The poor performance of the order-based GCAs on Iris2 can be explained by the large number of objects in the data set, which affects the speed of the local search (due to its complexity of $O(n^2k)$) and the size of the representation space (see Table 21). This meant that the time limit of 300 seconds restricted the GCA to only a few generations, limiting the GCAs' search to a very small portion of a very large representation space.

			Size of Space		
Data Set	n	k	Solution	Group-number Rep.[1]	Order-based Rep.[2]
Ruspini2	75	4	6×10^{43}	1×10^{45}	7×10^{112}
Towns2	59	7	1×10^{46}	7×10^{49}	2×10^{83}
Iris2	150	3	6×10^{70}	3×10^{71}	6×10^{266}

Table 21: Size of the representation space for generated data sets. [1]Size of group-number representation space is k^n. [2]Size of order-based representation space is effectively $\frac{1}{2}n(n+1)!$ (the local heuristic searches through $\frac{1}{2}n(n+1)$ clusterings for each chromosome, and there are $n!$ possible chromosomes).

When considering the results of the adaptions tests, it is important to remember that these tests were limited to 300 seconds. Thus the correctness models that have been presented feature adaptions that increase the probability of finding the correct clustering in under five minutes.

The adaption tests for the data sets resulted in a different model for each. This suggests that the type of data set should be an important consideration when selecting or designing a suitable GCA.

- **Population Size**

  Increasing the population size increases the size of the GCA's gene pool which is advantageous for group-number GCAs that are searching for clusterings with high k (for Ruspini and Iris2 GCAs with the smallest population size were not as successful at finding the correct solution as the GCAs with larger populations). The local search used for order-based encoding means that a relatively small population can span a considerable portion of the representation space (the results do not suggest that increasing the population size will improve performance). However, increasing the population size means that each reproductive phase takes longer and therefore less generations occur in the same time. This is important because it is the reproductive phase that evolves the population. It will also take longer to converge a larger pool of genes.

- **Fitness Transformation**
  Fitness transformations can allow certain chromosomes to dominate the population by giving them high relative fitness values. For smaller populations and larger solution spaces scaling is important as it increases the probability that the genes of fitter solutions remain in the population. However, too high a level of scaling can quickly converge the population to sub-optimal solutions. The results suggest that linear scaling (with C_{Mult} of 2.0 or 4.0) is appropriate for data sets used in the experiments. Some time advantage may be gained by using the adjusted transformation on data sets with small k.

- **Elite Constant**
  Elite selection ensures that good genes are not lost from the population. However, elite selection also decreases the size of the population that is actively searching for new solutions. The results suggest that elite selection is useful for decreasing the time taken to find the solution for group-number GCAs, and it does appear to improve the correctness for the larger data sets. However performance of the order-based GCAs was worse with elite selection, possibly because it reduced the size of the active population.

- **Crossover Type**
  The crossover type had a large influence on the correctness and time results for both the group-number and order-based GCAs. Single-point crossover was clearly sufficient for the data set with low k; this was also the fastest crossover method for the group-number GCAs. However, group-number GCAs with the slower edge-based crossover found more correct solutions on Ruspini2 and Towns2. Perhaps a context sensitive crossover is more important for data sets with higher k. With the order-based GCAs, the best performing crossover was edge-based which found the solution in less generations than borrow or PMX. Due to the time complexity of the local search, and consequently the reduced number of generations, it was important that the crossover operator combined genetic material effectively during reproduction.

- **Crossover Probability**
  The crossover probability had little effect on the performance of the group-number GCAs, although increasing the value tended to reduce the time to find the solution. The order-based GCAs also performed well with high crossover probabilities.

- **Mutation Type**
  There was only a small time difference between the two mutation types used for the order-based GCAs. There does not seem to be any reason to favour one operator over the other.

- **Mutation Rate**
  The largest affect of changing the mutation rate is seen in the time to solution. However, a certain level is necessary to introduce new genetic material to the population (note the generally poor performance of all GCAs with very low mutation). Too high a mutation rate

will also degrade the performance of a GCA by reducing the GCA's evolution to a random search.

The models produced from the results of adaption experiments all contain interaction terms, which means the combination of some adaptions will be more effective than others. Hence it is necessary to select a total set of adaptions carefully.

There appears to be advantages to using both clustering representations. Group-number encoding when combined with single-point crossover is a fast method of finding clusterings for data sets with low k. This method also seems to be able to cope with relatively large data sets, although size of the data sets used in these experiments was limited. The performance of the group-number GCAs on data sets with high k values was poorer, with many CGAs returning sub-optimal minima.

The performance of the order-based GCAs with edge-based crossover on the first two generated data sets was excellent. The time to find the solution for Ruspini2 was comparable with the group-number GCAs, whereas the time to find the solution for Towns2 was considerably less. However, this method was unable to find a single solution for the data set with the most objects. As a result of the complexity of the local search, this method cannot cope with large data sets, but is suitable for small data sets with large values of k.

The comparison of GCA performance on the generated and real data sets raises some important points. For both the group-number and order-based GCAs, the results on the Ruspini data sets matched the predictions very well (the fit of the time model for the order-based GCA was questionable, so we do not expect the time results to match too well). The correctness results for the other GCAs on the generated data sets were close to the predicted values (the group-number correctness model for Towns2 also had questionable fit), and the group-number correctness for the Iris data was equal to the predicted value. However, both the group-number and the order-based GCAs performed poorly on the German Towns data set.

The fact that the best adaptions for each data set were clearly different suggests that GAs should be modified in order to attain optimal performance for a particular data set. However, the results for the generated and real data sets were similar for Ruspini and Iris, indicating that it may be possible to adapt GAs to suit a class of data sets (with approximately the same number of objects, clusters, and distribution). The contrasting performance of the GCAs on the German Towns data set may be a result of the low similarity between the distribution of the generated and real objects. The generated data set is clearly separable into seven clusters, whereas the towns in the real data set do not fall into distinct clusters. For the real data this means there will be a large number of clusterings with objective function values close to the optimum value. Clearly, such a data set is harder for a GCA to cluster, and the chances of finding a local minima rather than the optimal solution are greatly increased. Increasing the rate of mutation in the early generations (to increase the available genetic material during the initial stages), may improve performance in this case.

Although the results presented in this chapter suggest that differing data sets require different adaptions for optimal performance (correctness and speed), this does not mean that a particular set of adaptions will not find the correct solution to a large number of data sets if given enough time. Indeed, it would seem that a number of adaptions (elite constant, fitness transformation, crossover probability, and mutation rate) have a large effect on the time taken to find the optimal solution, and that for simpler data sets these adaptions can be used to speed up the evolution of the GCA without sacrificing correctness. So although it seems important to invest some time in adapting the GA to suit the data set, it may not be necessary to find the optimal adaptions to find correct clusterings within a reasonable time.

Finally, the adaptions implemented in this experiment are only a sample of the numerous possible adaptions. Thus the performance of GCAs on these and other data sets may be improved by the investigation of further adaptions.

Chapter 3

Genetic versus Traditional Clustering Algorithms

In the last chapter we described how GAs can be adapted for the problem of clustering, and compared the effectiveness of some selected adaptions. In this chapter we compare these adapted GAs with three hierarchical clustering methods (SLINK, average-linkage, and Ward's method) and four optimisation clustering methods (k-means, iterated nearest neighbour, iterated hillclimber, and simulated annealing) in terms of correctness, consistency, and speed.

3.1 Clustering Algorithms

Traditional clustering methods were broadly described in Chapter 1. In this section we describe those algorithms selected for this experiment in further detail.

SLINK

Hierarchical clustering techniques use a series of successive fusions or divisions to construct a hierarchy of clusterings. The single-linkage method [20] starts with each object in its own cluster. Clusters are then merged according to the distance between their nearest members until only one cluster remains.

A simple implementation of the single-linkage algorithm involves calculating and storing the $n \times n$ distance matrix, $D = \{d_{ij}\}$ where d_{ij} is the Euclidean distance between the closest objects in Cluster i and Cluster j. At each stage the clusters with the minimum distance are merged, and the distance matrix is updated by: (i) deleting the rows and columns corresponding to the merged clusters; and (ii) adding a row and column giving distances between the new cluster and the remaining clusters.

SLINK is an optimally efficient algorithm for the single-linkage clustering method developed by Sibson [57] (Figure 28). Instead of storing a matrix of distances that is referred to and updated numerous times during the clustering process, the SLINK algorithm only reads (or calculates) each row of the initial distance matrix once.

```
1. Set $\Pi(n+1)$ to $n+1$, $\Lambda(n+1)$ to $\infty$
2. Set $M(i)$ to $d(i, n+1)$ for $i = 1, ..., n$
3. For $i$ increasing from 1 to $n$
        if $\Lambda(i) \geq M(i)$
                set $M(\Pi(i))$ to min $\{M(\Pi(i)), \Lambda(i)\}$
                set $\Lambda(i)$ to $M(i)$
                set $\Pi(i)$ to $n+1$
        if $\Lambda(i) < M(i)$
                set $M(\Pi(i))$ to min $\{M(\Pi(i)), M(i)\}$
4. For $i$ increasing from 1 to $n$
        if $\Lambda(i) \geq \Lambda(\Pi(i))$
                set $\Pi(i)$ to $n+1$
```

Figure 28: The SLINK algorithm (from [57]).

The SLINK algorithm manipulates three arrays of size n. The first, Λ, is used to store the levels at which the clusters are merged — $\lambda(i)$ is the lowest level at which i is no longer the last object in its cluster. The second array, Π, stores the merging sequence — $\pi(i)$ is the last object in the cluster which object i then joins. The final array, M, is used for the distance values ($d(i, j)$ is the Euclidean distance between objects i and j). The implemented algorithm starts by calculating a row of the lower triangular distance matrix and placing the values in M. These values are compared against those in Λ, and the values in Λ and Π are adjusted accordingly. The values of Λ are then compared against each other and any necessary adjustments made. This is repeated for each row of the distance matrix. A final step involves decoding the resulting clustering from Λ and Π.

Average-linkage

The average-linkage method [58] merges the clusters with the minimum average distance between all pairs of objects where one member of a pair belongs to each cluster. Hence the distance between two clusters, A and B, was calculated as

$$\frac{1}{n_A n_B} \sum_{i=1}^{n_A} \sum_{j=1}^{n_B} d(X_{Ai}, X_{Bj}),$$

where n_A, n_B are the number of objects in clusters A and B, respectively; and $d(X_{Ai}, X_{Bj})$ is the Euclidean distance between the ith object in A and the jth object in B. If there were more than one pair of clusters with the minimum distance between them, the pair to be merged was randomly selected from those with the minimum distance.

Since the distance measure was based on the Euclidean distance between pairs of objects, two distance matrices were stored. The first contained the Euclidean distances between all pairs of objects, and was used to calculate the distances between clusters. The second matrix stored the distance between the existing clusters (initially this is the same as the first matrix), and was updated during the clustering process. Only the lower triangular portion of the matrices were stored since they were both symmetric.

The clustering was stored as an array of n clusters, where each cluster was represented by a list of object numbers and the number of objects stored in this list. An array of cluster numbers was used as an index for the cluster array — the number of values stored in this array varied from n to 1 during the clustering process. When two clusters, say A and B were selected for merging, the data structure was updated as follows:

1. The objects in B were added to A's object list. The number of objects in A was incremented by the number of objects in B.
2. B was removed from the list of cluster numbers.
3. The distance matrix was updated by removing the rows and columns for B and recalculating those for A (now AB).

Ward's Method

The implementation of Ward's method was similar to that of average-linkage, but in this case the distance between two clusters was calculated as

$$\frac{2\,n_A n_B}{n_A + n_B} d^2(\overline{X}_A, \overline{X}_B),$$

where n_A, n_B are the number of objects in clusters A and B, respectively; and $\overline{X}_A$ and $\overline{X}_B$ are the centroids of the two clusters. This distance is a simplification of Ward's sum of squared distances statistic [39].

Since the distance between two clusters was based on the Euclidean distance between their centroids, a single distance matrix was sufficient. As with the average-linkage implementation, the clustering was stored as an array of clusters. However, each cluster was represented by its *centroid* (stored as a p-dimensional vector) and its number of objects.

A final difference was the method of updating the clustering structure during merging (Step 1, from above). For Ward's method, the centroid of Cluster A was replaced by the centroid of Cluster AB,

$$\overline{X}_{AB} = \frac{n_A \overline{X}_A + n_B \overline{X}_B}{n_A + n_B}.$$

The number of objects in A was then incremented by the number of objects in B. The remaining updating steps were the same as those for average-linkage.

k-means

Optimisation clustering techniques use a search method to find a clustering that optimises a pre-defined objective function. These techniques start with an initial clustering (containing the correct number of groups), and then reassign objects according to the objective function until some terminating criterion is met.

The objective function used for the optimisation clustering methods in this experiment was the sum of the squared Euclidean distance between objects and their cluster centroids. This criterion is equivalent to Trace (W), and is minimised over the representation space (Section 1.1.3). Using the same objective function for the optimisation algorithms means we are comparing the effectiveness of the search methods when we compare these algorithms. Further, this criterion is similar to the distance measures used in the above hierarchical methods.

The k-means algorithm was implemented as described by Hartigan [29]. The algorithm starts by generating a random clustering, which is the starting point for a local search that takes each object in sequence and moves it to the cluster that reduces the objective function value the most. If the objective function value is not reduced, the object remains in its current cluster; the algorithm terminates when the objective function value cannot be reduced by any move.

The current clustering for this algorithm was stored in an array using the group-number representation described in Section 2.1.1. The change in objective function value associated with transferring Object i from its cluster, A, to Cluster B, was calculated as

$$\frac{n_B\, d^2(X_i, \overline{X}_B)}{n_B + 1} - \frac{n_A\, d^2(X_i, \overline{X}_A)}{n_A - 1},$$

where n_A, n_B are the number of objects in clusters A and B, and $d^2(X_i, \overline{X}_j)$ is the squared Euclidean distance between Object i and the centroid of Cluster j [29]. Here, a negative value indicates a reduction in the objective function value. An array of clusters was used to store the centroid of, and the number of objects in, each cluster — the same structure as used in the implementation of Ward's method. However, since the current clustering always has k clusters, there were only k clusters in the data structure, and there was no need to keep track of cluster numbers.

Moving an object simply required changing the object's group-number in the current clustering, and updating the two relevant clusters — the cluster the object was moving from, and the cluster it was moving to.

Iterated Nearest Neighbour

The iterated nearest neighbour algorithm is adapted from the greedy permutation representation discussed in Section 2.1.1. GAs with this representation tend to find their optimal solution in the initial generation [36], which suggests that an iterated version of this local search would make an appropriate clustering algorithm.

The algorithm starts by generating a random permutation of the objects. The first k objects become seed points and each is assigned to a separate cluster. The remaining objects are, in the order they occur in the permutation, added to the cluster with the closest centroid (measured as the squared Euclidean distance between the object and the cluster centroid). These steps are repeated a number of times (determined by the user) and the clustering with the minimum objective function value is returned as the solution.

In general, if the seed points correspond to the clusters in the data set (that is, there is a seed point in each cluster), and the clusters are well separated, this algorithm will find the clustering that corresponds to the minimum sum of squared Euclidean distance between objects and their cluster centres. For more complicated data sets, only seed points close to the cluster centroids will lead to the optimal clustering. In either case, the probability of finding the solution is increased by increasing the number of iterations, as this improves the chances of finding good seed points.

The data structure for this implementation was the same as that for k-means. However, a second array was necessary to store the best clustering found during the iterations. Adding an object to a cluster involved setting the object's group-number in the clustering, updating the cluster's centroid, and incrementing the number of objects in the cluster.

Iterated Hillclimber

The hill-climbing or steepest descent algorithm starts from a random clustering and searches a local neighbourhood for the clustering which reduces the objective function value the most. If such a clustering is found, it becomes the current clustering and the algorithm then searches the neighbourhood of this clustering; otherwise the algorithm terminates. The success or failure of this algorithm is determined by the starting string, and the chances of finding the global optimum for a problem with many local optima are slim. An iterated version of the hill-climbing algorithm can be found in [48]. This simply repeats the normal hill-climbing algorithm a set number of times and saves the clustering with the minimum objective function value.

For this experiment, the local neighbourhood of a given clustering was considered to include the clusterings in which all the objects were in the same groups, except for a single object whose group-number differed by 1. For example, given a clustering with Object 1 in Cluster 4, we can create two neighbours by moving Object 1 to Cluster 3 and Cluster 5 respectively. This definition means that a clustering in a data set with n objects has 2^n neighbours.

The current clustering for the hillclimbing algorithm was stored using group-number representation, and the best clustering for the iterations was saved in a similar manner. The cluster centroids were re-calculated during each evaluation of a clustering.

Simulated Annealing

Simulated annealing [40] is a method of function optimisation that is modelled on the annealing process used in glass blowing and metallurgy. The annealing process involves heating the glass or metal to a temperature just below its melting point, and then allowing it to cool slowly. During the cooling stage the molecules in the material re-align themselves and crystalise, which reduces the internal stresses resulting from working the material.

In simulated annealing, the search is controlled by a parameter called "temperature", which occasionally allows the search to move uphill (and thus escape local optima). Moves that decrease the objective function value are always accepted; while moves that increase this value are accepted with a probability based on the temperature, and the change in objective function value. The temperature value — initially high — is lowered in stages to mimic the annealing process, and the probability of accepting an uphill move decreases with decreasing temperature.

Figure 29 contains an outline of the implemented simulated annealing algorithm. This is similar to the algorithm described in [48]; a more complex implementation for the clustering problem can be found in [41].

The algorithm contains two loops. The outer loop controls the temperature, which starts at T_0 and is successively decreased by a (cooling) factor of T_δ. The algorithm terminates when the temperature falls below the final temperature, T_F.

The inner loop, which is repeated T_{steps} times for this implementation, randomly generates a new clusterings by selecting a new group for an object. The new clustering is accepted as the

```
Initialise the temperature $T$ to $T_0$
Select a current clustering $C_c$ at random and calculate its objective function value $f(C_c)$
repeat
    for $T_{steps}$ do
        Select a new clustering $C_n$ by randomly selecting a new group for an object
        If $f(C_n) < f(C_c)$ then
            $C_n$ becomes the current clustering $C_c$
        else
            $C_n$ becomes the current clustering $C_c$ with probability $\exp(\frac{f(C_c)-f(C_n)}{T})$
    $T = T_\delta \times T$
until $T < T_F$
```

Figure 29: Clustering with simulated annealing.

current clustering: (1) if its objective function value is lower than that of the current clustering; or (2) with a probability based on the temperature, and the difference in the objective function values between the new and current clusterings.

The current and new clusterings were stored using group-number representation, and the cluster centroids were re-calculated for each clustering.

Genetic Clustering

The GCAs used for this experiment have two important differences to those developed in the previous chapter. Firstly, exponential mutation has been added to the GCAs for the Ruspini and German Towns data sets. The mutation rate is high for the initial generations and then decays exponentially with increasing generations. This increases the amount of new genetic material introduced in the early generations and reduces the chances of mutation destroying good solutions in later generations.

Exponential mutation is described by two mutation rates — the initial and final rate. Each mutation rate is expressed as the probability of mutating a chromosome, that is mutating a single gene on that chromosome. The mutation rate for a given generation, *gen*, is calculated as

$$MR_{gen} = MR_F \times \exp\left[\ln\left(\frac{MR_0}{MR_F}\right)\left(1 - \frac{gen}{MG}\right)\right]$$

where MR_0 and MR_F are the initial and final mutation rates, respectively; and MG is the maximum number of generations.

Secondly, the GCAs here are run for a set number of generations rather than for a set time interval as in the previous chapter. The number of generations to use in each case was calculated from the performance of the GCAs on the real data sets in the last chapter (the generation that the correct solution was found was recorded, and the average and standard deviation of these values were used to calculate a 95% confidence interval that was used as a guide for selecting an appropriate number of generations).

As in the previous chapter, each GCA is described by a list of parameters: PS (population size), MG (maximum number of generations), Tr (scaling transformation), El (elite constant), Cr (crossover type), CP (crossover probability), and MR (mutation rate(s)).

3.2 Methodology

All of the described algorithms were implemented by the author. The algorithms were coded in C and compiled using gcc with the optimisation flag set to level 3. The clustering algorithms were compared on the three data sets introduced in the previous chapter — Ruspini, German Towns, and Iris (see Section 2.2.1). Each data set was clustered 40 times by each algorithm. The following information was collected: (i) the number of runs that found the correct solution, (ii) the average and standard deviation of the correctness over the 40 runs, (iii) the consistency of the 40 solutions, and (iv) the average and standard deviation of the time taken by the algorithm.

The correctness was measured as the maximum percentage of objects that were clustered in the same groups as the correct clustering (there are $k!$ ways that the groups can be matched since the group-numbers do not necessarily correspond). For example, if the correct clustering was $\{\{X_1, X_3, X_6\}, \{X_2, X_4, X_5\}\}$, the clustering $\{\{X_2, X_3, X_4, X_5\}, \{X_1, X_6\}\}$ has a correctness of 83.3% (five out of six objects). This matches the first group in the correct clustering with the second group in the other clustering — the alternate matching has only one object correctly clustered.

A run was considered to be correct only if there was a perfect match between the clustering found by the algorithm and the correct clustering (that is, a correctness of 100.0%). For this experiment the correct clustering for each of the data sets was assumed to be the clustering that corresponded to the minimum sum of squared distances between the objects and their cluster centres. Consistency was calculated as the maximum percentage of objects that were clustered in the same clusters for all 40 runs (groups were matched in a similar manner to the correctness calculations). The time taken by the algorithm was measured in seconds CPU (resolution of the clock was 16.667 milliseconds), and did not include the time taken to store the objects' attribute values. In order to remain consistent with the previous chapter, the objects' attribute values were standardised so that each attribute had a mean of 0 and a standard deviation of 1. The tests were conducted on a single Sparc Ultra; the load average during testing was approximately 0.9.

The number of iterations for both the iterated nearest neighbour and iterated hillclimber algorithm were varied until the performance of these algorithms exceeded that of the GCA (either in terms of correctness, or time). The temperature parameters for the simulated annealing algorithm were adapted in a similar manner.

The mean correctness and time of the algorithms was compared against that of the GCA using one-sided z tests (null hypothesis of equal mean values), which were appropriate due to the large number of runs.

3.3 Results

Tables 22 to 24 contain the clustering results for the three data sets. All of the clustering algorithms, except for k-means, found the correct solution for the Ruspini data set in every run. SLINK was the fastest of the algorithms that found the correct clustering in all runs. The time for the iterated hillclimber was closest to that of the GCA, but all of the comparison algorithms were significantly faster than the GCA.

Algorithm	Correctness			Consistency	Time	
	Runs	%	std dev.	%	sec. CPU	std dev.
SLINK	40	100.0	0.0	100.0	0.004†	0.007
average linkage	40	100.0	0.0	100.0	0.016†	0.006
Ward's method	40	100.0	0.0	100.0	0.022†	0.008
k-means	35	95.8	11.2	66.7	0.002†	0.005
iterated nearest neighbour[1]	40	100.0	0.0	100.0	0.054†	0.007
iterated hillclimber[1]	40	100.0	0.0	100.0	46.808†	1.098
simulated annealing[2]	40	100.0	0.0	100.0	3.009†	0.022
GCA[3]	40	100.0	0.0	100.0	51.707	1.996

Table 22: Comparison of k-clustering algorithms on Ruspini. [1]100 iterations. [2]$T_0 = 10.0, T_{steps} = 200, T_\delta = 0.9$, and $T_F = 0.000001$. [3]PS = 200, MG = 200, Tr = scaled 4.0, El = 10, Cr = edge-based, CP = 0.50, MR = 0.90 – 0.10. †significantly faster than the GCA at 1% level.

Algorithm	Correctness			Consistency	Time	
	Runs	%	std dev.	%	sec. CPU	std dev.
SLINK	0	57.6	0.0	100.0	0.001†	0.004
average linkage	0	79.7	0.0	100.0	0.008†	0.008
Ward's method	0	94.9*	0.0	100.0	0.010†	0.008
k-means	9	85.5	9.9	44.1	0.004†	0.007
iterated nearest neighbour[1]	31	98.6*	4.0	79.7	4.870†	0.025
iterated hillclimber[2]	0	59.4	6.5	5.1	271.910†	4.767
simulated annealing[3]	15	90.1	7.8	76.3	249.229†	2.505
GCA[4]	14	86.4	12.2	45.8	284.695	10.269

Table 23: Comparison of k-clustering algorithms on German Towns. [1]10000 iterations. [2]1000 iterations. [3]$T_0 = 10.0, T_{steps} = 200, T_\delta = 0.999$, and $T_F = 0.00001$. [3]PS = 100, MG = 1500, Tr = scaled 2.0, El = 1, Cr = edge-based, CP = 0.70, MR = 0.90 – 0.10. *significantly more correct than the GCA at 1% level. †significantly faster than the GCA at 1% level.

Algorithm	Correctness			Consistency	Time	
	Runs	%	std dev.	%	sec. CPU	std dev.
SLINK	0	68.0	0.0	100.0	0.014†	0.006
average linkage	0	68.0	0.0	100.0	0.108†	0.008
Ward's method	0	80.0	0.0	100.0	0.165†	0.010
k-means	40	100.0	0.0	100.0	0.004†	0.007
iterated nearest neighbour[1]	40	100.0	0.0	100.0	1.415†	0.022
iterated hillclimber[2]	40	100.0	0.0	100.0	71.600	2.824
simulated annealing[3]	40	100.0	0.0	100.0	2.735†	0.015
GCA[4]	40	100.0	0.0	100.0	38.633	2.515

Table 24: Comparison of k-clustering algorithms on Iris. [1]1000 iterations. [2]10 iterations. [3]$T_0 = 1.0, T_{steps} = 100, T_\delta = 0.9$, and $T_F = 0.00001$. [3]PS = 50, MG = 1000, Tr = adjusted, El = 3, Cr = single-point, CP = 0.90, MR = 0.70. †significantly faster than the GCA at 1% level.

None of the hierarchical methods found the correct clustering for German Towns. However, the mean correctness of Ward's method was significantly higher than that of the GCA, and all of the hierarchical methods were significantly faster. Only two algorithms, iterated nearest neighbour and simulated annealing, found the solution for the German Towns data set more times than the GCA. Both of these algorithms were significantly faster than the GCA, and iterated nearest neighbour was also significantly more correct.

None of the hierarchical methods found correct solution for the Iris data set, but all of the optimisation methods, including the GCA, found the clustering with the minimum objective function value on every run. All of the algorithms, except for the iterated hillclimber, were significantly faster than the GCA.

3.4 Discussion

All clustering methods search a restricted subset of the solution space. Hierarchical methods tend to search a smaller subset than the optimisation methods. Thus the search is faster and the results are more consistent, though the constraints of the hierarchical techniques sometimes prevent the algorithms from finding the optimal solution (for example, the hierarchical methods were unable to find the correct clustering for either the German Towns or Iris data sets).

The performance of optimisation methods is limited by two factors: the suitability of the objective function, and the effectiveness of the search method. Optimisation clustering methods search for the clustering that corresponds to the minimum (or maximum) objective function value, which may or may not be the correct clustering. For example, all of the optimisation methods found the clustering with the minimum objective function value for the Iris data set, but this is not the real clustering of this data set (the clustering defined as the correct clustering for this experiment has a correctness of 83.3% when compared to the real clustering). For data sets such as Iris, with over-lapping clusters, it may be impossible to define an objective function that can identify the correct solution. However, the selected objective function was sufficient for the optimisation methods to find a more correct solution than the hierarchical algorithms.

Even if the objective function can identify the correct solution, there is no guarantee that the search method will find it. Consider the performance of the optimisation algorithms on the German Towns data. By definition, the correct clustering was the clustering with the minimum objective function value, yet the iterated nearest neighbour algorithm did not find this clustering on nine runs, and the iterated hillclimber was unable to find it on any run. Clearly, some of the search methods are more effective than others for certain data sets.

Optimisation methods such as k-means and iterated nearest neighbour search a relatively small subset of the solution space. These methods (in particular the iterated nearest neighbour method) are well suited to the objective function used for these experiments, and the subset they search contains solutions with high correctness. Hence these algorithms produce fast results with relatively high correctness.

The three remaining optimisation methods search a larger subset of the solution space, thus these algorithms are the slowest. For the iterated hillclimber and simulated annealing, this subset is determined by the definition of the neighbourhood, and the parameter values. The iterated hillclimber did not perform well on the German Towns data set, perhaps due to the large number of local optima for this data set. However, altering the neighbourhood definition may improve this algorithm's performance.

Overall, we are looking for a method that consistently produces correct clusterings in a relatively short amount of time for a wide range of data sets, where correctness is most important. Hence the GCA was clearly out-performed by two other clustering methods: iterated nearest neighbour and simulated annealing. This suggests that there is no advantage to using genetic clustering for the k-clustering problem for data sets with similar characteristics to those used in our experiment. The addition of further adaptions to the GCA may enhance its performance, but it is improbable that such improvements would allow the algorithm to match the performance of the iterated nearest neighbour algorithm.

We also expect that the iterated nearest neighbour clustering algorithm would continue to outperform the GCA on data sets with more objects and a larger number of clusters. However, the described iterated nearest neighbour algorithm is based around the given objective function. It may not be possible to use a similar method for other objective functions. Thus a GCA with a different objective function may outperform this method for certain data sets.

The trend across the three data sets indicates that the GCA may outperform simulated annealing on data sets with large numbers of clusters.

Thus, although the GCA can match or better the correctness results for all of the algorithms (except for iterated nearest neighbour) on the given data sets, it is also significantly slower than these clustering algorithms.

Chapter 4

Genetic Clustering for Unknown k

Thus far we have concentrated on the k-clustering problem, where the number of clusters is known. Here we extend our GCA to deal with the more general clustering problem, where the optimal number of clusters is determined as part of the clustering process.

4.1 Background

Determining the optimal number of clusters for a data set is one of the more difficult aspects to the clustering process. Most optimisation clustering methods require the user to specify the number of clusters, and hierarchical methods typically produce a series of clusterings from 1 to n clusters without specifying the most appropriate number of clusters.

Numerous procedures have been suggested for determining the optimal number of clusters [10, 11, 49, 15], including some criteria appropriate for use as the objective function for optimisation clustering methods. For these criteria, the clustering that results in the minimum (or maximum) value of the objective function should be the best possible clustering (with the optimal number of clusters) for the data set.

4.1.1 Criteria for Determining the Number of Clusters

Milligan and Cooper [49] compare 30 criteria for determining the number of clusters. The criteria were used in conjunction with four hierarchical clustering methods to determine the best number of clusters for artificial data sets with distinct non-overlapping clusters. The method suggested by Calinski and Harabasz [10],

$$\frac{(n-k) \times trace(B)}{(k-1) \times trace(W)},$$

produced the best results for the experiment (B and W are defined in Section 1.1.3). The optimal clustering is indicated by the maximum value of this measure.

The cluster separation method suggested by Davies and Bouldin [11, 33] also performed reasonably well. This criterion is based on the minimum ratio of within-cluster dispersions, S_i, and between-cluster separation, $T_{i,j}$:

$$W_k = \frac{1}{k}\sum_{i=1}^{k} R_i$$

where

$$R_i = \max_{j,j\neq i} \frac{S_i + S_j}{T_{i,j}}$$

$$S_i = \left[\frac{1}{n_i}\sum_{j=1}^{n_i} |x_{ij} - \overline{x}_i|^s\right]^{\frac{1}{s}}$$

$$T_{i,j} = \left[\sum_{l=1}^{p} |\overline{x}_i^l - \overline{x}_j^l|^t\right]^{\frac{1}{t}}$$

and the values of s and t are user-defined (although Davies and Bouldin offer some guidance for selecting appropriate values). The criterion value should be minimised over the solution space.

Further discussion of criteria for determining the number of clusters can be found in [63, 15]. Criteria for use with fuzzy clustering methods is described in [5, 22, 42].

4.1.2 Adaptions for the General Clustering Problem

It is possible to adapt the group-number GCAs we developed in Chapter 2 by simply exchanging the fitness function for a criterion appropriate for the general clustering problem. The k parameter now becomes an upper limit on the number of clusters, $maxk$ (which might even be n), rather than the required number of clusters. The group-number representation allows clusters to have anywhere between 1 and $maxk$ groups (missing group numbers indicate less than $maxk$ groups). The uniform mutation operator will randomly move objects between groups (perhaps even introducing a new group), while single-point and uniform crossover recombine group-numbers during crossover (edge-based crossover is not appropriate because it always produces offspring with k clusters). However, this is not necessarily the best or the only approach.

Representation

While the number of clusters is implicit in the group-number representation, this is not the case for all representations. The matrix representation can be extended in a similar manner to group-number, by increasing the number of rows to $maxk$. Adjacent separators in the permutation with separators encoding scheme could be used to indicate empty groups if $maxk - 1$ separators were included in each chromosome. However, both the greedy permutation and order-based representations require some independent means of storing the number of clusters for each chromosome. Adding an extra gene to store this value is one possible method.

Falkenauer [17, 18] describes an encoding scheme specifically designed for grouping GAs. Under this scheme each chromosome consists of two parts: an *object* part and a *group* part. The group part is exactly the same as a standard group-number encoding, and the object part is simply a list of the group-numbers that occur in the object part. Figure 30 contains Falkenauer's encoding of the clustering $\{\{X_1, X_3, X_6\}, \{X_2, X_4, X_5\}\}$. Since the number of groups in the object part can vary, so can the length of the chromosome. The genetic operators work with the group part of the representation, thus the operators are manipulating groups rather than objects.

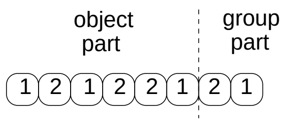


Figure 30: Chromosome representing the clustering $\{\{X_1, X_3, X_6\}, \{X_2, X_4, X_5\}\}$ for Falkenauer's encoding scheme.

Initialisation

For the k-clustering problem we initialised the group-number population by generating strings of random numbers between 1 and k. For the general clustering problem we could use the same method but increase the upper limit of the group-numbers to $maxk$. However, when n is much larger than $maxk$ most of the population would contain $maxk$ groups.

The initial population should provide a random selection of strings from the representation space, thus there should be approximately the same number of chromosomes with each possible value of k. Such a population could be generated by creating chromosomes as follows:

1. Randomly select the number of clusters, k, from [1,$maxk$].
2. Randomly select a k group-numbers from [1,k].
3. Randomly select a group-number (from the group-numbers selected in Step 2) for each object.

The second step is included to ensure that the group-numbers in the chromosomes are randomly distributed, rather than between 1 to k inclusive in each case.

Crossover

The context insensitive crossover operators, single-point and uniform, can be used for a group-number approach to the general clustering problem. However, again there is no definite relationship between the clusterings of the parents and the offspring. We saw in Chapter 2 that these crossovers were less effective for data sets with higher k values, so we consider these operators inappropriate for the general clustering problem (effectively we are looking for a maxk-clustering, where $maxk > k$).

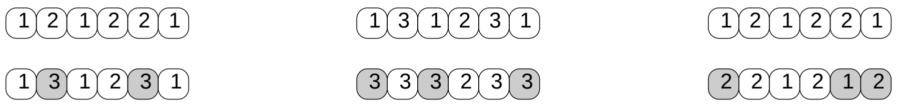
As mentioned previously, the edge-based crossover will always produce an offspring with k clusters. This is appropriate if both the parents have k clusters, but unsatisfactory otherwise. One solution to this problem is to modify the operator so that it produces two offspring and the k values of both parents are inherited by the children (the first child has the same number of clusters as the first parent, the second child has the same number of clusters as the second parent). Alternatively, the edge-based operator could be modified to produce a single child with its number of clusters randomly selected from the range of the parents' values. For example, if the parents have 3 and 6 clusters, the number of clusters in the child is selected from the interval $[3, 6]$. Thus the number of clusters in the child is influenced by the number of clusters in both parents.

Falkenauer's [18] crossover operator is similar to that of Bhuyan [7] and von Laszewski [66] in that it copies a number of groups from one parent to another and then uses a local search to produce the final child. Such an operator is easier to apply to Falkenauer's representation.

Mutation

The mutation operator for the k-clustering problem can be used for the general clustering problem. It is capable of both introducing new groups and removing existing groups in its current form. However, the mutation of a single gene is unlikely to result in any great increase in fitness, and mutations of this form will be quickly lost from the population [18]. Thus we need some method of introducing a new clusters that contain a number of objects.

The simplest approach may be to introduce a *split* operator that selects a group from a particular clustering and moves objects from that group into a new group with a set probability. Other mutation operators appropriate for this problem include a *merge* operator that moves all the objects in one cluster to another pre-existing cluster, and a *move* operator which shifts objects between groups already existing on a chromosome. Figure 31 illustrates the effects of these three operators. All three operators may be applied, with differing rates, to the population.



(a) Split (b) Merge (c) Move

Figure 31: Mutation operators for the general clustering problem: (a) Split randomly moves objects from group 2 into the new group 3; Merge moves all of the objects from group 1 into group 3; and Move randomly shuffles objects between pre-existing groups.

Falkenauer [17, 18] outlines similar strategies for his mutation operators — creating a new group, eliminating existing groups, and shuffling objects among their respective groups. However, once again, local search is used to place objects in groups during the mutation process. For example, a group is eliminated by moving all of the objects in that group to other groups according to a local search.

4.2 Methods

4.2.1 Objective Function

Our first step in adapting a GA for the general clustering problem involved finding an appropriate objective function. Thus the first part of our experiment was the evaluation of a number of clustering criterion. Each of our three data sets (Ruspini, German Towns, and Iris see Section 2.2.1) was clustered into 1 to 10 clusters using the iterated nearest neighbour algorithm from the previous chapter. Then the values of the criteria described in Section 4.1.1 were calculated for each clustering of each data set (the data was not standardised for this experiment). The objective function was selected on the basis of these values. The values of $s = 2.0$ and $t = 2.0$ were used for the Davies and Bouldin criterion during this experiment.

4.2.2 Clustering Algorithms

Genetic clustering was compared with three traditional clustering algorithms — Ward's method, iterated nearest neighbour, and simulated annealing — that were modified for the general clustering problem. For the first two methods, this involved finding the best clusterings into 2 to $maxk$ groups and then using the objective function value to select the optimal clustering from these (this required one run of Ward's method, and $maxk - 1$ runs of the iterated nearest neighbour algorithm). The neighbourhood definition used in the implementation of the simulated annealing algorithm allows the number of clusters to vary (moving the last object in one group into a pre-existing group will decrease the number of clusters by one, introducing a new group number will increase the number of clusters by one), so this was not changed. However, the initial clustering was created using the initialisation process described in Section 4.1.2, and the objective function was replaced.

To adapt the GCA for the general clustering problem, changes were made to the representation, fitness function, and the initialisation, crossover, and mutation operators. A group-number representation was used, but each chromosome also stored its number of clusters. Since the GCA was to search for the maximum objective function value, the objective function could be used as the fitness function without transformation. However, the local and scaled transformations were updated to suit this objective function. The initialisation operator described in Section 4.1.2, *choosek*, was added as an alternative to *random* initialisation. Two edge-based crossover operators were implemented: the first, *edge1*, producing two children with the same number of clusters as the first and second parent, respectively; and the second, *edge2*, producing a single child with its number of clusters randomly selected from $[k_1, k_2]$ where k_i is the number of clusters in the ith parent. The *split*, *move*, and *merge* mutation operators described previously were also implemented.

Each data set was clustered 40 times with each algorithm, and the clusterings found were ranked in order of frequency. The objective function value and the correctness (percentage of objects correctly clustered, defined in the previous chapter) of the most frequent clustering were also recorded. For this experiment the correctness of the clusterings was calculated from the *real* clusterings of the data sets. The correctness for the German Towns data set was not recorded since the real clustering for this data set is unknown. The average clustering time of the algorithms was measured in seconds of CPU time. A $maxk$ value of 10 was used throughout this experiment.

All of the tests were conducted on a Sparc Ultra, with load average around 0.9. The algorithms were written in C and compiled using gcc with the optimisation flag set to level 3.

The number of iterations for the iterated nearest neighbour algorithm were the same as those used in the previous experiment (Section 3.3). The parameters for the simulated annealing algorithms and GCAs were varied until a reasonable level of performance was achieved, or until the clustering time became excessive.

The mean time of the algorithms was compared against that of the GCAs using one-sided z tests (null hypothesis of equal mean values).

4.3 Results

4.3.1 Objective Function

Tables 25 to 27 list the values of the selected criteria for the clusterings of the three data sets. The correct clustering for the Ruspini data set gives the minimum Calinski and Harabasz value, and the maximum Davies and Bouldin criterion value, for the given clusterings.

k	Calinski and Harabasz	Davies and Bouldin
1	undefined	undefined
2	126.68	0.7582
3	136.28	0.5377
4^1	$425.33^\dagger$	0.4003^*
5	404.80	0.4835
6	379.46	0.6785
7	376.09	0.8189
8	370.29	0.7819
9	380.24	0.7798
10	389.72	0.7480

Table 25: Clustering criteria values for ten clusterings of the Ruspini data set. ^{1}correct clustering. *minimum criterion value. †maximum criterion value.

For the German Towns data set, the clustering with seven clusters has the maximum Calinski and Harabasz criterion value; whereas the clustering with ten clusters has the minimum Davies and Bouldin criterion value. In fact, the Davies and Bouldin values indicate that none of these clusterings represent the natural structure of the data (Davies and Bouldin [11] suggest that values above 0.6 ($s = t = 2.0$) for two-dimensional data indicate a particularly inappropriate clustering).

k	Calinski and Harabasz	Davies and Bouldin
1	undefined	undefined
2	76.34	0.8641
3	75.28	0.7955
4	86.66	0.8471
5	85.55	0.7545
6	88.00	0.7984
7	$92.09^\dagger$	0.8081
8	89.05	0.7809
9	87.56	0.8096
10	87.216	0.7014^*

Table 26: Clustering criteria values for ten clusterings of the German Towns data set. *minimum criterion value. †maximum criterion value.

For the Iris data set, the maximum Calinski and Harabasz criterion value is given by the clustering with three clusters; whereas the Davies and Bouldin criterion indicates that the clustering with two clusters is the best. The 3-clustering evaluated in Table 27 is not the correct clustering for the Iris data set. The criteria values for the correct clustering — 486.32 and 0.8446, respectively — are not the maximum (or minimum) value for either criterion. Consequently, neither of the criterion are suitable for finding the real clustering for this data set.

k	Calinski and Harabasz	Davies and Bouldin
1	undefined	undefined
2	513.30	0.4747*
3[2]	560.40†	0.7259
4	529.40	0.8438
5	494.09	0.8573
6	474.52	0.9913
7	450.77	1.0349
8	436.61	0.9990
9	409.70	1.0005
10	385.52	1.1162

Table 27: Clustering criteria values for ten clusterings of the Iris data set. [2]not the correct clustering (the criteria values for the correct clustering are 486.32 and 0.8446, respectively). *minimum criterion value. †maximum criterion value.

Since the Calinski and Harabasz criterion correctly indicated the number of clusters in the Ruspini and Iris data sets (the correct clustering for the German Towns data set is not defined) it was selected as the objective function for the clustering algorithms. However, since this measure is undefined when there is only one cluster, the implemented clustering algorithms were restricted to clusterings with more than one cluster.

4.3.2 Clustering Algorithms

The results of the comparison between the selected clustering algorithms can be found in Tables 28 to 30. Both Ward's method and the iterated nearest neighbour algorithm found the correct clustering of the Ruspini data on every run. However, the simulated annealing algorithm was unable to find the correct clustering. In fact, this algorithm always clustered the Ruspini data set into *maxk* clusters. The GCA performed reasonably well, finding the correct clustering on 35 of the 40 runs. Ward's method and the iterated nearest neighbour algorithm were significantly faster than the GCA.

Ward's method found the same clustering of the German Towns data set on each run; this clustering divided the objects between ten groups, and had the lowest objective function value of the clusterings found by the algorithms. The simulated annealing algorithm also clustered the data set into ten groups, but this clustering still had a relatively high objective function value. The most frequent clustering for both the iterated nearest neighbour and GCAs had seven groups,

Algorithm	Frequency	k	Obj. Function	Correctness	Time	
			Value	%	sec. CPU	std dev.
Ward's method	40	4	425.33	100.0	0.022†	0.008
iterated nearest neighbour[1]	40	4	425.33	100.0	0.549†	0.012
simulated annealing[2]	7	10	356.80	54.7	712.151	6.284
GCA[3]	35	4	425.33	100.0	602.182	51.245

Table 28: Comparison of clustering algorithms on Ruspini. [1]100 iterations. [2]$T_0 = 10.0, T_{steps} = 300, T_\delta = 0.999$, and $T_F = 0.000001$. [3]PS = 200, MG = 200, In = random, Tr = none, El = 5, Cr = edge2, CP = 1.00, MR = move 0.70, merge 0.10 – 0.50. †significantly faster than the GCA at 1% level.

Algorithm	Frequency	k	Obj. Function	Time	
			Value	sec. CPU	std dev.
Ward's method	40	10	78.87	0.011†	0.008
iterated nearest neighbour[1]	35	7	92.09	42.973†	0.686
simulated annealing[2]	7	10	86.88	564.614	9.416
GCA[3]	17	7	92.09	546.022	18.288

Table 29: Comparison of clustering algorithms on German Towns. [1]10000 iterations. [2]$T_0 = 100.0, T_{steps} = 300, T_\delta = 0.999$, and $T_F = 0.000001$. [3]PS = 100, MG = 2000, In = random, Tr = none, El = 5, Cr = edge2, CP = 1.00, MR = move 0.70, merge 0.10 – 0.40. †significantly faster than the GCA at 1% level.

and was the clustering with the highest objective function value for this data set. However, the iterated nearest neighbour algorithm was twice as successful at finding this clusters as the GCA. In addition, the iterated nearest neighbour algorithm was significantly faster than the GCA.

The iterated nearest neighbour algorithm was the only method that divided the Iris data into the correct number of clusters. Ward's method, and the GCA, found two clusters on every run, while the simulated annealing algorithm always found a clustering with ten groups.

Algorithm	Frequency	k	Obj. Function	Correctness	Time	
			Value	%	sec. CPU	std dev.
Ward's method	40	2	501.93	66.7	0.161†	0.008
iterated nearest neighbour[1]	40	3	560.40	89.3	14.863†	0.120
simulated annealing[2]	21	10	380.59	41.3	947.744	15.657
GCA[3]	40	2	513.30	66.7	668.516	74.050

Table 30: Comparison of clustering algorithms on Iris. [1]1000 iterations. [2]$T_0 = 10.0, T_{steps} = 200, T_\delta = 0.999$, and $T_F = 0.000001$. [3]PS = 50, MG = 1000, In = random, Tr = none , El = 5, Cr = edge2, CP = 1.00, MR = move 0.70, merge 0.10 – 0.30. †significantly faster than the GCA at 1% level.

Numerous parameter combinations were tested for the simulated annealing algorithm on all of the data sets. However, the algorithm always found clusterings with nine or ten clusters, with the most frequent clusterings having ten clusters.

Initial attempts at finding good parameter values for the GCA involved using the *choosek* initialisation operator, the *edge1* crossover operator, and various rates of the three mutation operators. The resulting performance was poor, with the population converging rapidly to a sub-optimal number of clusters (Figure 32). Subsequent attempts used the random initialisation operator to seed the population with clusterings containing *maxk* clusters. The probability of the *merge* operator was exponentially increased over the generations to force the GCA to explore clusterings with less groups. The *edge2* crossover operator was used since it can create a child with a different number of groups than either parent, and the *move* mutation operator used to move objects between pre-existing groups. These parameter values forced the GCA's population to move through different k values over the generations (Figure 33). The results presented in Tables 28 to 30 use these parameter values.

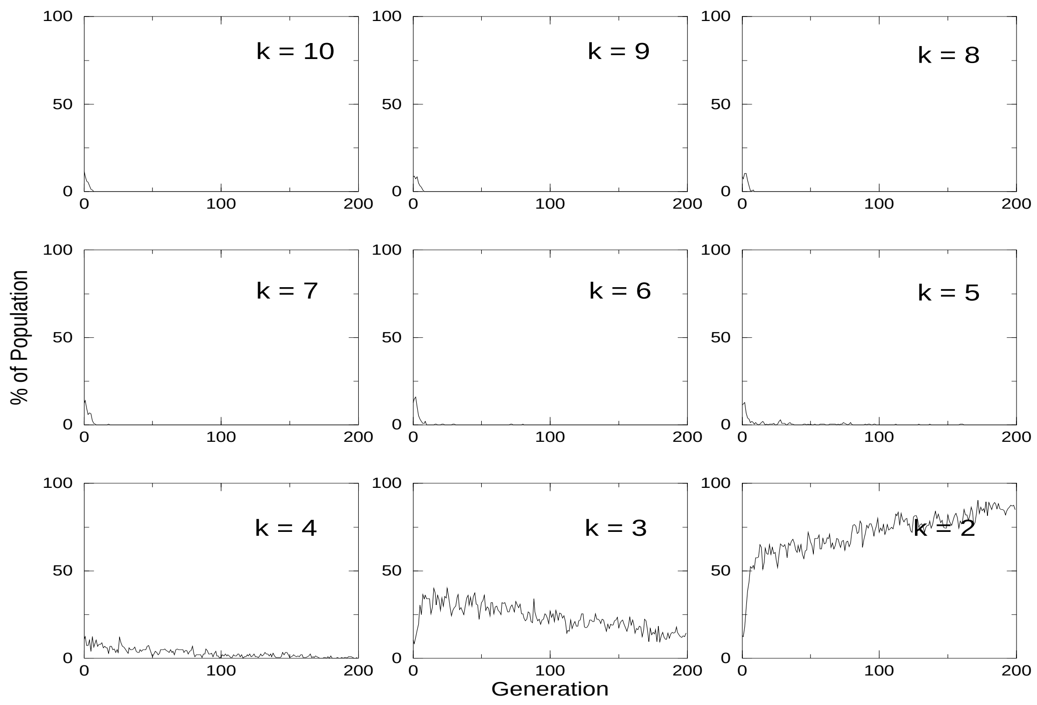


Figure 32: Distribution of k values during genetic clustering of the Ruspini data set with initial parameter values: PS = 200, MG = 200, In = choosek, Tr = none, El = 5, Cr = edge1, CP = 1.00, MR = split 0.30 – 0.10, merge 0.30 – 0.10, move 0.70.

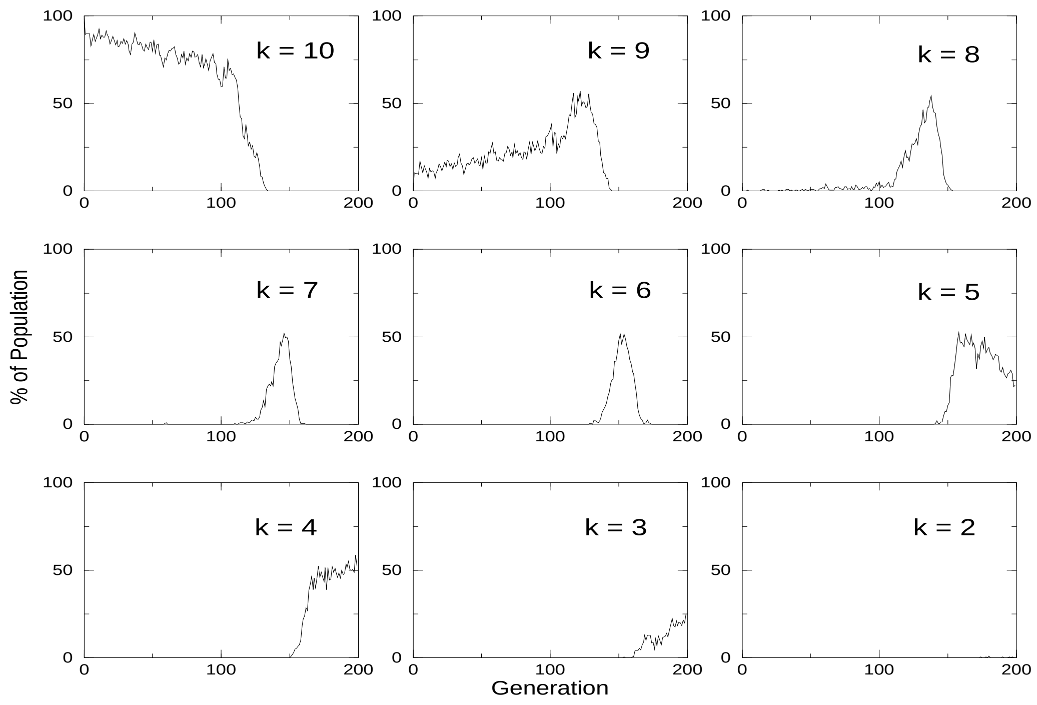


Figure 33: Distribution of k values during genetic clustering of the Ruspini data set with subsequent parameter values: PS = 200, MG = 200, In = random, Tr = none, El = 5, Cr = edge2, CP = 1.00, MR = merge 0.10 – 0.50, move 0.70.

4.3.3 Local Search

A final attempt at improving the GCA's performance involved modifying the *merge* mutation operator. Instead of joining two random groups, the operator was modified to place objects from a randomly selected group into the clusters (not including the selected group) with the closest centroid — each object was placed in the cluster with the closest centroid to it. The overall strategy was similar to the one described previously, with the population seeded with maxk-clusterings which were then merged to form clusterings with less groups. The modified *merge* should enhance the fitness of these clusterings. In order to prevent the population from converging to too few groups, *split* mutation was used.

The resulting performance of the GCA on our three data sets is contained in Table 31. This algorithm performed better on both Ruspini and Iris, finding the correct clustering in every run for the Ruspini data, and every run except one for the Iris data (this is comparable with the performance of the iterated nearest neighbour algorithm). In addition, on these data sets, the algorithm was significantly faster (at 1% level) with these parameter values.

Modifying the *merge* operator also improved the performance of the GCA on the German Towns data set, but the algorithm still only found the correct solution on 60% of the runs. Further, the modified algorithm was significantly slower (at 1% level) than the previous version of the algorithm. Clearly, the iterated nearest neighbour algorithm still produced the best results for this data set.

Data	Frequency	k	Obj. Function	Correctness	Time	
			Value	%	sec. CPU	std dev.
Ruspini[1]	40	4	425.33	100.0	307.556	18.671
German Towns[2]	24	7	92.09	-	643.814	42.077
Iris[3]	39	3	560.40	89.33	522.471	56.755

Table 31: Performance of GCAs with modified merge mutation. [1]PS = 200, MG = 200, In = random, Tr = none , El = 5, Cr = edge2, CP = 1.00, MR = move 0.70, merge 0.10 – 0.30, split 0.05 – 0.25. [2]PS = 100, MG = 2000, In = random, Tr = none , El = 5, Cr = edge2, CP = 1.00, MR = move 0.70, merge 0.10 – 0.40, split = 0.05 – 0.20. [3]PS = 50, MG = 1000, In = random, Tr = none , El = 5, Cr = edge2, CP = 1.00, MR = move 0.70, merge 0.10 – 0.30, split 0.05 – 0.15.

4.4 Discussion

4.4.1 Objective Function

The objective function for a clustering algorithm should: (i) be defined for every clustering the algorithm can produce; and (ii) provide a relative measure of each clustering's worth — the minimum (or maximum) value should correspond to the optimal clustering for the data set. None of the clustering criteria tested in this chapter satisfied these requirements for all three data sets. The Calinski and Harabasz, and Davies and Bouldin measures are undefined when all objects are placed in a single cluster, and neither could identify the real clustering of the Iris data set. However, the Calinski and Harabasz criterion can be used to determine the correct number of clusters in all of the given data sets, whereas the Davies and Bouldin measure only identified the correct number of clusters for the Ruspini data set. Further, the values of this measure for the German Towns data set suggest that none of the clusterings for this data set are appropriate. These results agree with those of Milligan and Cooper [49] in which the Calinski and Harabasz criterion outperformed the Davies and Bouldin measure.

4.4.2 Clustering Algorithms

The fact that the objective function was an external criterion for Ward's method (it was used to select from clusterings that had been produced by the hierarchical algorithm) meant that this algorithm could not always find the clustering with the maximum objective function value. In fact, this algorithm found clusterings with the wrong number of clusters for both German Towns and Iris, since these clusterings had higher objective function values than any of the other clusterings produced by the algorithm. However, a different external criterion may produce better results with this method.

The iterated nearest neighbour algorithm clearly out-performed the other clustering algorithms on all three data sets. This is a result of the high correspondence between the method used to create the clusterings and the external objective function. The algorithm creates clusterings by placing objects into clusters with the closest centroid; this minimises the sum of the Euclidean distance between objects and their cluster centres, which is the denominator of the Calinski and Harabasz criterion. However, since these clusters are built around random seed points, the algorithm must be iterated; and complex data sets (high k, close clusters) will require a large number of iterations to produce good results (for complex data sets, there are fewer seed points that will lead to good clusterings). Despite this, this method is still a relatively fast method of clustering data sets.

The simulated annealing algorithm always clustered the data sets into $maxk$ groups. This is most probably due to the interaction between the neighbourhood definition, which allows a single object to be moved into a random group, and the objective function. Moving a single object will not often increase the objective function value, unless the object is being moved to a new group. Hence the algorithm will tend to divide the objects into as many clusters as possible.

The performance of the GCA on the first two data sets is promising, although this requires a particular set of adaptions: random initialisation, exponentially increasing *merge* mutation, and *edge2* crossover. This combination of adaptions uses the same concept as the hierarchical clustering algorithms, that is merging two clusters in a good k-clustering will most likely result in a good (k-1)-clustering. Thus the random initialisation operator is used to create a population of maxk-clusterings, and the *merge* mutation operator is applied exponentially to allow the population to develop some good clusterings for each number of clusters, before the population evolves to a new number of clusters. Elite selection ensures that the best clusterings found so far remain in the population, and ensures that the final population is based around clusterings with the correct number of clusters (assuming that these are also the fittest clusterings).

The *edge2* crossover operator enables a child chromosome to have a different number of clusters to its parents. But the number of clusters is restricted to the range of the number of clusters in the parents. Given that edge-based crossover creates a child by finding groups in both parents, this operator will often have the same effect as the *merge* operator.

The emphasis on merging explains the poor performance of the GCA on the Iris data set. The

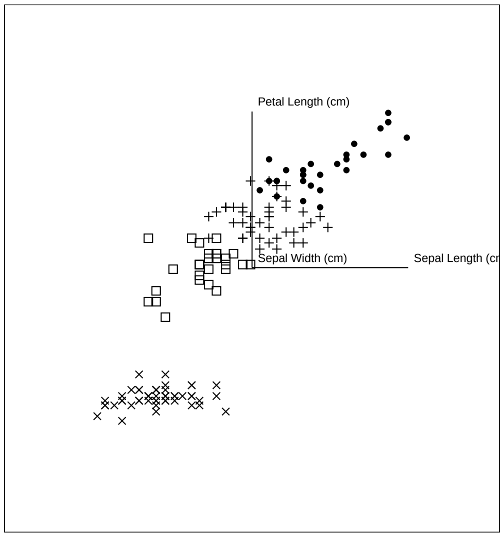


Figure 34: Optimal 4-clustering for the Iris data set.

best clustering of the Iris data set into four clusters divides two of the correct groups between three different clusters, one of which has equal numbers of the objects that belong in different groups (Figure 34). Thus the optimal clustering cannot be reached by merging any two of the existing clusters, and such a merging would result in a clustering with a relatively low fitness value. However, if the three closest clusters are merged, a good 2-cluster results, which is why the GCA finds this clustering rather than the optimal one. A similar effect can be observed in the results for Ward's method since the hierarchical algorithms also work by merging clusters.

Since the GCA works by exchanging information (in this case cluster membership) between members of its population, it appears that the GCA (as described in this Chapter) is not suitable for the k-clustering problem — the clusters contained in a good k-clustering will not necessarily lead to a good (k-1)-clustering. Thus, allowing the clusterings in the population to have differing numbers of clusters, and exchanging clusters between them using an edge-based crossover, is perhaps not the best approach to this problem.

4.4.3 Local Search

The modified *merge* operator used a local search to place objects from a randomly selected group into the clusters with the closest centroids. Thus the GCA could form better (k-1)-clusterings from the k-clusterings in its population (hence the GCA was more likely to find the optimal solution). Due to the increased fitness of the clusterings with less groups, the GCA converged to clusterings with fewer clusters in a shorter space of time (except for the German Towns data, where the relatively high fitness of clusterings with seven clusters, kept the number of clusters in the population high). Given that the majority of time per generation is spent on crossover, and that the complexity of the edge-based crossover is proportional to the number of clusters, this meant that this modification actually decreased the time taken for the GCA on the Ruspini and Iris data sets. This was despite the fact that the modification to the *merge* operator increased the time taken for this operator. However, since the number of clusters in the population of the GCA for the German Towns data set remained high, there was a noticeable increase in the time taken for this data set.

The success of the modified GCA was comparable to the iterated nearest neighbour algorithm for the Ruspini and Iris data sets. However, the iterated nearest neighbour algorithm was more successful on the German Towns data set, and was significantly faster than the GCA on every data set. Thus there is no advantage to choosing the GCA over the iterated nearest neighbour algorithm, even with the addition of a local search.

Chapter 5

Conclusions

This research has been an investigation into the use of GAs for clustering data. We reviewed the various adaptions that enable GAs to cluster into a pre-defined number of groups, and compared the performance of 4320 combinations of adaptions on three generated data sets. Each adaption combination was ranked according to: (1) the number of times (out of five replications) that it found the clustering with the minimum objective function value in less than five processor minutes; and (2) the average time taken to find this clustering. Independent generalised linear models were fitted to the correctness and time results for each of the generated data sets, and the models were used to predict the performance of the GCAs on three real data sets with similar structure. The linear models quantified the effects of the various adaptions on the performance of the GCAs; and accurate predictions were produced, by models with good fit, when there was high similarity between the real and generated data.

A number of adaptions are essential for GCAs, namely an appropriate representation, fitness function, and suitable operators — the GCA needs to be able to encode potential solutions; the fitness function is necessary to drive the evolution of the population toward the optimal clustering; and the operators must be able to produce valid offspring by manipulating the representation. Other adaptions such as context sensitive operators, elite selection, and parameter values such as the population size and the mutation rate can have a large effect on the time taken to find the optimal clustering for a given data set. Further, the effects of these adaptions and parameters varies for different data sets, although comparison between the real and generated data sets suggests that the variation is not significant for data sets with the same number of objects, clusters, attributes, and a similar distribution of objects. However, the group-number representation is better suited for the k-clustering problem than the order-based representation, and high dynamic scaling of the fitness function values, elite selection, and high mutation rates increase the probability of finding the correct clustering within a reasonable time. Larger population sizes and context sensitive crossover appear more important for data sets with a high number of clusters.

Comparison of GCAs with traditional k-clustering algorithms for the three data sets shows, that although the GCAs cluster the data successfully, the method provides no advantages over traditional methods. This is due to the high complexity of the GCAs when compared to the traditional algorithms, which are based around simple distance measures. Further, these same distance measures are used in several clustering criteria, which limits the correctness of both GCAs using these clustering criteria, and the traditional algorithms based around these measures.

Only simple modifications to the developed GCAs were needed for the general clustering problem, where the value of k is unknown. The best performance resulted from exploiting the merging concept behind hierarchical clustering, by seeding the population with clusterings with high numbers of clusters and then forcing the GCA to evolve clusterings with fewer clusters. If the optimal solution is found during this evolution process, the elite selection strategy will ensure that it remains in the population. However, such a GCA can become trapped in a local minimum in a similar manner to various hierarchical algorithms, since merging clusters in a good k-clustering does not always lead to a good (k-1)-clustering. In fact, the population structure of the GCA (as described in this thesis) is not appropriate for the general clustering problem, as given free reign it evolves the population by swapping genetic information between randomly selected members of the population (effectively, the GCA is trying to build good k-clusterings using subsets of groupings found in good 2-clusterings to maxk-clusterings).

The addition of local search to the merge operator markedly improved the performance of the GCAs on both the German Towns, and the Iris data sets. For the Iris data the GCA found the clustering with the maximum objective function value in 39 out of 40 trials (this clustering correctly allocates 89.3% of the objects according to the real clustering of the data, and is comparable to the performance of other clustering GAs [6]).

There are two factors limiting the performance of GCAs: the choice of objective function, and the high complexity of the GCA. The objective function drives the evolution of the GCA, and as such if the objective function does not suit the structure of the data, the GCA will be unable to find good clusterings for the data set. Further, traditional clustering algorithms are based on concepts similar to the majority of clustering criteria. Thus the solutions given by these methods match, or better any that can be found by using these clustering criteria. Plus the complexity of these traditional algorithms is low. There is no advantage to conducting a more thorough search of the solution space, as the objective function is not sufficient to find a better solution. Incorporating heuristics into the GCA will improve its performance, but for the same reasons will not match the performance of the heuristic on its own.

There are four possible areas of improvement for the genetic clustering algorithms. Firstly, the GCAs could be adapted to control their own operator probabilities [13, 61]. This may even involve different rates according to the fitness of each chromosome, so that, for example, less fit chromosomes are more likely to undergo mutation. This would remove the need to determine good parameter values, and should enhance the performance of the GCAs.

Secondly, better clustering criteria may give the GCA an advantage over other clustering methods. Certainly, work in the area of fuzzy clustering suggests that genetic clustering may benefit from the clustering criteria in this field (the matrix representation is suitable for fuzzy clustering) [22, 42]. GCAs may perform better when there are large numbers of sub-optimal minima that trap other search techniques.

Thirdly, parallel implementations of GAs [66, 50] divide the algorithm between a number of computers by creating a number of small populations rather than a single larger population. Such a method may offer both time and correctness advantages. Further, individual sub-populations could be devoted to a certain number of clusters, thus this method may give worthwhile results for the general clustering problem.

Finally, the addition of further adaptions, such as modifications to the population structure and better reproductive operators, may enhance the effectiveness of the GCAs' search.

Bibliography

[1] Cesare Alippi and Rita Cucchiara. Cluster partitioning in image analysis classification : A genetic algorithm approach. In *CompEuro 1992 Proceedings. Computer Systems and Software Engineering*, pages 139–44. IEEE Computer Society Press, 1992.

[2] Michael R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[3] Jim Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 86–91. Morgan Kaufman Publishers, 1989.

[4] James Edward Baker. Reducing bias and inefficiency in the selection algorithm. In J J Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[5] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, 1981.

[6] James C. Bezdek, Srinivas Boggavarapu, Lawrence O. Hall, and Amine Bensaid. Genetic algorithm guided clustering. In *Conference Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 34–9. IEEE, 1994.

[7] Jay Bhuyan. A combination of genetic algorithm and simulated evolution techniques for clustering. In C. Jinshong Hwang and Betty W. Hwang, editors, *Proceedings of 1995 ACM Computer Science Conference*, pages 127–34. The Association for Computing Machinery, Inc., 1995.

[8] Jay N. Bhuyan, Vijay V. Raghavan, and Venkatesh K. Elayavalli. Genetic algorithms for clustering with an ordered representation. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 408–15. Morgan Kaufman Publishers, 1991.

[9] A. Brindle. *Genetic Algorithms for Function Optimization.* PhD thesis, University of Alberta, Edmonton, 1981. Cited in [24].

[10] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in statistics*, 3(1):1–27, 1974.

[11] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, PAMI-1(2):224–7, 1979.

[12] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–164, 1985. Cited in [48].

[13] Lawrence Davis. Adapting operator probabilities in genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufman Publishers, 1989.

[14] Lawrence Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[15] Brian S. Everitt. *Cluster Analysis.* Halsted Press, third edition, 1993.

[16] Ludwig Fahrmeir and Gerhard Tutz. *Multivariate Statistical Modelling Based on Generalized Linear Models.* Springer-Verlag, 1994.

[17] E. Falkenauer. The grouping genetic algorithms: widening the scope of the gas. *Belgian Journal of Operations Research, Statistics and Computer Science*, 33(1,2):79–102, 1993.

[18] Emanuel Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–44, 1994.

[19] W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53:789–798, 1958. Cited in [8].

[20] K. Florek, J. Lukaszewiez, J. Perkal, H. Steinhaus, and S. Zubrzchi. Sur la liason et la division des points d'un ensemble fini. *Colloquium Mathematicum*, 2:282–285, 1951. Cited in [15].

[21] M. Funk, R. D. Appel, Ch. Roch, D. Hochstrasser, Ch. Pellegrini, and A.F. Müller. Knowledge acquisition in expert system assisted diagnosis: a machine learning approach. In *Proc. AIME-87*, pages 99–103. Springer Verlag, 1987.

[22] I. Gath and A. B. Geva. Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):773–81, 1989.

[23] D. E. Goldberg and R. Lingle. Alleles, loci, and the tsp. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985. Cited in [48].

[24] David E. Goldberg. *Genetic Algorithms - in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc, 1989.

[25] David E. Goldberg and Philip Segrest. Finite markov chain analysis of genetic algorithms. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 1–8. Lawrence Erlbaum Associates, New Jersey, 1987.

[26] A. D. Gordon. *Classification : Methods for the Exploratory Analysis of Multivariate Data.* Chapman and Hall Ltd, London, 1981.

[27] Paul E. Green, Ronald E. Frank, and Patrick J. Robinson. Cluster analysis in test market selection. *Management Science*, 13(8):387–400, 1967. Series B.

[28] John J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1):122–8, 1986.

[29] John A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.

[30] F. Roy Hodson. Numerical typology and prehistoric archaeology. In F. R. Hodson, D. G. Kendall, and P. A. Taŭtu, editors, *Mathematics in the Archaeological and Historical Sciences*. University Press, Edinburgh, 1971.

[31] John Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[32] S. Levine I. Pilowski and D. M. Boulton. The classification of depression by numerical taxonomy. *The British Journal of Psychiatry*, 115:937–945, 1969.

[33] A. K. Jain and J. V. Moreau. Bootstrap technique in cluster analysis. *Pattern Recognition*, 20(5):547–68, 1987.

[34] Cezary Z. Janikow. An experimental comparison of binary and floating point representations in genetic algorithms. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 31–36. Morgan Kaufmann Publishers, San Mateo, California, 1991.

[35] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, New Jersey, third edition, 1992.

[36] Donald R. Jones and Mark A. Beltramo. Solving partitioning problems with genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 442–9. Morgan Kaufman Publishers, 1991.

[37] K. A. De Jong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975. Dissertation Abstracts International, 36(10), 5140B, (University Microfilms No. 76-9381).

[38] Kenneth A. De Jong. Genetic algorithms are not function optimizers. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 5–17. Morgan Kaufmann, San Mateo, California, 1993.

[39] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, Inc., 1990.

[40] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[41] Raymond W. Klein and Richard C. Dubes. Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, 22(2):213–20, 1989.

[42] Raghu Krishnapuram and Chih-Pin Freg. Fuzzy algorithms to find linear and planar clusters and their applications. In *Proceedings 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 426–31. IEEE Comput. Soc. Press, 1991.

[43] Ravindra Krovi. Genetic algorithms for clustering : A preliminary investigation. In V. Milutinovic, B. D. Shriver, J. F. Jr. Numaker, and R. H. Jr. Sprague, editors, *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, pages 540–4. IEEE Computer Society Press, 1991.

[44] G. N. Lance and W. T. Williams. A general theory of classificatory sorting strategies. 1. hierarchical systems. *The Computer Journal*, 9:373–380, 1966. Cited in [65].

[45] E. Levrat, V. Bombardier, M. Lamotte, and J. Bremont. Multi-level image segmentation using fuzzy clustering and local membership variations detection. In *IEEE International Conference on Fuzzy Systems*, pages 221–8. IEEE, New York, 1992.

[46] G. L. Liu. *Introduction to combinatorial mathematics.* McGraw Hill, 1968.

[47] P. McCullagh and J. A. Nelder. *Generalized Linear Models.* Chapman and Hall, London, 2nd edition, 1989.

[48] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, third, revised and extended edition, 1996.

[49] Glenn W. Milligan and Martha C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–79, 1985.

[50] Heinz Mühlenbein. Asynchronous parallel search by the parallel genetic algorithm. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 526–33. IEEE Computer Society Press, 1991.

[51] P. M. Murphy and D. W. Aha. UCI Repository of machine learning databases. [Machine-readable data repository]. Irvine, CA: University of California, Department of Information and Computer Science, 1994.

[52] Allen E. Nix and Michael D. Vose. Modeling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:79–88, 1992.

[53] Girish Punj and David W. Stewart. Cluster analysis in marketing research: Review and suggestions for application. *Journal of Marketing Research*, XX:134–48, 1983.

[54] Y. Reich and S. Fenves. The formation and use of abstract concepts in design. In D. Fisher, M. Pazzani, and P. Langley, editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pages 323–54. Morgan Kaufmann, San Mateo, Calif., 1991.

[55] E. H. Ruspini. Numerical methods for fuzzy clustering. *Inform. Sci.*, 2:319–150, 1970.

[56] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufman Publishers, 1989.

[57] R. Sibson. Slink: An optimally efficient algorithm for the single-link cluster method. *Computer J.*, 16:30–4, 1973.

[58] Robert R. Sokal and Peter H. A. Sneath. *Principles of numerical taxonomy.* W. H. Freeman, San Francisco, 1963. Cited in [65].

[59] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons. *Biol Skr*, 5:1–34, 1948. Cited in [65].

[60] Helmuth Späth. *Cluster analysis algorithms: For data reduction and classification of objects.* Ellis Horwood Limited, 1980.

[61] M. Srinivas and L. M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[62] Joe Suzuki. A markov chain analysis on simple genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(4):655–659, 1995.

[63] M. J. Symons. Clustering criteria and multivariate normal mixtures. *Biometrics*, 37:35–43, 1981.

[64] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-Plus.* Springer-Verlag, 1994.

[65] Wolfgang Vogt and Dorothea Nagel. Cluster analysis in diagnosis. *Clinical Chemistry*, 38(2):192–98, 1992.

[66] Gregor von Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufman Publishers, 1991.

[67] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58:236–244, 1963. Cited in [15].

Appendix A

Models for GA adaptions

This appendix contains the coefficients for all of the models used in Chapter 2. There are two models for each representation/data set combination, one that gives a probability of finding the solution within 300 seconds, the other giving a time to find the solution. These models were used to determine the best set of adaptions for each data set (Tables 14 to 18). Since the order-based GCAs were unable to find the correct clustering for the Iris2 data set there are no models for this representation/data set combination.

In each case the correctness models were derived from the complete set of correctness data. The time models were based only on the GCAs which found one or more solutions, since there was no time data for the GCAs that didn't find a solution. The data for GCAs with PMX crossover was removed from the time models for the order-based GCAs since the number with correct runs was extremely low (6 for Ruspini2 and 9 for Towns2). Some of the models have coefficients that are not defined due to singularity (the data was insufficient to determine the effect of the factor). This is an indication of a low probability of a GCA with this particular factor finding the correct solution in under 300 seconds.

The fit of each model is also assessed, using the residual deviance, comparison of the experimental and fitted values, and plots of the residuals for the time models. The final model in each case was the model that gave the best fit with no higher than third-order terms (the time taken to fit higher order models was prohibitive).

For both the correctness and time models a good fit is indicated by a low residual deviance, preferably lower than the corresponding degrees of freedom. The residual plots should be randomly distributed around a mean of zero, and the range of the residuals should be approximately uniform along the fitted scale (the fitted values have been transformed to the constant-information scale of the error distribution, $2\sqrt{\mu}$ for Poisson errors [47]).

A.1 Group-number and Ruspini2

We will start with the correctness model for the group-number GCAs on the Ruspini2 data set. The coefficients are listed in Table 32, with Figures 35 containing the comparative histograms. The residual deviance of 2084.3 on 2904 degrees of freedom indicates that this model fits well, although

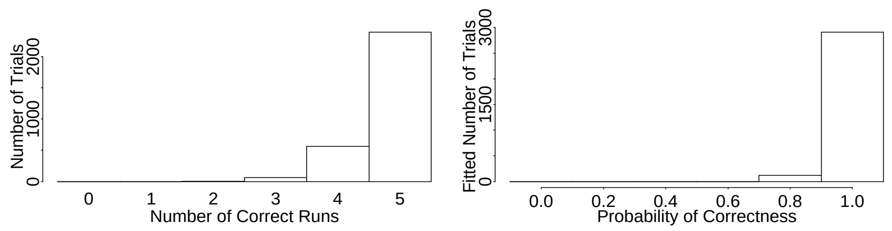


Figure 35: Comparison of experimental and fitted correctness values for group-number GCAs on Ruspini2.

the comparative histograms show that the model over-predicts the number of trials with five runs correct.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	2.6618	PS100	0.0071	PS200	1.2169
PS400	0.7550	Trlocal	0.5587	Trscale2	0.0502
Trscale4	0.1576	Crsingle	-0.4156	Cruniform	0.4430
MR0.05	-0.1014	MR0.10	-0.5742	MR0.20	-0.0000
MR0.50	0.3748	MR0.70	-0.1014	MR0.90	0.1114
Crsingle:MR0.05	0.3431	Cruniform:MR0.05	0.4974	Crsingle:MR0.10	1.0104
Cruniform:MR0.10	0.6614	Crsingle:MR0.20	0.5475	Cruniform:MR0.20	0.6612
Crsingle:MR0.50	1.0133	Cruniform:MR0.50	0.2864	Crsingle:MR0.70	0.3431
Cruniform:MR0.70	0.3850	Crsingle:MR0.90	1.2767	Cruniform:MR0.90	0.1722
PS100:Crsingle	0.7751	PS200:Crsingle	-1.4620	PS400:Crsingle	0.2854
PS100:Cruniform	-0.6665	PS200:Cruniform	-2.0313	PS400:Cruniform	-1.0767
PS100:MR0.05	0.1014	PS200:MR0.05	-1.0358	PS400:MR0.05	-0.6167
PS100:MR0.10	0.3090	PS200:MR0.10	-0.5630	PS400:MR0.10	1.2797
PS100:MR0.20	-0.1405	PS200:MR0.20	4.9454	PS400:MR0.20	-0.4179
PS100:MR0.50	0.5000	PS200:MR0.50	4.5706	PS400:MR0.50	-0.3748
PS100:MR0.70	1.3885	PS200:MR0.70	-1.0358	PS400:MR0.70	-0.8524
PS100:MR0.90	7.0244	PS200:MR0.90	-0.1114	PS400:MR0.90	-1.8224
Trlocal:Crsingle	-0.4395	Trscale2:Crsingle	0.0079	Trscale4:Crsingle	0.3295
Trlocal:Cruniform	-0.6575	Trscale2:Cruniform	-0.7235	Trscale4:Cruniform	-1.2511
Crsingle:MR0.05:PS100	-0.7734	Cruniform:MR0.05:PS100	-0.3309	Crsingle:MR0.10:PS100	-1.4177
Cruniform:MR0.10:PS100	-0.5422	Crsingle:MR0.20:PS100	-0.2674	Cruniform:MR0.20:PS100	-0.3542
Crsingle:MR0.50:PS100	-2.4048	Cruniform:MR0.50:PS100	-0.9015	Crsingle:MR0.70:PS100	-2.2273
Cruniform:MR0.70:PS100	-0.9362	Crsingle:MR0.90:PS100	-8.4125	Cruniform:MR0.90:PS100	-6.4120
Crsingle:MR0.05:PS200	1.4132	Cruniform:MR0.05:PS200	1.0473	Crsingle:MR0.10:PS200	0.5329
Cruniform:MR0.10:PS200	1.5213	Crsingle:MR0.20:PS200	-4.9854	Cruniform:MR0.20:PS200	-5.0974
Crsingle:MR0.50:PS200	-5.3397	Cruniform:MR0.50:PS200	-4.4867	Crsingle:MR0.70:PS200	1.8369
Cruniform:MR0.70:PS200	1.4973	Crsingle:MR0.90:PS200	-0.3941	Cruniform:MR0.90:PS200	1.5851
Crsingle:MR0.05:PS400	-0.0172	Cruniform:MR0.05:PS400	-0.1165	Crsingle:MR0.10:PS400	-1.9287
Cruniform:MR0.10:PS400	-1.0057	Crsingle:MR0.20:PS400	-0.2411	Cruniform:MR0.20:PS400	0.3532
Crsingle:MR0.50:PS400	-0.5899	Cruniform:MR0.50:PS400	0.3101	Crsingle:MR0.70:PS400	1.0341
Cruniform:MR0.70:PS400	1.4650	Crsingle:MR0.90:PS400	0.8577	Cruniform:MR0.90:PS400	1.9000
PS100:Trlocal:Credge	0.8135	PS200:Trlocal:Credge	1.7935	PS400:Trlocal:Credge	1.3700
PS100:Trscale2:Credge	0.6108	PS200:Trscale2:Credge	6.8054	PS400:Trscale2:Credge	1.5857
PS100:Trscale4:Credge	0.3810	PS200:Trscale4:Credge	6.6980	PS400:Trscale4:Credge	2.8800
PS100:Trlocal:Crsingle	-0.1716	PS200:Trlocal:Crsingle	0.4107	PS400:Trlocal:Crsingle	-0.6371
PS100:Trscale2:Crsingle	-0.2551	PS200:Trscale2:Crsingle	0.3992	PS400:Trscale2:Crsingle	-0.5152
PS100:Trscale4:Crsingle	0.0527	PS200:Trscale4:Crsingle	-0.0981	PS400:Trscale4:Crsingle	-0.8117
PS100:Trlocal:Cruniform	-0.1431	PS200:Trlocal:Cruniform	-0.1405	PS400:Trlocal:Cruniform	-0.2260
PS100:Trscale2:Cruniform	0.9113	PS200:Trscale2:Cruniform	0.5475	PS400:Trscale2:Cruniform	0.4065
PS100:Trscale4:Cruniform	1.2672	PS200:Trscale4:Cruniform	1.8184	PS400:Trscale4:Cruniform	0.5155

Table 32: Coefficients for the correctness model for group-number GCAs on Ruspini2.

The coefficients for the corresponding time model are listed in Table 33. The residual deviance for this model is 2405.7 on 2694 degrees of freedom, indicating a good fit which is supported by the similarities between the experimental and predicted values shown in Figure 36. The plot of the residual values (Figure 37) shows perhaps a little skewness, but no indications of serious violations of the model assumptions.

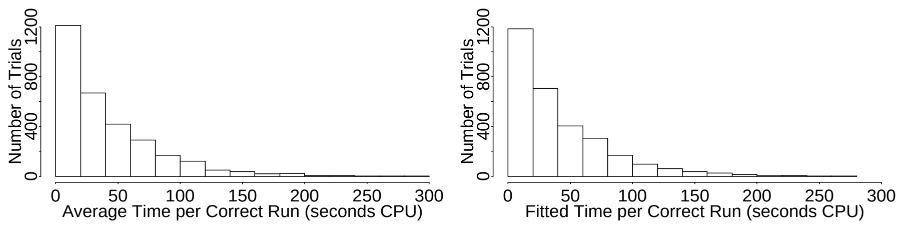


Figure 36: Comparison of experimental and fitted time values for group-number GCAs on Ruspini2.

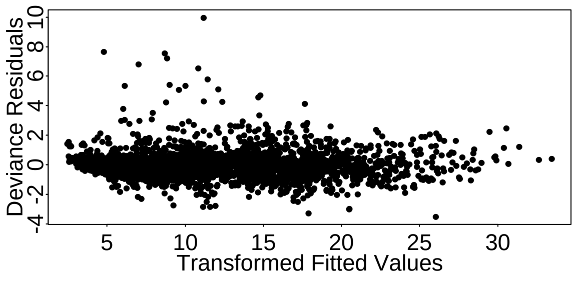


Figure 37: Residual plot for time model for group-number GCAs on Ruspini2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	3.8172	PS100	0.0833	PS200	0.4025
PS400	0.5480	Trlocal	0.2539	Trscale2	0.2394
Trscale4	0.0869	El1	0.0889	El5%	0.2455
Crsingle	0.1355	Cruniform	0.4006	CP0.70	0.1554
CP0.90	0.3129	MR0.05	-1.1686	MR0.10	-1.3989
MR0.20	-1.5000	MR0.50	-1.6459	MR0.70	-1.5176
MR0.90	-1.3481	PS100:Crsingle	-0.0183	PS200:Crsingle	-0.3530
PS400:Crsingle	-0.3351	PS100:Cruniform	0.0365	PS200:Cruniform	-0.1112
PS400:Cruniform	-0.1556	PS100:MR0.05	0.3162	PS200:MR0.05	0.3653
PS400:MR0.05	0.8118	PS100:MR0.10	0.3002	PS200:MR0.10	0.5214
PS400:MR0.10	0.9020	PS100:MR0.20	0.3726	PS200:MR0.20	0.5291
PS400:MR0.20	0.9506	PS100:MR0.50	0.6151	PS200:MR0.50	0.6688
PS400:MR0.50	1.0893	PS100:MR0.70	0.4308	PS200:MR0.70	0.6207
PS400:MR0.70	1.0514	PS100:MR0.90	0.2880	PS200:MR0.90	0.5423
PS400:MR0.90	0.9445	Crsingle:MR0.05	-0.0348	Cruniform:MR0.05	-0.0993
Crsingle:MR0.10	-0.4892	Cruniform:MR0.10	-0.2756	Crsingle:MR0.20	-0.7342
Cruniform:MR0.20	-0.5717	Crsingle:MR0.50	-0.9949	Cruniform:MR0.50	-0.5581
Crsingle:MR0.70	-1.4153	Cruniform:MR0.70	-0.6337	Crsingle:MR0.90	-1.4267
Cruniform:MR0.90	-0.7264	PS100:Trlocal	0.1157	PS200:Trlocal	0.0801
PS400:Trlocal	0.2156	PS100:Trscale2	0.0059	PS200:Trscale2	0.0740
PS400:Trscale2	0.2542	PS100:Trscale4	0.0385	PS200:Trscale4	0.0830
PS400:Trscale4	0.3267	PS100:El1	-0.0013	PS200:El1	-0.0176
PS400:El1	0.0363	PS100:El5%	-0.0584	PS200:El5%	-0.1452
PS400:El5%	-0.1421	Trlocal:Crsingle	-0.0248	Trscale2:Crsingle	-0.1396
Trscale4:Crsingle	-0.0302	Trlocal:Cruniform	-0.0116	Trscale2:Cruniform	0.0181
cont.					

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
Trscale4:Cruniform	0.0399	El1:Crsingle	-0.0516	El5%:Crsingle	-0.0069
El1:Cruniform	-0.1070	El5%:Cruniform	-0.0852	Trlocal:MR0.05	0.2710
Trscale2:MR0.05	0.2853	Trscale4:MR0.05	0.2666	Trlocal:MR0.10	0.4613
Trscale2:MR0.10	0.3633	Trscale4:MR0.10	0.4504	Trlocal:MR0.20	0.7203
Trscale2:MR0.20	0.6231	Trscale4:MR0.20	0.7330	Trlocal:MR0.50	0.9926
Trscale2:MR0.50	0.8689	Trscale4:MR0.50	0.9441	Trlocal:MR0.70	1.1261
Trscale2:MR0.70	1.0173	Trscale4:MR0.70	1.1368	Trlocal:MR0.90	1.2712
Trscale2:MR0.90	1.2053	Trscale4:MR0.90	1.2705	El1:MR0.05	-0.1258
El5%:MR0.05	-0.0862	El1:MR0.10	-0.1865	El5%:MR0.10	-0.2508
El1:MR0.20	-0.2771	El5%:MR0.20	-0.4409	El1:MR0.50	-0.3001
El5%:MR0.50	-0.5506	El1:MR0.70	-0.4742	El5%:MR0.70	-0.6912
El1:MR0.90	-0.6190	El5%:MR0.90	-0.8497	Trlocal:El1	-0.2397
Trscale2:El1	-0.1863	Trscale4:El1	-0.1799	Trlocal:El5%	-0.4311
Trscale2:El5%	-0.4016	Trscale4:El5%	-0.3522	PS100:CP0.70	-0.0183
PS200:CP0.70	0.0002	PS400:CP0.70	0.0080	PS100:CP0.90	-0.0129
PS200:CP0.90	-0.0264	PS400:CP0.90	0.0248	Crsingle:CP0.70	-0.2394
Cruniform:CP0.70	-0.1990	Crsingle:CP0.90	-0.4469	Cruniform:CP0.90	-0.4137
PS100:Crsingle:MR0.05	-0.1735	PS200:Crsingle:MR0.05	-0.1910	PS400:Crsingle:MR0.05	-0.2218
PS100:Cruniform:MR0.05	-0.2362	PS200:Cruniform:MR0.05	-0.1668	PS400:Cruniform:MR0.05	-0.1826
PS100:Crsingle:MR0.10	-0.1872	PS200:Crsingle:MR0.10	-0.0768	PS400:Crsingle:MR0.10	-0.1443
PS100:Cruniform:MR0.10	-0.1475	PS200:Cruniform:MR0.10	-0.1054	PS400:Cruniform:MR0.10	-0.1215
PS100:Crsingle:MR0.20	-0.2512	PS200:Crsingle:MR0.20	-0.0478	PS400:Crsingle:MR0.20	-0.0864
PS100:Cruniform:MR0.20	-0.1295	PS200:Cruniform:MR0.20	-0.0176	PS400:Cruniform:MR0.20	0.0054
PS100:Crsingle:MR0.50	-0.0996	PS200:Crsingle:MR0.50	0.0002	PS400:Crsingle:MR0.50	0.1964
PS100:Cruniform:MR0.50	-0.2583	PS200:Cruniform:MR0.50	-0.1921	PS400:Cruniform:MR0.50	-0.1946
PS100:Crsingle:MR0.70	-0.0686	PS200:Crsingle:MR0.70	0.2049	PS400:Crsingle:MR0.70	0.1723
PS100:Cruniform:MR0.70	-0.2231	PS200:Cruniform:MR0.70	-0.1413	PS400:Cruniform:MR0.70	-0.1666
PS100:Crsingle:MR0.90	-0.0459	PS200:Crsingle:MR0.90	0.2150	PS400:Crsingle:MR0.90	0.2797
PS100:Cruniform:MR0.90	0.0141	PS200:Cruniform:MR0.90	0.0889	PS400:Cruniform:MR0.90	0.0318
PS100:Crsingle:Trlocal	-0.0891	PS200:Crsingle:Trlocal	0.0738	PS400:Crsingle:Trlocal	-0.0636
PS100:Cruniform:Trlocal	0.0053	PS200:Cruniform:Trlocal	0.0635	PS400:Cruniform:Trlocal	0.0783
PS100:Crsingle:Trscale2	0.0122	PS200:Crsingle:Trscale2	0.0851	PS400:Crsingle:Trscale2	-0.1260
PS100:Cruniform:Trscale2	0.0414	PS200:Cruniform:Trscale2	-0.0479	PS400:Cruniform:Trscale2	-0.0621
PS100:Crsingle:Trscale4	0.0085	PS200:Crsingle:Trscale4	0.0320	PS400:Crsingle:Trscale4	-0.1337
PS100:Cruniform:Trscale4	0.0358	PS200:Cruniform:Trscale4	0.0300	PS400:Cruniform:Trscale4	0.0502
Crsingle:MR0.05:Trlocal	-0.3072	Cruniform:MR0.05:Trlocal	-0.0300	Crsingle:MR0.10:Trlocal	-0.1748
Cruniform:MR0.10:Trlocal	-0.0980	Crsingle:MR0.20:Trlocal	-0.1357	Cruniform:MR0.20:Trlocal	-0.1029
Crsingle:MR0.50:Trlocal	-0.2732	Cruniform:MR0.50:Trlocal	-0.0217	Crsingle:MR0.70:Trlocal	-0.1557
Cruniform:MR0.70:Trlocal	0.1746	Crsingle:MR0.90:Trlocal	-0.2301	Cruniform:MR0.90:Trlocal	0.5176
Crsingle:MR0.05:Trscale2	-0.3513	Cruniform:MR0.05:Trscale2	-0.0449	Crsingle:MR0.10:Trscale2	-0.0937
Cruniform:MR0.10:Trscale2	-0.1241	Crsingle:MR0.20:Trscale2	-0.1335	Cruniform:MR0.20:Trscale2	-0.1913
Crsingle:MR0.50:Trscale2	-0.1873	Cruniform:MR0.50:Trscale2	-0.0619	Crsingle:MR0.70:Trscale2	-0.0771
Cruniform:MR0.70:Trscale2	0.0884	Crsingle:MR0.90:Trscale2	-0.1754	Cruniform:MR0.90:Trscale2	0.4398
Crsingle:MR0.05:Trscale4	-0.3559	Cruniform:MR0.05:Trscale4	-0.0728	Crsingle:MR0.10:Trscale4	-0.0822
Cruniform:MR0.10:Trscale4	-0.1476	Crsingle:MR0.20:Trscale4	-0.0013	Cruniform:MR0.20:Trscale4	-0.1287
Crsingle:MR0.50:Trscale4	-0.3236	Cruniform:MR0.50:Trscale4	-0.0099	Crsingle:MR0.70:Trscale4	-0.1825
Cruniform:MR0.70:Trscale4	0.1298	Crsingle:MR0.90:Trscale4	-0.2488	Cruniform:MR0.90:Trscale4	0.4637
Crsingle:MR0.05:El1	0.0440	Cruniform:MR0.05:El1	-0.0037	Crsingle:MR0.10:El1	0.0122
Cruniform:MR0.10:El1	-0.0475	Crsingle:MR0.20:El1	-0.0588	Cruniform:MR0.20:El1	-0.0099
Crsingle:MR0.50:El1	-0.0849	Cruniform:MR0.50:El1	-0.2352	Crsingle:MR0.70:El1	0.1683
Cruniform:MR0.70:El1	-0.4336	Crsingle:MR0.90:El1	0.0282	Cruniform:MR0.90:El1	-0.9027
Crsingle:MR0.05:El5%	-0.0271	Cruniform:MR0.05:El5%	-0.0446	Crsingle:MR0.10:El5%	-0.0341
Cruniform:MR0.10:El5%	-0.0483	Crsingle:MR0.20:El5%	-0.0015	Cruniform:MR0.20:El5%	0.0380
Crsingle:MR0.50:El5%	-0.0278	Cruniform:MR0.50:El5%	-0.2005	Crsingle:MR0.70:El5%	0.1161
Cruniform:MR0.70:El5%	-0.4089	Crsingle:MR0.90:El5%	0.0956	Cruniform:MR0.90:El5%	-0.9237
PS100:MR0.05:Trlocal	0.0822	PS200:MR0.05:Trlocal	0.2034	PS400:MR0.05:Trlocal	-0.0479
PS100:MR0.10:Trlocal	0.0412	PS200:MR0.10:Trlocal	0.0364	PS400:MR0.10:Trlocal	-0.0523
PS100:MR0.20:Trlocal	-0.1589	PS200:MR0.20:Trlocal	-0.1265	PS400:MR0.20:Trlocal	-0.2460
PS100:MR0.50:Trlocal	-0.2276	PS200:MR0.50:Trlocal	-0.2271	PS400:MR0.50:Trlocal	-0.3493
PS100:MR0.70:Trlocal	-0.2277	PS200:MR0.70:Trlocal	-0.3095	PS400:MR0.70:Trlocal	-0.4269

cont.

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
PS100:MR0.90:Trlocal	-0.1573	PS200:MR0.90:Trlocal	-0.2880	PS400:MR0.90:Trlocal	-0.4557
PS100:MR0.05:Trscale2	0.0896	PS200:MR0.05:Trscale2	0.2031	PS400:MR0.05:Trscale2	-0.0160
PS100:MR0.10:Trscale2	0.1346	PS200:MR0.10:Trscale2	0.1788	PS400:MR0.10:Trscale2	0.0691
PS100:MR0.20:Trscale2	-0.0456	PS200:MR0.20:Trscale2	0.0593	PS400:MR0.20:Trscale2	-0.1186
PS100:MR0.50:Trscale2	-0.1927	PS200:MR0.50:Trscale2	-0.0943	PS400:MR0.50:Trscale2	-0.1999
PS100:MR0.70:Trscale2	-0.1313	PS200:MR0.70:Trscale2	-0.1117	PS400:MR0.70:Trscale2	-0.2947
PS100:MR0.90:Trscale2	-0.0845	PS200:MR0.90:Trscale2	-0.2116	PS400:MR0.90:Trscale2	-0.3642
PS100:MR0.05:Trscale4	0.1817	PS200:MR0.05:Trscale4	0.2702	PS400:MR0.05:Trscale4	-0.0490
PS100:MR0.10:Trscale4	0.0982	PS200:MR0.10:Trscale4	0.1248	PS400:MR0.10:Trscale4	-0.0829
PS100:MR0.20:Trscale4	-0.1450	PS200:MR0.20:Trscale4	-0.0601	PS400:MR0.20:Trscale4	-0.3241
PS100:MR0.50:Trscale4	-0.1354	PS200:MR0.50:Trscale4	-0.1367	PS400:MR0.50:Trscale4	-0.3680
PS100:MR0.70:Trscale4	-0.1359	PS200:MR0.70:Trscale4	-0.2443	PS400:MR0.70:Trscale4	-0.4919
PS100:MR0.90:Trscale4	-0.0686	PS200:MR0.90:Trscale4	-0.2152	PS400:MR0.90:Trscale4	-0.5424
PS100:MR0.05:El1	-0.0294	PS200:MR0.05:El1	0.0791	PS400:MR0.05:El1	0.1035
PS100:MR0.10:El1	0.0792	PS200:MR0.10:El1	0.1435	PS400:MR0.10:El1	0.1338
PS100:MR0.20:El1	0.1488	PS200:MR0.20:El1	0.2371	PS400:MR0.20:El1	0.2480
PS100:MR0.50:El1	0.0350	PS200:MR0.50:El1	0.3079	PS400:MR0.50:El1	0.2706
PS100:MR0.70:El1	0.2255	PS200:MR0.70:El1	0.4302	PS400:MR0.70:El1	0.3955
PS100:MR0.90:El1	0.3383	PS200:MR0.90:El1	0.4833	PS400:MR0.90:El1	0.5515
PS100:MR0.05:El5%	-0.1229	PS200:MR0.05:El5%	-0.0526	PS400:MR0.05:El5%	-0.0509
PS100:MR0.10:El5%	0.0301	PS200:MR0.10:El5%	0.0941	PS400:MR0.10:El5%	0.0959
PS100:MR0.20:El5%	0.1688	PS200:MR0.20:El5%	0.2601	PS400:MR0.20:El5%	0.2687
PS100:MR0.50:El5%	0.1268	PS200:MR0.50:El5%	0.3426	PS400:MR0.50:El5%	0.3645
PS100:MR0.70:El5%	0.2596	PS200:MR0.70:El5%	0.3916	PS400:MR0.70:El5%	0.4028
PS100:MR0.90:El5%	0.2654	PS200:MR0.90:El5%	0.3992	PS400:MR0.90:El5%	0.4825

Table 33: Coefficients for the time model for group-number GCAs on Ruspini2.

A.2 Group-number and Towns2

Table 34 lists the coefficients for the correctness model for the Towns2 data set. The residual deviance of this model (3493.9 on 2634 degrees of freedom) is large, suggesting that the model does not fit the data well. The histograms in Figure 38 confirm this. The addition of higher order terms may improve the fit of the model, although initial attempts at this failed to do so. Smoothing techniques may be more appropriate for the analysis of this data.

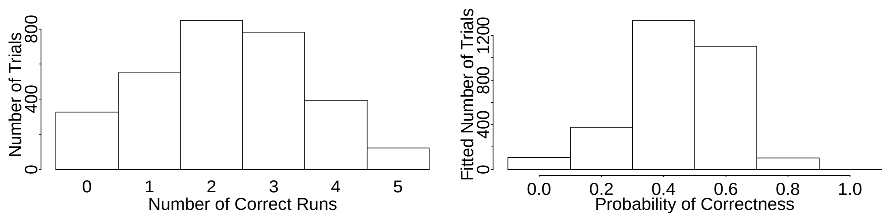


Figure 38: Comparison of experimental and fitted correctness for group-number GCAs on Towns2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	0.0163	PS100	0.0592	PS200	-0.4136
PS400	-1.2579	Trlocal	-0.4924	Trscale2	-0.0251
Trscale4	0.0649	El1	-0.6090	El5%	-0.2441
Crsingle	-0.6793	Cruniform	-0.7935	CP0.70	-0.1819
cont.					

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
CP0.90	0.0790	MR0.05	0.4244	MR0.10	-0.0872
MR0.20	0.2146	MR0.50	-0.0274	MR0.70	0.2263
MR0.90	-0.3584	PS100:Trlocal	-0.0570	PS200:Trlocal	-0.3305
PS400:Trlocal	-0.2988	PS100:Trscale2	-0.7732	PS200:Trscale2	-0.2080
PS400:Trscale2	-1.5268	PS100:Trscale4	-0.3497	PS200:Trscale4	-0.2279
PS400:Trscale4	-1.1307	PS100:El1	-0.0702	PS200:El1	0.4306
PS400:El1	0.8390	PS100:El5%	-0.3911	PS200:El5%	0.6444
PS400:El5%	0.9410	PS100:Crsingle	0.2175	PS200:Crsingle	0.5694
PS400:Crsingle	1.4896	PS100:Cruniform	-0.1638	PS200:Cruniform	-0.2041
PS400:Cruniform	0.6541	PS100:CP0.70	0.0754	PS200:CP0.70	0.2575
PS400:CP0.70	-0.0134	PS100:CP0.90	-0.0679	PS200:CP0.90	-0.3923
PS400:CP0.90	0.0371	PS100:MR0.05	-0.1367	PS200:MR0.05	-0.2002
PS400:MR0.05	0.0418	PS100:MR0.10	0.3696	PS200:MR0.10	0.0084
PS400:MR0.10	0.6928	PS100:MR0.20	-0.2155	PS200:MR0.20	-0.1927
PS400:MR0.20	0.8380	PS100:MR0.50	-0.0929	PS200:MR0.50	0.1983
PS400:MR0.50	0.9534	PS100:MR0.70	-0.0410	PS200:MR0.70	-0.0272
PS400:MR0.70	0.2401	PS100:MR0.90	0.3261	PS200:MR0.90	0.5527
PS400:MR0.90	0.2470	Trlocal:El1	1.2626	Trscale2:El1	1.3048
Trscale4:El1	0.8623	Trlocal:El5%	0.5797	Trscale2:El5%	0.2377
Trscale4:El5%	0.4699	Trlocal:Crsingle	0.5308	Trscale2:Crsingle	1.1076
Trscale4:Crsingle	0.2667	Trlocal:Cruniform	0.3639	Trscale2:Cruniform	0.3767
Trscale4:Cruniform	0.7979	Trlocal:CP0.70	0.0101	Trscale2:CP0.70	-0.3035
Trscale4:CP0.70	0.2550	Trlocal:CP0.90	0.0152	Trscale2:CP0.90	-0.3099
Trscale4:CP0.90	0.1515	Trlocal:MR0.05	-0.2313	Trscale2:MR0.05	-0.6699
Trscale4:MR0.05	-1.4110	Trlocal:MR0.10	0.1214	Trscale2:MR0.10	-0.4448
Trscale4:MR0.10	-0.9189	Trlocal:MR0.20	0.1059	Trscale2:MR0.20	-0.3195
Trscale4:MR0.20	-0.8365	Trlocal:MR0.50	-0.1956	Trscale2:MR0.50	-0.4191
Trscale4:MR0.50	-1.0612	Trlocal:MR0.70	-1.3977	Trscale2:MR0.70	-1.3987
Trscale4:MR0.70	-1.9714	Trlocal:MR0.90	-1.6628	Trscale2:MR0.90	-2.8230
Trscale4:MR0.90	-2.9995	El1:Crsingle	0.7620	El5%:Crsingle	1.1177
El1:Cruniform	0.7530	El5%:Cruniform	1.2061	El1:CP0.70	0.2190
El5%:CP0.70	0.3604	El1:CP0.90	0.1442	El5%:CP0.90	0.2142
El1:MR0.05	-0.1687	El5%:MR0.05	-0.0381	El1:MR0.10	0.2568
El5%:MR0.10	-0.5664	El1:MR0.20	0.0016	El5%:MR0.20	-0.1489
El1:MR0.50	0.0738	El5%:MR0.50	-0.0242	El1:MR0.70	0.8053
El5%:MR0.70	0.0929	El1:MR0.90	1.1449	El5%:MR0.90	0.9163
Crsingle:MR0.05	-0.6166	Cruniform:MR0.05	0.1206	Crsingle:MR0.10	-0.2023
Cruniform:MR0.10	0.5849	Crsingle:MR0.20	-0.0357	Cruniform:MR0.20	0.4234
Crsingle:MR0.50	-0.5131	Cruniform:MR0.50	0.5680	Crsingle:MR0.70	0.5162
Cruniform:MR0.70	1.0597	Crsingle:MR0.90	1.0559	Cruniform:MR0.90	1.5341
PS100:Trlocal:El1	0.0916	PS200:Trlocal:El1	-0.1509	PS400:Trlocal:El1	-0.0330
PS100:Trscale2:El1	-0.0257	PS200:Trscale2:El1	-0.1329	PS400:Trscale2:El1	-0.0945
PS100:Trscale4:El1	-0.0774	PS200:Trscale4:El1	-0.1183	PS400:Trscale4:El1	0.2117
PS100:Trlocal:El5%	-0.1157	PS200:Trlocal:El5%	-0.0203	PS400:Trlocal:El5%	1.1924
PS100:Trscale2:El5%	0.0316	PS200:Trscale2:El5%	0.3178	PS400:Trscale2:El5%	1.3507
PS100:Trscale4:El5%	0.1628	PS200:Trscale4:El5%	-0.3612	PS400:Trscale4:El5%	1.3883
PS100:Trlocal:CP0.70	-0.1877	PS200:Trlocal:CP0.70	0.1399	PS400:Trlocal:CP0.70	-0.2312
PS100:Trscale2:CP0.70	0.3627	PS200:Trscale2:CP0.70	0.5063	PS400:Trscale2:CP0.70	0.3807
PS100:Trscale4:CP0.70	-0.6548	PS200:Trscale4:CP0.70	-0.1328	PS400:Trscale4:CP0.70	0.0151
PS100:Trlocal:CP0.90	-0.3893	PS200:Trlocal:CP0.90	0.4392	PS400:Trlocal:CP0.90	-0.4901
PS100:Trscale2:CP0.90	0.5204	PS200:Trscale2:CP0.90	0.6193	PS400:Trscale2:CP0.90	0.0362
PS100:Trscale4:CP0.90	-0.5900	PS200:Trscale4:CP0.90	0.3665	PS400:Trscale4:CP0.90	-0.6317
PS100:Trlocal:MR0.05	-0.0692	PS200:Trlocal:MR0.05	0.4122	PS400:Trlocal:MR0.05	0.6231
PS100:Trscale2:MR0.05	-0.1647	PS200:Trscale2:MR0.05	-0.2042	PS400:Trscale2:MR0.05	0.9546
PS100:Trscale4:MR0.05	0.6849	PS200:Trscale4:MR0.05	0.4664	PS400:Trscale4:MR0.05	0.7748
PS100:Trlocal:MR0.10	0.0807	PS200:Trlocal:MR0.10	0.4725	PS400:Trlocal:MR0.10	0.2655
PS100:Trscale2:MR0.10	0.2638	PS200:Trscale2:MR0.10	-0.2469	PS400:Trscale2:MR0.10	0.7317
PS100:Trscale4:MR0.10	0.7133	PS200:Trscale4:MR0.10	0.9757	PS400:Trscale4:MR0.10	0.7997
PS100:Trlocal:MR0.20	0.3592	PS200:Trlocal:MR0.20	0.2528	PS400:Trlocal:MR0.20	-0.5889
PS100:Trscale2:MR0.20	0.5722	PS200:Trscale2:MR0.20	-0.4241	PS400:Trscale2:MR0.20	0.3746

cont.

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
PS100:Trscale4:MR0.20	0.8317	PS200:Trscale4:MR0.20	0.4785	PS400:Trscale4:MR0.20	0.2606
PS100:Trlocal:MR0.50	-0.0943	PS200:Trlocal:MR0.50	0.0868	PS400:Trlocal:MR0.50	-1.1640
PS100:Trscale2:MR0.50	0.6654	PS200:Trscale2:MR0.50	-0.3871	PS400:Trscale2:MR0.50	0.0265
PS100:Trscale4:MR0.50	1.1694	PS200:Trscale4:MR0.50	-0.0237	PS400:Trscale4:MR0.50	-0.2410
PS100:Trlocal:MR0.70	0.0625	PS200:Trlocal:MR0.70	-0.0883	PS400:Trlocal:MR0.70	-0.7016
PS100:Trscale2:MR0.70	0.0708	PS200:Trscale2:MR0.70	-1.2915	PS400:Trscale2:MR0.70	-0.6538
PS100:Trscale4:MR0.70	0.5010	PS200:Trscale4:MR0.70	-0.1179	PS400:Trscale4:MR0.70	-0.5025
PS100:Trlocal:MR0.90	0.2524	PS200:Trlocal:MR0.90	-0.4611	PS400:Trlocal:MR0.90	-1.0660
PS100:Trscale2:MR0.90	0.5682	PS200:Trscale2:MR0.90	-0.8716	PS400:Trscale2:MR0.90	-0.6303
PS100:Trscale4:MR0.90	0.7817	PS200:Trscale4:MR0.90	-0.5102	PS400:Trscale4:MR0.90	-1.1989
PS100:El1:Crsingle	-0.4868	PS200:El1:Crsingle	-0.0064	PS400:El1:Crsingle	-0.0652
PS100:El5%:Crsingle	-0.0271	PS200:El5%:Crsingle	-0.5608	PS400:El5%:Crsingle	-1.5655
PS100:El1:Cruniform	-0.3206	PS200:El1:Cruniform	0.4398	PS400:El1:Cruniform	0.4516
PS100:El5%:Cruniform	-0.1253	PS200:El5%:Cruniform	0.0455	PS400:El5%:Cruniform	-1.0153
PS100:El1:CP0.70	0.3798	PS200:El1:CP0.70	-0.7366	PS400:El1:CP0.70	-0.2884
PS100:El5%:CP0.70	0.1269	PS200:El5%:CP0.70	-0.6140	PS400:El5%:CP0.70	0.0315
PS100:El1:CP0.90	-0.0230	PS200:El1:CP0.90	-0.3499	PS400:El1:CP0.90	-0.2389
PS100:El5%:CP0.90	0.2772	PS200:El5%:CP0.90	-0.0042	PS400:El5%:CP0.90	-0.0613
PS100:El1:MR0.05	0.6082	PS200:El1:MR0.05	0.4234	PS400:El1:MR0.05	-0.3330
PS100:El5%:MR0.05	0.9577	PS200:El5%:MR0.05	0.2178	PS400:El5%:MR0.05	-0.1702
PS100:El1:MR0.10	-0.0485	PS200:El1:MR0.10	0.2921	PS400:El1:MR0.10	-0.6182
PS100:El5%:MR0.10	0.0624	PS200:El5%:MR0.10	-0.0795	PS400:El5%:MR0.10	-0.7380
PS100:El1:MR0.20	0.1810	PS200:El1:MR0.20	0.8461	PS400:El1:MR0.20	-0.5836
PS100:El5%:MR0.20	0.1940	PS200:El5%:MR0.20	0.5061	PS400:El5%:MR0.20	-0.3672
PS100:El1:MR0.50	0.9379	PS200:El1:MR0.50	0.6032	PS400:El1:MR0.50	-0.1090
PS100:El5%:MR0.50	0.7728	PS200:El5%:MR0.50	0.2042	PS400:El5%:MR0.50	-0.0158
PS100:El1:MR0.70	0.3750	PS200:El1:MR0.70	-0.1725	PS400:El1:MR0.70	-0.2501
PS100:El5%:MR0.70	0.0527	PS200:El5%:MR0.70	0.5216	PS400:El5%:MR0.70	0.2961
PS100:El1:MR0.90	0.0293	PS200:El1:MR0.90	-0.5382	PS400:El1:MR0.90	0.2145
PS100:El5%:MR0.90	-0.0658	PS200:El5%:MR0.90	-0.1129	PS400:El5%:MR0.90	0.1263
Trlocal:El1:Crsingle	-1.2683	Trscale2:El1:Crsingle	-1.2155	Trscale4:El1:Crsingle	-0.8199
Trlocal:El5%:Crsingle	-1.4448	Trscale2:El5%:Crsingle	-1.4345	Trscale4:El5%:Crsingle	-1.1153
Trlocal:El1:Cruniform	-0.6899	Trscale2:El1:Cruniform	-0.8144	Trscale4:El1:Cruniform	-0.9308
Trlocal:El5%:Cruniform	-0.7348	Trscale2:El5%:Cruniform	-0.7863	Trscale4:El5%:Cruniform	-1.1547
Trlocal:El1:MR0.05	-0.2528	Trscale2:El1:MR0.05	-0.1546	Trscale4:El1:MR0.05	0.4280
Trlocal:El5%:MR0.05	-0.7807	Trscale2:El5%:MR0.05	0.0085	Trscale4:El5%:MR0.05	0.1764
Trlocal:El1:MR0.10	0.0730	Trscale2:El1:MR0.10	-0.2196	Trscale4:El1:MR0.10	0.1891
Trlocal:El5%:MR0.10	0.4752	Trscale2:El5%:MR0.10	1.0540	Trscale4:El5%:MR0.10	0.4916
Trlocal:El1:MR0.20	0.1910	Trscale2:El1:MR0.20	-0.2181	Trscale4:El1:MR0.20	-0.0078
Trlocal:El5%:MR0.20	0.2562	Trscale2:El5%:MR0.20	0.3671	Trscale4:El5%:MR0.20	0.3862
Trlocal:El1:MR0.50	0.0733	Trscale2:El1:MR0.50	-0.0055	Trscale4:El1:MR0.50	0.2469
Trlocal:El5%:MR0.50	0.5409	Trscale2:El5%:MR0.50	0.5119	Trscale4:El5%:MR0.50	0.3165
Trlocal:El1:MR0.70	0.5823	Trscale2:El1:MR0.70	0.9482	Trscale4:El1:MR0.70	0.9889
Trlocal:El5%:MR0.70	1.5940	Trscale2:El5%:MR0.70	1.8066	Trscale4:El5%:MR0.70	1.6550
Trlocal:El1:MR0.90	0.7059	Trscale2:El1:MR0.90	1.5149	Trscale4:El1:MR0.90	1.4046
Trlocal:El5%:MR0.90	1.8066	Trscale2:El5%:MR0.90	2.4345	Trscale4:El5%:MR0.90	2.5292
Trlocal:Crsingle:MR0.05	1.0303	Trscale2:Crsingle:MR0.05	0.6468	Trscale4:Crsingle:MR0.05	1.1297
Trlocal:Cruniform:MR0.05	0.1603	Trscale2:Cruniform:MR0.05	0.6945	Trscale4:Cruniform:MR0.05	0.4819
Trlocal:Crsingle:MR0.10	-0.0174	Trscale2:Crsingle:MR0.10	-0.2466	Trscale4:Crsingle:MR0.10	0.2421
Trlocal:Cruniform:MR0.10	-0.4287	Trscale2:Cruniform:MR0.10	0.0975	Trscale4:Cruniform:MR0.10	-0.1205
Trlocal:Crsingle:MR0.20	0.3442	Trscale2:Crsingle:MR0.20	-0.1343	Trscale4:Crsingle:MR0.20	0.5397
Trlocal:Cruniform:MR0.20	0.0809	Trscale2:Cruniform:MR0.20	0.3830	Trscale4:Cruniform:MR0.20	0.3837
Trlocal:Crsingle:MR0.50	1.2783	Trscale2:Crsingle:MR0.50	0.5028	Trscale4:Crsingle:MR0.50	1.1198
Trlocal:Cruniform:MR0.50	0.4900	Trscale2:Cruniform:MR0.50	0.1254	Trscale4:Cruniform:MR0.50	0.4454
Trlocal:Crsingle:MR0.70	1.6183	Trscale2:Crsingle:MR0.70	0.8683	Trscale4:Crsingle:MR0.70	1.0846
Trlocal:Cruniform:MR0.70	0.4110	Trscale2:Cruniform:MR0.70	0.4336	Trscale4:Cruniform:MR0.70	0.2885
Trlocal:Crsingle:MR0.90	1.3624	Trscale2:Crsingle:MR0.90	1.6787	Trscale4:Crsingle:MR0.90	2.1103
Trlocal:Cruniform:MR0.90	0.6762	Trscale2:Cruniform:MR0.90	1.0238	Trscale4:Cruniform:MR0.90	0.6941
El1:Crsingle:MR0.05	0.1553	El5%:Crsingle:MR0.05	-0.0662	El1:Cruniform:MR0.05	-0.2909
El5%:Cruniform:MR0.05	-0.4194	El1:Crsingle:MR0.10	0.2435	El5%:Crsingle:MR0.10	0.4049

cont.

cont. Covariate	Effect	Covariate	Effect	Covariate	Effect
El1:Cruniform:MR0.10	-0.6569	El5%:Cruniform:MR0.10	-0.2549	El1:Crsingle:MR0.20	-0.0942
El5%:Crsingle:MR0.20	-0.3240	El1:Cruniform:MR0.20	-0.3865	El5%:Cruniform:MR0.20	-0.9746
El1:Crsingle:MR0.50	-0.3585	El5%:Crsingle:MR0.50	-0.2245	El1:Cruniform:MR0.50	-0.6551
El5%:Cruniform:MR0.50	-1.1894	El1:Crsingle:MR0.70	-1.1167	El5%:Crsingle:MR0.70	-1.7169
El1:Cruniform:MR0.70	-1.1085	El5%:Cruniform:MR0.70	-1.6653	El1:Crsingle:MR0.90	-1.7653
El5%:Crsingle:MR0.90	-2.6483	El1:Cruniform:MR0.90	-1.8884	El5%:Cruniform:MR0.90	-2.4812

Table 34: Coefficients for the correctness model for group-number GCAs on Towns2.

The fit of time model for the Town2 data is also questionable due to a high residual deviance (22985 on 2073 degrees of freedom). The range of time values predicted by this model to not match the experimental values (as can be seen in Figure 39). The residual plot shows a definite change in the variance of the residuals over the fitted scale (Figure 40). This model includes all first, second and third order terms which suggests the addition of higher order terms may improve the fit. The coefficients for this model are given in Table 35.

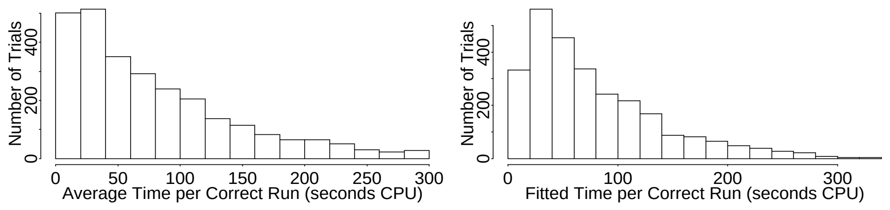


Figure 39: Comparison of experimental and fitted time for group-number GCAs on Towns2.

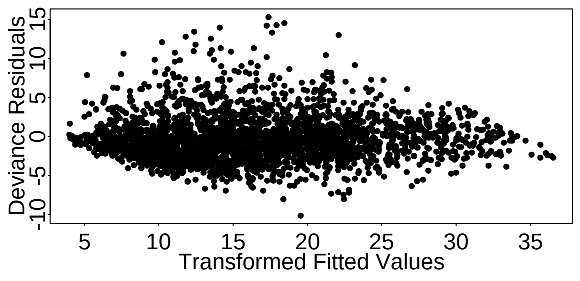


Figure 40: Residual plot for time model for group-number GCAs on Towns2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	4.8261	PS100	0.0704	PS200	-0.0492
PS400	0.3912	Trlocal	0.0637	Trscale2	0.1773
Trscale4	-0.1928	El1	-0.1370	El5%	-0.1151
Crsingle	-0.8216	Cruniform	-0.3055	CP0.70	-0.1627
CP0.90	-0.1516	MR0.05	-1.2388	MR0.10	-1.3392
MR0.20	-1.6200	MR0.50	-1.5342	MR0.70	-1.5299
MR0.90	-1.4749	PS100:Trlocal	0.2025	PS200:Trlocal	0.2561
PS400:Trlocal	0.3257	PS100:Trscale2	-0.1432	PS200:Trscale2	0.2271
PS400:Trscale2	0.4857	PS100:Trscale4	0.0536	PS200:Trscale4	0.4134
PS400:Trscale4	0.4241	PS100:El1	-0.2079	PS200:El1	-0.2430
cont.					

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
PS400:El1	-0.3495	PS100:El5%	-0.1150	PS200:El5%	-0.0382
PS400:El5%	-0.2306	PS100:Crsingle	0.2071	PS200:Crsingle	0.2455
PS400:Crsingle	0.3135	PS100:Cruniform	-0.0511	PS200:Cruniform	0.2417
PS400:Cruniform	0.2144	PS100:CP0.70	0.0163	PS200:CP0.70	0.4829
PS400:CP0.70	0.2968	PS100:CP0.90	0.1011	PS200:CP0.90	0.5309
PS400:CP0.90	0.3478	PS100:MR0.05	0.2609	PS200:MR0.05	0.2836
PS400:MR0.05	0.2483	PS100:MR0.10	0.2560	PS200:MR0.10	0.6391
PS400:MR0.10	0.5631	PS100:MR0.20	0.5402	PS200:MR0.20	0.3580
PS400:MR0.20	0.7901	PS100:MR0.50	0.2455	PS200:MR0.50	0.2847
PS400:MR0.50	0.9076	PS100:MR0.70	0.0337	PS200:MR0.70	0.5842
PS400:MR0.70	0.5966	PS100:MR0.90	0.5928	PS200:MR0.90	0.8055
PS400:MR0.90	0.4494	Trlocal:El1	-0.1407	Trscale2:El1	-0.1339
Trscale4:El1	0.1309	Trlocal:El5%	-0.2194	Trscale2:El5%	-0.0854
Trscale4:El5%	-0.0649	Trlocal:Crsingle	0.5365	Trscale2:Crsingle	0.3135
Trscale4:Crsingle	0.5245	Trlocal:Cruniform	0.1651	Trscale2:Cruniform	0.1060
Trscale4:Cruniform	0.0900	Trlocal:CP0.70	0.1251	Trscale2:CP0.70	0.1645
Trscale4:CP0.70	0.0772	Trlocal:CP0.90	0.1797	Trscale2:CP0.90	0.0701
Trscale4:CP0.90	0.2878	Trlocal:MR0.05	0.6037	Trscale2:MR0.05	0.4086
Trscale4:MR0.05	0.5882	Trlocal:MR0.10	0.9162	Trscale2:MR0.10	0.4879
Trscale4:MR0.10	1.1464	Trlocal:MR0.20	1.2002	Trscale2:MR0.20	0.7018
Trscale4:MR0.20	1.1365	Trlocal:MR0.50	1.6268	Trscale2:MR0.50	1.4779
Trscale4:MR0.50	1.6248	Trlocal:MR0.70	2.1103	Trscale2:MR0.70	2.0117
Trscale4:MR0.70	2.1821	Trlocal:MR0.90	1.7292	Trscale2:MR0.90	1.6623
Trscale4:MR0.90	2.3046	El1:Crsingle	-0.1959	El5%:Crsingle	0.0475
El1:Cruniform	0.1395	El5%:Cruniform	0.1241	El1:CP0.70	0.0474
El5%:CP0.70	0.0181	El1:CP0.90	0.1311	El5%:CP0.90	0.1433
El1:MR0.05	0.1543	El5%:MR0.05	-0.2669	El1:MR0.10	-0.6172
El5%:MR0.10	-1.0622	El1:MR0.20	-0.0080	El5%:MR0.20	-0.0402
El1:MR0.50	-0.5907	El5%:MR0.50	-0.8498	El1:MR0.70	-0.3675
El5%:MR0.70	-0.7265	El1:MR0.90	0.1303	El5%:MR0.90	-0.5188
Crsingle:CP0.70	0.2818	Cruniform:CP0.70	-0.1676	Crsingle:CP0.90	0.1745
Cruniform:CP0.90	0.0553	Crsingle:MR0.05	0.2987	Cruniform:MR0.05	0.7027
Crsingle:MR0.10	0.5318	Cruniform:MR0.10	0.1743	Crsingle:MR0.20	0.2612
Cruniform:MR0.20	0.7599	Crsingle:MR0.50	-0.1652	Cruniform:MR0.50	0.2857
Crsingle:MR0.70	-0.1862	Cruniform:MR0.70	-0.1506	Crsingle:MR0.90	-0.0824
Cruniform:MR0.90	0.5263	CP0.70:MR0.05	0.0800	CP0.90:MR0.05	0.4124
CP0.70:MR0.10	0.2621	CP0.90:MR0.10	0.4495	CP0.70:MR0.20	0.3725
CP0.90:MR0.20	0.0539	CP0.70:MR0.50	0.1964	CP0.90:MR0.50	0.0915
CP0.70:MR0.70	0.1091	CP0.90:MR0.70	0.4189	CP0.70:MR0.90	0.4626
CP0.90:MR0.90	-0.0091	PS100:Trlocal:El1	-0.0462	PS200:Trlocal:El1	0.1628
PS400:Trlocal:El1	0.3141	PS100:Trscale2:El1	0.0024	PS200:Trscale2:El1	0.1332
PS400:Trscale2:El1	0.1038	PS100:Trscale4:El1	-0.1150	PS200:Trscale4:El1	0.0394
PS400:Trscale4:El1	0.2659	PS100:Trlocal:El5%	-0.1411	PS200:Trlocal:El5%	0.0076
PS400:Trlocal:El5%	0.0846	PS100:Trscale2:El5%	-0.2512	PS200:Trscale2:El5%	-0.1666
PS400:Trscale2:El5%	-0.2326	PS100:Trscale4:El5%	0.0314	PS200:Trscale4:El5%	-0.0891
PS400:Trscale4:El5%	-0.0575	PS100:Trlocal:Crsingle	-0.0482	PS200:Trlocal:Crsingle	-0.3027
PS400:Trlocal:Crsingle	-0.2827	PS100:Trscale2:Crsingle	-0.0033	PS200:Trscale2:Crsingle	-0.1604
PS400:Trscale2:Crsingle	-0.3755	PS100:Trscale4:Crsingle	-0.2041	PS200:Trscale4:Crsingle	-0.3237
PS400:Trscale4:Crsingle	-0.5469	PS100:Trlocal:Cruniform	0.2483	PS200:Trlocal:Cruniform	0.1478
PS400:Trlocal:Cruniform	0.1003	PS100:Trscale2:Cruniform	0.3226	PS200:Trscale2:Cruniform	0.0738
PS400:Trscale2:Cruniform	-0.1592	PS100:Trscale4:Cruniform	0.3954	PS200:Trscale4:Cruniform	0.1607
PS400:Trscale4:Cruniform	-0.0047	PS100:Trlocal:CP0.70	-0.1353	PS200:Trlocal:CP0.70	-0.3525
PS400:Trlocal:CP0.70	-0.1790	PS100:Trscale2:CP0.70	-0.0008	PS200:Trscale2:CP0.70	-0.3062
PS400:Trscale2:CP0.70	-0.0963	PS100:Trscale4:CP0.70	-0.0285	PS200:Trscale4:CP0.70	-0.3139
PS400:Trscale4:CP0.70	-0.0248	PS100:Trlocal:CP0.90	-0.1069	PS200:Trlocal:CP0.90	-0.1037
PS400:Trlocal:CP0.90	-0.0441	PS100:Trscale2:CP0.90	0.1084	PS200:Trscale2:CP0.90	-0.0397
PS400:Trscale2:CP0.90	0.0432	PS100:Trscale4:CP0.90	-0.0842	PS200:Trscale4:CP0.90	-0.1475
PS400:Trscale4:CP0.90	0.0429	PS100:Trlocal:MR0.05	0.0642	PS200:Trlocal:MR0.05	0.3815
PS400:Trlocal:MR0.05	0.1514	PS100:Trscale2:MR0.05	0.3857	PS200:Trscale2:MR0.05	0.3887
PS400:Trscale2:MR0.05	0.1590	PS100:Trscale4:MR0.05	0.2851	PS200:Trscale4:MR0.05	0.3632

cont.

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
PS400:Trscale4:MR0.05	0.2886	PS100:Trlocal:MR0.10	-0.0577	PS200:Trlocal:MR0.10	-0.1557
PS400:Trlocal:MR0.10	-0.1870	PS100:Trscale2:MR0.10	0.3891	PS200:Trscale2:MR0.10	0.1103
PS400:Trscale2:MR0.10	0.1904	PS100:Trscale4:MR0.10	-0.2982	PS200:Trscale4:MR0.10	-0.4066
PS400:Trscale4:MR0.10	-0.4717	PS100:Trlocal:MR0.20	-0.3867	PS200:Trlocal:MR0.20	-0.0497
PS400:Trlocal:MR0.20	-0.5962	PS100:Trscale2:MR0.20	0.1031	PS200:Trscale2:MR0.20	0.3800
PS400:Trscale2:MR0.20	-0.1839	PS100:Trscale4:MR0.20	-0.1883	PS200:Trscale4:MR0.20	0.2214
PS400:Trscale4:MR0.20	-0.2213	PS100:Trlocal:MR0.50	0.0973	PS200:Trlocal:MR0.50	0.0907
PS400:Trlocal:MR0.50	-0.5383	PS100:Trscale2:MR0.50	0.3241	PS200:Trscale2:MR0.50	0.2542
PS400:Trscale2:MR0.50	-0.3400	PS100:Trscale4:MR0.50	0.3362	PS200:Trscale4:MR0.50	-0.0035
PS400:Trscale4:MR0.50	-0.4269	PS100:Trlocal:MR0.70	-0.1460	PS200:Trlocal:MR0.70	-0.2299
PS400:Trlocal:MR0.70	-0.7035	PS100:Trscale2:MR0.70	0.2924	PS200:Trscale2:MR0.70	0.1463
PS400:Trscale2:MR0.70	-0.5560	PS100:Trscale4:MR0.70	0.1254	PS200:Trscale4:MR0.70	-0.1765
PS400:Trscale4:MR0.70	-0.6106	PS100:Trlocal:MR0.90	0.0431	PS200:Trlocal:MR0.90	-0.3133
PS400:Trlocal:MR0.90	-0.2514	PS100:Trscale2:MR0.90	0.2298	PS200:Trscale2:MR0.90	-0.2393
PS400:Trscale2:MR0.90	0.0737	PS100:Trscale4:MR0.90	-0.0386	PS200:Trscale4:MR0.90	-0.5498
PS400:Trscale4:MR0.90	0.0773	PS100:El1:Crsingle	0.3495	PS200:El1:Crsingle	0.2748
PS400:El1:Crsingle	0.3952	PS100:El5%:Crsingle	0.1770	PS200:El5%:Crsingle	0.0932
PS400:El5%:Crsingle	-0.0059	PS100:El1:Cruniform	0.0682	PS200:El1:Cruniform	-0.1105
PS400:El1:Cruniform	0.1382	PS100:El5%:Cruniform	0.1144	PS200:El5%:Cruniform	-0.0035
PS400:El5%:Cruniform	0.0217	PS100:El1:CP0.70	0.1619	PS200:El1:CP0.70	-0.0583
PS400:El1:CP0.70	-0.0476	PS100:El5%:CP0.70	0.0145	PS200:El5%:CP0.70	-0.2614
PS400:El5%:CP0.70	-0.1765	PS100:El1:CP0.90	0.0641	PS200:El1:CP0.90	-0.0488
PS400:El1:CP0.90	-0.1906	PS100:El5%:CP0.90	-0.1679	PS200:El5%:CP0.90	-0.3572
PS400:El5%:CP0.90	-0.2069	PS100:El1:MR0.05	-0.2520	PS200:El1:MR0.05	-0.1123
PS400:El1:MR0.05	0.1063	PS100:El5%:MR0.05	-0.0057	PS200:El5%:MR0.05	0.1329
PS400:El5%:MR0.05	0.3863	PS100:El1:MR0.10	0.5227	PS200:El1:MR0.10	1.0112
PS400:El1:MR0.10	0.9922	PS100:El5%:MR0.10	0.7703	PS200:El5%:MR0.10	1.0136
PS400:El5%:MR0.10	0.9113	PS100:El1:MR0.20	0.4139	PS200:El1:MR0.20	0.6507
PS400:El1:MR0.20	0.7819	PS100:El5%:MR0.20	0.2971	PS200:El5%:MR0.20	0.6330
PS400:El5%:MR0.20	0.7636	PS100:El1:MR0.50	0.4170	PS200:El1:MR0.50	0.9139
PS400:El1:MR0.50	0.8368	PS100:El5%:MR0.50	0.4564	PS200:El5%:MR0.50	0.8843
PS400:El5%:MR0.50	0.9183	PS100:El1:MR0.70	0.5567	PS200:El1:MR0.70	0.5661
PS400:El1:MR0.70	1.1095	PS100:El5%:MR0.70	0.6051	PS200:El5%:MR0.70	0.6518
PS400:El5%:MR0.70	1.1223	PS100:El1:MR0.90	0.2449	PS200:El1:MR0.90	0.3150
PS400:El1:MR0.90	0.5996	PS100:El5%:MR0.90	-0.0188	PS200:El5%:MR0.90	0.2850
PS400:El5%:MR0.90	0.6141	PS100:Crsingle:CP0.70	-0.0904	PS200:Crsingle:CP0.70	-0.0921
PS400:Crsingle:CP0.70	-0.1444	PS100:Cruniform:CP0.70	0.0941	PS200:Cruniform:CP0.70	-0.0982
PS400:Cruniform:CP0.70	-0.2333	PS100:Crsingle:CP0.90	-0.2016	PS200:Crsingle:CP0.90	-0.3046
PS400:Crsingle:CP0.90	-0.3501	PS100:Cruniform:CP0.90	-0.0389	PS200:Cruniform:CP0.90	-0.3279
PS400:Cruniform:CP0.90	-0.3874	PS100:Crsingle:MR0.05	-0.2860	PS200:Crsingle:MR0.05	-0.2127
PS400:Crsingle:MR0.05	-0.2852	PS100:Cruniform:MR0.05	-0.4600	PS200:Cruniform:MR0.05	-0.3675
PS400:Cruniform:MR0.05	-0.2699	PS100:Crsingle:MR0.10	-0.5361	PS200:Crsingle:MR0.10	-0.8165
PS400:Crsingle:MR0.10	-0.6451	PS100:Cruniform:MR0.10	-0.5231	PS200:Cruniform:MR0.10	-0.6083
PS400:Cruniform:MR0.10	-0.4318	PS100:Crsingle:MR0.20	-1.0316	PS200:Crsingle:MR0.20	-1.0198
PS400:Crsingle:MR0.20	-0.9058	PS100:Cruniform:MR0.20	-0.9619	PS200:Cruniform:MR0.20	-0.8419
PS400:Cruniform:MR0.20	-0.7767	PS100:Crsingle:MR0.50	-0.1961	PS200:Crsingle:MR0.50	0.0504
PS400:Crsingle:MR0.50	-0.1088	PS100:Cruniform:MR0.50	-0.4037	PS200:Cruniform:MR0.50	-0.3580
PS400:Cruniform:MR0.50	-0.6042	PS100:Crsingle:MR0.70	-0.2065	PS200:Crsingle:MR0.70	0.1391
PS400:Crsingle:MR0.70	0.0995	PS100:Cruniform:MR0.70	-0.0610	PS200:Cruniform:MR0.70	-0.2957
PS400:Cruniform:MR0.70	-0.2202	PS100:Crsingle:MR0.90	-0.3472	PS200:Crsingle:MR0.90	0.2949
PS400:Crsingle:MR0.90	-0.1836	PS100:Cruniform:MR0.90	-0.5127	PS200:Cruniform:MR0.90	-0.1243
PS400:Cruniform:MR0.90	-0.6526	PS100:CP0.70:MR0.05	0.2870	PS200:CP0.70:MR0.05	0.2941
PS400:CP0.70:MR0.05	0.3385	PS100:CP0.90:MR0.05	-0.0652	PS200:CP0.90:MR0.05	-0.1683
PS400:CP0.90:MR0.05	0.0129	PS100:CP0.70:MR0.10	0.1491	PS200:CP0.70:MR0.10	0.0559
PS400:CP0.70:MR0.10	0.0832	PS100:CP0.90:MR0.10	0.1031	PS200:CP0.90:MR0.10	-0.3014
PS400:CP0.90:MR0.10	-0.1866	PS100:CP0.70:MR0.20	-0.0667	PS200:CP0.70:MR0.20	-0.0736
PS400:CP0.70:MR0.20	-0.2124	PS100:CP0.90:MR0.20	0.2008	PS200:CP0.90:MR0.20	-0.0419
PS400:CP0.90:MR0.20	0.0217	PS100:CP0.70:MR0.50	-0.2009	PS200:CP0.70:MR0.50	-0.2819
PS400:CP0.70:MR0.50	-0.2514	PS100:CP0.90:MR0.50	-0.2991	PS200:CP0.90:MR0.50	-0.2069
PS400:CP0.90:MR0.50	-0.2186	PS100:CP0.70:MR0.70	0.1942	PS200:CP0.70:MR0.70	-0.0597

cont.

cont.					
Covariate	Effect	Covariate	Effect	Covariate	Effect
PS400:CP0.70:MR0.70	-0.0373	PS100:CP0.90:MR0.70	-0.1299	PS200:CP0.90:MR0.70	-0.3095
PS400:CP0.90:MR0.70	-0.1242	PS100:CP0.70:MR0.90	-0.1667	PS200:CP0.70:MR0.90	-0.1296
PS400:CP0.70:MR0.90	0.0476	PS100:CP0.90:MR0.90	-0.0224	PS200:CP0.90:MR0.90	-0.0855
PS400:CP0.90:MR0.90	0.5474	Trlocal:El1:Crsingle	-0.2649	Trscale2:El1:Crsingle	-0.1724
Trscale4:El1:Crsingle	-0.1065	Trlocal:El5%:Crsingle	-0.3236	Trscale2:El5%:Crsingle	-0.0789
Trscale4:El5%:Crsingle	-0.2180	Trlocal:El1:Cruniform	-0.5018	Trscale2:El1:Cruniform	-0.1962
Trscale4:El1:Cruniform	-0.3689	Trlocal:El5%:Cruniform	-0.5550	Trscale2:El5%:Cruniform	-0.3825
Trscale4:El5%:Cruniform	-0.2486	Trlocal:El1:CP0.70	0.0522	Trscale2:El1:CP0.70	0.0119
Trscale4:El1:CP0.70	-0.0100	Trlocal:El5%:CP0.70	0.1609	Trscale2:El5%:CP0.70	0.1215
Trscale4:El5%:CP0.70	0.1555	Trlocal:El1:CP0.90	-0.0431	Trscale2:El1:CP0.90	-0.1101
Trscale4:El1:CP0.90	-0.2044	Trlocal:El5%:CP0.90	0.1615	Trscale2:El5%:CP0.90	0.0076
Trscale4:El5%:CP0.90	-0.0667	Trlocal:El1:MR0.05	-0.0086	Trscale2:El1:MR0.05	0.1346
Trscale4:El1:MR0.05	-0.1495	Trlocal:El5%:MR0.05	-0.1620	Trscale2:El5%:MR0.05	0.0891
Trscale4:El5%:MR0.05	-0.2351	Trlocal:El1:MR0.10	-0.1551	Trscale2:El1:MR0.10	-0.0769
Trscale4:El1:MR0.10	-0.2154	Trlocal:El5%:MR0.10	0.1766	Trscale2:El5%:MR0.10	0.1791
Trscale4:El5%:MR0.10	0.0670	Trlocal:El1:MR0.20	-0.3971	Trscale2:El1:MR0.20	-0.3011
Trscale4:El1:MR0.20	-0.6303	Trlocal:El5%:MR0.20	-0.5180	Trscale2:El5%:MR0.20	-0.5984
Trscale4:El5%:MR0.20	-0.6966	Trlocal:El1:MR0.50	-0.3277	Trscale2:El1:MR0.50	-0.4076
Trscale4:El1:MR0.50	-0.3712	Trlocal:El5%:MR0.50	-0.6607	Trscale2:El5%:MR0.50	-0.7748
Trscale4:El5%:MR0.50	-0.7497	Trlocal:El1:MR0.70	-0.4596	Trscale2:El1:MR0.70	-0.6811
Trscale4:El1:MR0.70	-0.6682	Trlocal:El5%:MR0.70	-0.8603	Trscale2:El5%:MR0.70	-0.8512
Trscale4:El5%:MR0.70	-0.9105	Trlocal:El1:MR0.90	-0.4336	Trscale2:El1:MR0.90	-0.6385
Trscale4:El1:MR0.90	-0.8994	Trlocal:El5%:MR0.90	-0.4841	Trscale2:El5%:MR0.90	-0.7810
Trscale4:El5%:MR0.90	-0.9093	Trlocal:Crsingle:CP0.70	-0.1683	Trscale2:Crsingle:CP0.70	-0.2114
Trscale4:Crsingle:CP0.70	-0.0579	Trlocal:Cruniform:CP0.70	0.1913	Trscale2:Cruniform:CP0.70	0.0816
Trscale4:Cruniform:CP0.70	0.0857	Trlocal:Crsingle:CP0.90	-0.2665	Trscale2:Crsingle:CP0.90	-0.1988
Trscale4:Crsingle:CP0.90	-0.1383	Trlocal:Cruniform:CP0.90	0.0175	Trscale2:Cruniform:CP0.90	-0.1619
Trscale4:Cruniform:CP0.90	-0.1452	Trlocal:Crsingle:MR0.05	-0.3103	Trscale2:Crsingle:MR0.05	-0.2001
Trscale4:Crsingle:MR0.05	-0.3887	Trlocal:Cruniform:MR0.05	-0.6278	Trscale2:Cruniform:MR0.05	-0.4601
Trscale4:Cruniform:MR0.05	-0.5563	Trlocal:Crsingle:MR0.10	-0.6691	Trscale2:Crsingle:MR0.10	-0.3436
Trscale4:Crsingle:MR0.10	-0.2394	Trlocal:Cruniform:MR0.10	-0.1465	Trscale2:Cruniform:MR0.10	-0.0907
Trscale4:Cruniform:MR0.10	-0.1533	Trlocal:Crsingle:MR0.20	-0.0692	Trscale2:Crsingle:MR0.20	0.0904
Trscale4:Crsingle:MR0.20	0.0040	Trlocal:Cruniform:MR0.20	-0.6063	Trscale2:Cruniform:MR0.20	-0.3280
Trscale4:Cruniform:MR0.20	-0.3921	Trlocal:Crsingle:MR0.50	-0.5743	Trscale2:Crsingle:MR0.50	-0.4848
Trscale4:Crsingle:MR0.50	-0.5752	Trlocal:Cruniform:MR0.50	-0.3034	Trscale2:Cruniform:MR0.50	-0.3257
Trscale4:Cruniform:MR0.50	-0.3244	Trlocal:Crsingle:MR0.70	-0.6056	Trscale2:Crsingle:MR0.70	-0.7781
Trscale4:Crsingle:MR0.70	-0.8550	Trlocal:Cruniform:MR0.70	-0.1862	Trscale2:Cruniform:MR0.70	-0.3171
Trscale4:Cruniform:MR0.70	-0.0360	Trlocal:Crsingle:MR0.90	-0.4616	Trscale2:Crsingle:MR0.90	-0.4520
Trscale4:Crsingle:MR0.90	-0.9147	Trlocal:Cruniform:MR0.90	-0.5368	Trscale2:Cruniform:MR0.90	-0.4225
Trscale4:Cruniform:MR0.90	-0.7563	Trlocal:CP0.70:MR0.05	-0.0357	Trscale2:CP0.70:MR0.05	-0.2862
Trscale4:CP0.70:MR0.05	0.0126	Trlocal:CP0.90:MR0.05	-0.3605	Trscale2:CP0.90:MR0.05	-0.2423
Trscale4:CP0.90:MR0.05	-0.2805	Trlocal:CP0.70:MR0.10	-0.2147	Trscale2:CP0.70:MR0.10	-0.2185
Trscale4:CP0.70:MR0.10	-0.2268	Trlocal:CP0.90:MR0.10	-0.2510	Trscale2:CP0.90:MR0.10	-0.0960
Trscale4:CP0.90:MR0.10	-0.1342	Trlocal:CP0.70:MR0.20	0.0513	Trscale2:CP0.70:MR0.20	-0.0786
Trscale4:CP0.70:MR0.20	-0.0723	Trlocal:CP0.90:MR0.20	0.0408	Trscale2:CP0.90:MR0.20	0.2989
Trscale4:CP0.90:MR0.20	0.2408	Trlocal:CP0.70:MR0.50	-0.1200	Trscale2:CP0.70:MR0.50	-0.1771
Trscale4:CP0.70:MR0.50	0.0890	Trlocal:CP0.90:MR0.50	-0.1083	Trscale2:CP0.90:MR0.50	0.2023
Trscale4:CP0.90:MR0.50	0.0943	Trlocal:CP0.70:MR0.70	-0.0434	Trscale2:CP0.70:MR0.70	-0.2032
Trscale4:CP0.70:MR0.70	0.0961	Trlocal:CP0.90:MR0.70	-0.2772	Trscale2:CP0.90:MR0.70	-0.1666
Trscale4:CP0.90:MR0.70	-0.3207	Trlocal:CP0.70:MR0.90	-0.2297	Trscale2:CP0.70:MR0.90	-0.1601
Trscale4:CP0.70:MR0.90	-0.1549	Trlocal:CP0.90:MR0.90	0.1401	Trscale2:CP0.90:MR0.90	0.3014
Trscale4:CP0.90:MR0.90	0.1538	El1:Crsingle:CP0.70	-0.0547	El5%:Crsingle:CP0.70	-0.0737
El1:Cruniform:CP0.70	0.0953	El5%:Cruniform:CP0.70	0.1447	El1:Crsingle:CP0.90	-0.0189
El5%:Crsingle:CP0.90	-0.1839	El1:Cruniform:CP0.90	0.0369	El5%:Cruniform:CP0.90	0.1275
El1:Crsingle:MR0.05	0.1108	El5%:Crsingle:MR0.05	0.1352	El1:Cruniform:MR0.05	-0.1705
El5%:Cruniform:MR0.05	-0.0980	El1:Crsingle:MR0.10	0.2119	El5%:Crsingle:MR0.10	-0.0314
El1:Cruniform:MR0.10	-0.0338	El5%:Cruniform:MR0.10	0.0188	El1:Crsingle:MR0.20	-0.2689
El5%:Crsingle:MR0.20	-0.0070	El1:Cruniform:MR0.20	-0.0052	El5%:Cruniform:MR0.20	0.0321
El1:Crsingle:MR0.50	-0.2978	El5%:Crsingle:MR0.50	-0.0026	El1:Cruniform:MR0.50	-0.3947
El5%:Cruniform:MR0.50	-0.2799	El1:Crsingle:MR0.70	-0.1714	El5%:Crsingle:MR0.70	0.0132
cont.					

Covariate	Effect	Covariate	Effect	Covariate	Effect
cont.					
EI1:Cruniform:MR0.70	-0.5565	EI5%:Cruniform:MR0.70	-0.2815	EI1:Crsingle:MR0.90	-0.1281
EI5%:Crsingle:MR0.90	-0.1658	EI1:Cruniform:MR0.90	-0.5794	EI5%:Cruniform:MR0.90	-0.5479
EI1:CP0.70:MR0.05	-0.1813	EI5%:CP0.70:MR0.05	-0.1406	EI1:CP0.90:MR0.05	0.0387
EI5%:CP0.90:MR0.05	-0.0742	EI1:CP0.70:MR0.10	-0.0611	EI5%:CP0.70:MR0.10	-0.0372
EI1:CP0.90:MR0.10	-0.0082	EI5%:CP0.90:MR0.10	-0.2141	EI1:CP0.70:MR0.20	-0.0757
EI5%:CP0.70:MR0.20	-0.2779	EI1:CP0.90:MR0.20	-0.0502	EI5%:CP0.90:MR0.20	-0.2838
EI1:CP0.70:MR0.50	0.2511	EI5%:CP0.70:MR0.50	0.3670	EI1:CP0.90:MR0.50	0.5270
EI5%:CP0.90:MR0.50	0.3791	EI1:CP0.70:MR0.70	-0.0918	EI5%:CP0.70:MR0.70	0.0335
EI1:CP0.90:MR0.70	0.1304	EI5%:CP0.90:MR0.70	0.1236	EI1:CP0.70:MR0.90	-0.2758
EI5%:CP0.70:MR0.90	0.0054	EI1:CP0.90:MR0.90	-0.1352	EI5%:CP0.90:MR0.90	0.1876
Crsingle:CP0.70:MR0.05	-0.4013	Cruniform:CP0.70:MR0.05	-0.1519	Crsingle:CP0.90:MR0.05	-0.3403
Cruniform:CP0.90:MR0.05	-0.1417	Crsingle:CP0.70:MR0.10	-0.3668	Cruniform:CP0.70:MR0.10	-0.2228
Crsingle:CP0.90:MR0.10	-0.1181	Cruniform:CP0.90:MR0.10	-0.1989	Crsingle:CP0.70:MR0.20	-0.1177
Cruniform:CP0.70:MR0.20	-0.1851	Crsingle:CP0.90:MR0.20	0.1825	Cruniform:CP0.90:MR0.20	-0.2783
Crsingle:CP0.70:MR0.50	-0.1358	Cruniform:CP0.70:MR0.50	-0.0720	Crsingle:CP0.90:MR0.50	-0.2277
Cruniform:CP0.90:MR0.50	-0.1527	Crsingle:CP0.70:MR0.70	-0.1476	Cruniform:CP0.70:MR0.70	-0.1562
Crsingle:CP0.90:MR0.70	-0.1542	Cruniform:CP0.90:MR0.70	-0.1776	Crsingle:CP0.70:MR0.90	-0.2764
Cruniform:CP0.70:MR0.90	-0.2750	Crsingle:CP0.90:MR0.90	-0.3850	Cruniform:CP0.90:MR0.90	-0.4017

Table 35: Coefficients for the time model for group-number GCAs on Towns2.

A.3 Group-number and Iris2

Coefficients for correctness model for the Iris2 data set can be found in Table 36. This model fits the data well (residual deviance of 925.7 on 2908 degrees of freedom, Figure 41).

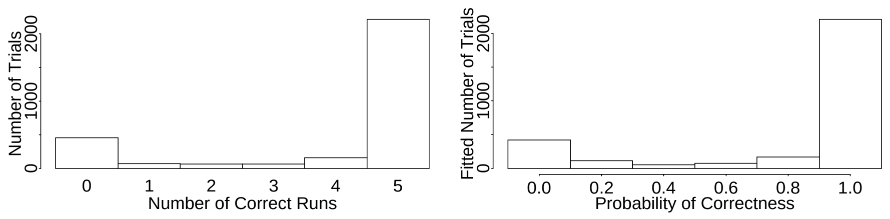


Figure 41: Comparison of experimental and fitted correctness for group-number GCAs on Iris2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	3.6483	PS100	0.0303	PS200	-3.8453
PS400	-9.8023	Trlocal	-0.9514	Trscale2	-0.9241
Trscale4	0.7082	EI1	1.7657	EI5%	1.5270
Crsingle	-1.9064	Cruniform	-3.3407	CP0.70	-2.2171
CP0.90	-2.6410	MR0.05	16.7435	MR0.10	16.5498
MR0.20	15.5493	MR0.50	15.5942	MR0.70	9.6481
MR0.90	11.0793	PS100:Crsingle	0.3943	PS200:Crsingle	3.5026
PS400:Crsingle	9.4241	PS100:Cruniform	-0.1440	PS200:Cruniform	2.0207
PS400:Cruniform	6.0565	Crsingle:MR0.05	5.1168	Cruniform:MR0.05	3.5895
Crsingle:MR0.10	10.2388	Cruniform:MR0.10	4.2681	Crsingle:MR0.20	13.0922
Cruniform:MR0.20	7.9403	Crsingle:MR0.50	15.2249	Cruniform:MR0.50	9.7813
Crsingle:MR0.70	15.0433	Cruniform:MR0.70	7.2473	Crsingle:MR0.90	12.4543
Cruniform:MR0.90	8.1281	Trlocal:Crsingle	2.5753	Trscale2:Crsingle	2.0225
Trscale4:Crsingle	0.8130	Trlocal:Cruniform	1.0837	Trscale2:Cruniform	0.7759
Trscale4:Cruniform	0.1122	PS100:MR0.05	-8.7878	PS200:MR0.05	-9.8168
PS400:MR0.05	-10.6095	PS100:MR0.10	-7.9343	PS200:MR0.10	-8.2787
cont.					

cont. Covariate	Effect	Covariate	Effect	Covariate	Effect
PS400:MR0.10	-8.5357	PS100:MR0.20	-2.5203	PS200:MR0.20	-4.7470
PS400:MR0.20	-5.4922	PS100:MR0.50	-2.8162	PS200:MR0.50	-5.3126
PS400:MR0.50	-5.5382	PS100:MR0.70	-1.8084	PS200:MR0.70	-1.8033
PS400:MR0.70	-0.1238	PS100:MR0.90	-4.4764	PS200:MR0.90	-3.9219
PS400:MR0.90	-1.9889	PS100:Trlocal	-1.3073	PS200:Trlocal	-2.7165
PS400:Trlocal	-4.0142	PS100:Trscale2	-1.0285	PS200:Trscale2	-2.2246
PS400:Trscale2	-3.7362	PS100:Trscale4	-1.0908	PS200:Trscale4	-2.3681
PS400:Trscale4	-3.7813	PS100El1	-0.0011	PS200El1	0.2187
PS400El1	-0.4128	PS100El5%	0.4390	PS200El5%	1.1998
PS400El5%	0.9041	Trlocal:MR0.05	-7.5766	Trscale2:MR0.05	-7.0973
Trscale4:MR0.05	-8.3979	Trlocal:MR0.10	-7.4056	Trscale2:MR0.10	-7.2346
Trscale4:MR0.10	-8.3486	Trlocal:MR0.20	-12.6637	Trscale2:MR0.20	-12.4918
Trscale4:MR0.20	-13.5055	Trlocal:MR0.50	-14.9067	Trscale2:MR0.50	-14.2789
Trscale4:MR0.50	-15.4905	Trlocal:MR0.70	-13.1867	Trscale2:MR0.70	-13.1047
Trscale4:MR0.70	-13.8578	Trlocal:MR0.90	-16.1629	Trscale2:MR0.90	-16.4954
Trscale4:MR0.90	-17.1303	El1:MR0.05	1.9172	El5%:MR0.05	4.1203
El1:MR0.10	1.1792	El5%:MR0.10	4.3200	El1:MR0.20	3.5082
El5%:MR0.20	6.7296	El1:MR0.50	3.7658	El5%:MR0.50	7.9447
El1:MR0.70	4.7965	El5%:MR0.70	8.2603	El1:MR0.90	7.8553
El5%:MR0.90	11.0439	Crsingle:CP0.70	1.9581	Cruniform:CP0.70	2.0465
Crsingle:CP0.90	3.9334	Cruniform:CP0.90	3.6640	PS100:CP0.70	0.5873
PS200:CP0.70	0.8287	PS400:CP0.70	0.6567	PS100:CP0.90	-0.1047

Table 36: Coefficients for the correctness model for group-number GCAs on Iris2.

The corresponding time model also provides a good fit (residual deviance of 1635.6 on 2057 degrees of freedom). The suitability of this model is also supported by the comparative histograms (Figure 42) and the residual plot (Figure 43). Model coefficients are listed in Table 37.

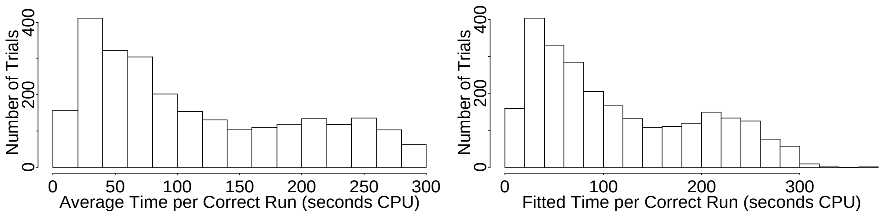


Figure 42: Comparison of experimental and fitted time for group-number GCAs on Iris2.

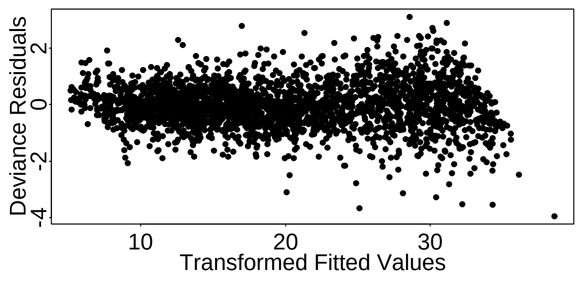


Figure 43: Residuals plot for time model for group-number GCAs on Iris2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	5.2076	PS100	0.1574	PS200	0.4526
PS400	0.5529	Trlocal	0.1546	Trscale2	0.2322
Trscale4	0.1606	El1	0.0395	El5%	-0.0927
Crsingle	-0.0784	Cruniform	0.1244	CP0.70	0.0472
CP0.90	0.0765	MR0.05	-1.2663	MR0.10	-1.4853
MR0.20	-1.6929	MR0.50	-1.7090	MR0.70	-1.7038
MR0.90	-1.6420	PS100:Trlocal	-0.0224	PS200:Trlocal	-0.1268
PS400:Trlocal	-0.0645	PS100:Trscale2	-0.0675	PS200:Trscale2	-0.1731
PS400:Trscale2	-0.1537	PS100:Trscale4	-0.0815	PS200:Trscale4	-0.1246
PS400:Trscale4	-0.4615	PS100:El1	-0.0292	PS200:El1	-0.2091
PS400:El1	-0.2827	PS100:El5%	0.0353	PS200:El5%	-0.0691
PS400:El5%	-0.1440	PS100:Crsingle	-0.0591	PS200:Crsingle	-0.1955
PS400:Crsingle	-0.1744	PS100:Cruniform	-0.0966	PS200:Cruniform	-0.2925
PS400:Cruniform	-0.3538	PS100:CP0.70	0.0008	PS200:CP0.70	-0.0051
PS400:CP0.70	0.0422	PS100:CP0.90	0.0176	PS200:CP0.90	-0.0091
PS400:CP0.90	0.0079	PS100:MR0.05	0.3763	PS200:MR0.05	0.5901
PS400:MR0.05	1.0665	PS100:MR0.10	0.3276	PS200:MR0.10	0.6658
PS400:MR0.10	1.1672	PS100:MR0.20	0.3548	PS200:MR0.20	0.7175
PS400:MR0.20	1.2180	PS100:MR0.50	0.3285	PS200:MR0.50	0.6613
PS400:MR0.50	1.1837	PS100:MR0.70	0.2041	PS200:MR0.70	0.5807
PS400:MR0.70	1.1156	PS100:MR0.90	0.3904	PS200:MR0.90	0.6313
PS400:MR0.90	1.1073	Trlocal:El1	-0.1402	Trscale2:El1	-0.2324
Trscale4:El1	-0.1728	Trlocal:El5%	0.0463	Trscale2:El5%	-0.0725
Trscale4:El5%	-0.0349	Trlocal:Crsingle	-0.0206	Trscale2:Crsingle	0.0047
Trscale4:Crsingle	-0.0306	Trlocal:Cruniform	-0.0146	Trscale2:Cruniform	-0.0280
Trscale4:Cruniform	-0.0188	Trlocal:CP0.70	-0.0347	Trscale2:CP0.70	-0.0079
Trscale4:CP0.70	-0.0259	Trlocal:CP0.90	-0.0748	Trscale2:CP0.90	-0.0304
Trscale4:CP0.90	-0.0497	Trlocal:MR0.05	0.6607	Trscale2:MR0.05	0.5679
Trscale4:MR0.05	0.5389	Trlocal:MR0.10	1.0267	Trscale2:MR0.10	0.8042
Trscale4:MR0.10	0.9062	Trlocal:MR0.20	1.2434	Trscale2:MR0.20	1.1186
Trscale4:MR0.20	1.2328	Trlocal:MR0.50	1.5981	Trscale2:MR0.50	1.4927
Trscale4:MR0.50	1.5385	Trlocal:MR0.70	1.8863	Trscale2:MR0.70	1.7570
Trscale4:MR0.70	1.8583	Trlocal:MR0.90	2.0437	Trscale2:MR0.90	2.0585
Trscale4:MR0.90	2.1694	El1:Crsingle	0.0824	El5%:Crsingle	0.1441
El1:Cruniform	0.1453	El5%:Cruniform	0.1854	El1:CP0.70	-0.0429
El5%:CP0.70	0.0110	El1:CP0.90	-0.0153	El5%:CP0.90	-0.0015
El1:MR0.05	0.0421	El5%:MR0.05	0.0382	El1:MR0.10	-0.0622
El5%:MR0.10	-0.0363	El1:MR0.20	-0.0775	El5%:MR0.20	-0.0597
El1:MR0.50	-0.2031	El5%:MR0.50	-0.2258	El1:MR0.70	-0.2036
El5%:MR0.70	-0.2251	El1:MR0.90	-0.3295	El5%:MR0.90	-0.3388
Crsingle:CP0.70	-0.0628	Cruniform:CP0.70	-0.0118	Crsingle:CP0.90	-0.0903
Cruniform:CP0.90	-0.0355	Crsingle:MR0.05	-0.1107	Cruniform:MR0.05	0.0224
Crsingle:MR0.10	-0.4067	Cruniform:MR0.10	-0.1920	Crsingle:MR0.20	-0.6807
Cruniform:MR0.20	-0.4149	Crsingle:MR0.50	-1.0613	Cruniform:MR0.50	-0.6421
Crsingle:MR0.70	-1.1962	Cruniform:MR0.70	-0.5692	Crsingle:MR0.90	-1.1734
Cruniform:MR0.90	-0.6549	CP0.70:MR0.05	0.0782	CP0.90:MR0.05	0.1519
CP0.70:MR0.10	0.0691	CP0.90:MR0.10	0.2310	CP0.70:MR0.20	0.1716
CP0.90:MR0.20	0.2923	CP0.70:MR0.50	0.1056	CP0.90:MR0.50	0.2560
CP0.70:MR0.70	0.1141	CP0.90:MR0.70	0.2523	CP0.70:MR0.90	0.1155
CP0.90:MR0.90	0.1545	PS100:Trlocal:El1	0.1534	PS200:Trlocal:El1	0.2273
PS400:Trlocal:El1	0.3952	PS100:Trscale2:El1	0.1169	PS200:Trscale2:El1	0.1934
PS400:Trscale2:El1	0.3633	PS100:Trscale4:El1	0.1708	PS200:Trscale4:El1	0.2134
PS400:Trscale4:El1	0.3561	PS100:Trlocal:El5%	0.0044	PS200:Trlocal:El5%	0.0583
PS400:Trlocal:El5%	0.2398	PS100:Trscale2:El5%	-0.0131	PS200:Trscale2:El5%	0.0414
PS400:Trscale2:El5%	0.1792	PS100:Trscale4:El5%	-0.0157	PS200:Trscale4:El5%	0.0438
PS400:Trscale4:El5%	0.1364	PS100:Trlocal:Crsingle	0.0167	PS200:Trlocal:Crsingle	0.1238
PS400:Trlocal:Crsingle	0.0247	PS100:Trscale2:Crsingle	0.0382	PS200:Trscale2:Crsingle	0.0945
PS400:Trscale2:Crsingle	-0.0001	PS100:Trscale4:Crsingle	0.0323	PS200:Trscale4:Crsingle	0.0535
PS400:Trscale4:Crsingle	0.3706	PS100:Trlocal:Cruniform	0.0637	PS200:Trlocal:Cruniform	0.1841
PS400:Trlocal:Cruniform	0.0000	PS100:Trscale2:Cruniform	0.0629	PS200:Trscale2:Cruniform	0.2088
cont.					

cont.

Covariate	Effect	Covariate	Effect	Covariate	Effect
PS400:Trscale2:Cruniform	0.0000	PS100:Trscale4:Cruniform	0.0808	PS200:Trscale4:Cruniform	0.1639
PS400:Trscale4:Cruniform	0.4030	PS100:Trlocal:MR0.05	0.1563	PS200:Trlocal:MR0.05	0.2068
PS400:Trlocal:MR0.05	0.1146	PS100:Trscale2:MR0.05	0.2212	PS200:Trscale2:MR0.05	0.2722
PS400:Trscale2:MR0.05	0.3034	PS100:Trscale4:MR0.05	0.2933	PS200:Trscale4:MR0.05	0.3370
PS400:Trscale4:MR0.05	0.3464	PS100:Trlocal:MR0.10	0.0587	PS200:Trlocal:MR0.10	0.0490
PS400:Trlocal:MR0.10	-0.0171	PS100:Trscale2:MR0.10	0.1946	PS200:Trscale2:MR0.10	0.1628
PS400:Trscale2:MR0.10	0.1629	PS100:Trscale4:MR0.10	0.1719	PS200:Trscale4:MR0.10	0.0979
PS400:Trscale4:MR0.10	0.0468	PS100:Trlocal:MR0.20	0.0091	PS200:Trlocal:MR0.20	-0.0268
PS400:Trlocal:MR0.20	-0.0537	PS100:Trscale2:MR0.20	0.0595	PS200:Trscale2:MR0.20	0.0304
PS400:Trscale2:MR0.20	0.0820	PS100:Trscale4:MR0.20	0.0666	PS200:Trscale4:MR0.20	0.0194
PS400:Trscale4:MR0.20	0.0148	PS100:Trlocal:MR0.50	-0.1330	PS200:Trlocal:MR0.50	-0.2032
PS400:Trlocal:MR0.50	-0.3277	PS100:Trscale2:MR0.50	-0.1105	PS200:Trscale2:MR0.50	-0.1509
PS400:Trscale2:MR0.50	-0.1330	PS100:Trscale4:MR0.50	-0.0673	PS200:Trscale4:MR0.50	-0.1363
PS400:Trscale4:MR0.50	-0.1906	PS100:Trlocal:MR0.70	-0.1420	PS200:Trlocal:MR0.70	-0.2776
PS400:Trlocal:MR0.70	-0.3148	PS100:Trscale2:MR0.70	-0.0488	PS200:Trscale2:MR0.70	-0.2272
PS400:Trscale2:MR0.70	-0.1876	PS100:Trscale4:MR0.70	-0.1185	PS200:Trscale4:MR0.70	-0.2936
PS400:Trscale4:MR0.70	-0.2930	PS100:Trlocal:MR0.90	-0.0523	PS200:Trlocal:MR0.90	-0.3351
PS400:Trlocal:MR0.90	-0.3855	PS100:Trscale2:MR0.90	-0.1285	PS200:Trscale2:MR0.90	-0.3288
PS400:Trscale2:MR0.90	-0.3026	PS100:Trscale4:MR0.90	-0.1616	PS200:Trscale4:MR0.90	-0.3795
PS400:Trscale4:MR0.90	-0.4527	PS100:El1:Crsingle	-0.1029	PS200:El1:Crsingle	0.0011
PS400:El1:Crsingle	0.0165	PS100:El5%:Crsingle	-0.0753	PS200:El5%:Crsingle	-0.0708
PS400:El5%:Crsingle	-0.0885	PS100:El1:Cruniform	-0.0974	PS200:El1:Cruniform	0.0750
PS400:El1:Cruniform	0.2119	PS100:El5%:Cruniform	-0.1027	PS200:El5%:Cruniform	-0.0458
PS400:El5%:Cruniform	0.1164	PS100:El1:MR0.05	-0.0527	PS200:El1:MR0.05	0.0782
PS400:El1:MR0.05	0.0778	PS100:El5%:MR0.05	-0.0723	PS200:El5%:MR0.05	0.0180
PS400:El5%:MR0.05	0.0123	PS100:El1:MR0.10	0.0611	PS200:El1:MR0.10	0.1508
PS400:El1:MR0.10	0.2055	PS100:El5%:MR0.10	0.0111	PS200:El5%:MR0.10	0.0978
PS400:El5%:MR0.10	0.1658	PS100:El1:MR0.20	0.0963	PS200:El1:MR0.20	0.1642
PS400:El1:MR0.20	0.2823	PS100:El5%:MR0.20	0.1131	PS200:El5%:MR0.20	0.1393
PS400:El5%:MR0.20	0.2286	PS100:El1:MR0.50	0.2357	PS200:El1:MR0.50	0.3528
PS400:El1:MR0.50	0.4360	PS100:El5%:MR0.50	0.2253	PS200:El5%:MR0.50	0.2954
PS400:El5%:MR0.50	0.3724	PS100:El1:MR0.70	0.2759	PS200:El1:MR0.70	0.4810
PS400:El1:MR0.70	0.5153	PS100:El5%:MR0.70	0.2900	PS200:El5%:MR0.70	0.4642
PS400:El5%:MR0.70	0.4577	PS100:El1:MR0.90	0.1835	PS200:El1:MR0.90	0.5762
PS400:El1:MR0.90	0.6153	PS100:El5%:MR0.90	0.2297	PS200:El5%:MR0.90	0.4972
PS400:El5%:MR0.90	0.5123	PS100:Crsingle:MR0.05	-0.2360	PS200:Crsingle:MR0.05	-0.3412
PS400:Crsingle:MR0.05	-0.5218	PS100:Cruniform:MR0.05	-0.1074	PS200:Cruniform:MR0.05	-0.1420
PS400:Cruniform:MR0.05	-0.2453	PS100:Crsingle:MR0.10	-0.1781	PS200:Crsingle:MR0.10	-0.2837
PS400:Crsingle:MR0.10	-0.4303	PS100:Cruniform:MR0.10	-0.0355	PS200:Cruniform:MR0.10	-0.0631
PS400:Cruniform:MR0.10	-0.1380	PS100:Crsingle:MR0.20	-0.1136	PS200:Crsingle:MR0.20	-0.1649
PS400:Crsingle:MR0.20	-0.3676	PS100:Cruniform:MR0.20	0.0290	PS200:Cruniform:MR0.20	0.0554
PS400:Cruniform:MR0.20	-0.0793	PS100:Crsingle:MR0.50	-0.0421	PS200:Crsingle:MR0.50	-0.0626
PS400:Crsingle:MR0.50	-0.1285	PS100:Cruniform:MR0.50	0.0725	PS200:Cruniform:MR0.50	0.1151
PS400:Cruniform:MR0.50	0.0238	PS100:Crsingle:MR0.70	-0.0290	PS200:Crsingle:MR0.70	0.0131
PS400:Crsingle:MR0.70	-0.1068	PS100:Cruniform:MR0.70	0.1015	PS200:Cruniform:MR0.70	0.0597
PS400:Cruniform:MR0.70	0.0167	PS100:Crsingle:MR0.90	-0.1447	PS200:Crsingle:MR0.90	-0.0813
PS400:Crsingle:MR0.90	-0.1561	PS100:Cruniform:MR0.90	-0.0478	PS200:Cruniform:MR0.90	0.0178
PS400:Cruniform:MR0.90	0.0000	PS100:CP0.70:MR0.05	-0.0016	PS200:CP0.70:MR0.05	-0.0141
PS400:CP0.70:MR0.05	-0.0548	PS100:CP0.90:MR0.05	-0.0378	PS200:CP0.90:MR0.05	-0.0276
PS400:CP0.90:MR0.05	-0.0737	PS100:CP0.70:MR0.10	0.0443	PS200:CP0.70:MR0.10	0.0095
PS400:CP0.70:MR0.10	-0.0717	PS100:CP0.90:MR0.10	-0.0698	PS200:CP0.90:MR0.10	-0.0461
PS400:CP0.90:MR0.10	-0.1321	PS100:CP0.70:MR0.20	-0.0668	PS200:CP0.70:MR0.20	-0.0194
PS400:CP0.70:MR0.20	-0.0672	PS100:CP0.90:MR0.20	-0.0588	PS200:CP0.90:MR0.20	-0.0267
PS400:CP0.90:MR0.20	-0.0469	PS100:CP0.70:MR0.50	0.0166	PS200:CP0.70:MR0.50	-0.0008
PS400:CP0.70:MR0.50	-0.0640	PS100:CP0.90:MR0.50	-0.0694	PS200:CP0.90:MR0.50	-0.0400
PS400:CP0.90:MR0.50	-0.0371	PS100:CP0.70:MR0.70	0.0990	PS200:CP0.70:MR0.70	0.0310
PS400:CP0.70:MR0.70	-0.0227	PS100:CP0.90:MR0.70	0.1044	PS200:CP0.90:MR0.70	0.0139
PS400:CP0.90:MR0.70	0.0388	PS100:CP0.70:MR0.90	-0.0206	PS200:CP0.70:MR0.90	-0.0647
PS400:CP0.70:MR0.90	-0.0770	PS100:CP0.90:MR0.90	-0.0518	PS200:CP0.90:MR0.90	-0.0303
PS400:CP0.90:MR0.90	0.0079	Trlocal:El1:Crsingle	-0.1540	Trscale2:El1:Crsingle	-0.0836

cont.

cont. Covariate	Effect	Covariate	Effect	Covariate	Effect
Trscale4:El1:Crsingle	-0.0801	Trlocal:El5%:Crsingle	-0.2333	Trscale2:El5%:Crsingle	-0.1511
Trscale4:El5%:Crsingle	-0.1262	Trlocal:El1:Cruniform	-0.1551	Trscale2:El1:Cruniform	-0.0721
Trscale4:El1:Cruniform	-0.1023	Trlocal:El5%:Cruniform	-0.1779	Trscale2:El5%:Cruniform	-0.1122
Trscale4:El5%:Cruniform	-0.1058	Trlocal:El1:MR0.05	-0.2334	Trscale2:El1:MR0.05	-0.2032
Trscale4:El1:MR0.05	-0.2802	Trlocal:El5%:MR0.05	-0.4556	Trscale2:El5%:MR0.05	-0.3656
Trscale4:El5%:MR0.05	-0.3617	Trlocal:El1:MR0.10	-0.3352	Trscale2:El1:MR0.10	-0.2028
Trscale4:El1:MR0.10	-0.3162	Trlocal:El5%:MR0.10	-0.5448	Trscale2:El5%:MR0.10	-0.4105
Trscale4:El5%:MR0.10	-0.4455	Trlocal:El1:MR0.20	-0.3420	Trscale2:El1:MR0.20	-0.2695
Trscale4:El1:MR0.20	-0.3686	Trlocal:El5%:MR0.20	-0.5926	Trscale2:El5%:MR0.20	-0.4923
Trscale4:El5%:MR0.20	-0.5659	Trlocal:El1:MR0.50	-0.4032	Trscale2:El1:MR0.50	-0.3383
Trscale4:El1:MR0.50	-0.3906	Trlocal:El5%:MR0.50	-0.6868	Trscale2:El5%:MR0.50	-0.5758
Trscale4:El5%:MR0.50	-0.6243	Trlocal:El1:MR0.70	-0.5864	Trscale2:El1:MR0.70	-0.4930
Trscale4:El1:MR0.70	-0.5725	Trlocal:El5%:MR0.70	-0.9127	Trscale2:El5%:MR0.70	-0.8143
Trscale4:El5%:MR0.70	-0.8504	Trlocal:El1:MR0.90	-0.6271	Trscale2:El1:MR0.90	-0.5880
Trscale4:El1:MR0.90	-0.6417	Trlocal:El5%:MR0.90	-1.0298	Trscale2:El5%:MR0.90	-0.9412
Trscale4:El5%:MR0.90	-0.9838	Trlocal:Crsingle:MR0.05	-0.2053	Trscale2:Crsingle:MR0.05	-0.3123
Trscale4:Crsingle:MR0.05	-0.2761	Trlocal:Cruniform:MR0.05	-0.0778	Trscale2:Cruniform:MR0.05	-0.1864
Trscale4:Cruniform:MR0.05	-0.1855	Trlocal:Crsingle:MR0.10	-0.2590	Trscale2:Crsingle:MR0.10	-0.3202
Trscale4:Crsingle:MR0.10	-0.2447	Trlocal:Cruniform:MR0.10	-0.1393	Trscale2:Cruniform:MR0.10	-0.1639
Trscale4:Cruniform:MR0.10	-0.1634	Trlocal:Crsingle:MR0.20	-0.3715	Trscale2:Crsingle:MR0.20	-0.4120
Trscale4:Crsingle:MR0.20	-0.3958	Trlocal:Cruniform:MR0.20	-0.1268	Trscale2:Cruniform:MR0.20	-0.1820
Trscale4:Cruniform:MR0.20	-0.1992	Trlocal:Crsingle:MR0.50	-0.2681	Trscale2:Crsingle:MR0.50	-0.4002
Trscale4:Crsingle:MR0.50	-0.3521	Trlocal:Cruniform:MR0.50	0.0246	Trscale2:Cruniform:MR0.50	-0.0810
Trscale4:Cruniform:MR0.50	-0.0617	Trlocal:Crsingle:MR0.70	-0.2603	Trscale2:Crsingle:MR0.70	-0.3328
Trscale4:Crsingle:MR0.70	-0.3101	Trlocal:Cruniform:MR0.70	0.1024	Trscale2:Cruniform:MR0.70	0.0523
Trscale4:Cruniform:MR0.70	0.0663	Trlocal:Crsingle:MR0.90	-0.1631	Trscale2:Crsingle:MR0.90	-0.3274
Trscale4:Crsingle:MR0.90	-0.3398	Trlocal:Cruniform:MR0.90	0.1604	Trscale2:Cruniform:MR0.90	0.0203
Trscale4:Cruniform:MR0.90	0.0290	El1:Crsingle:MR0.05	-0.0831	El5%:Crsingle:MR0.05	-0.0534
El1:Cruniform:MR0.05	-0.1887	El5%:Cruniform:MR0.05	-0.1364	El1:Crsingle:MR0.10	-0.1143
El5%:Crsingle:MR0.10	-0.0802	El1:Cruniform:MR0.10	-0.1984	El5%:Cruniform:MR0.10	-0.1862
El1:Crsingle:MR0.20	0.0050	El5%:Crsingle:MR0.20	0.0556	El1:Cruniform:MR0.20	-0.2223
El5%:Cruniform:MR0.20	-0.1981	El1:Crsingle:MR0.50	-0.0384	El5%:Crsingle:MR0.50	0.0685
El1:Cruniform:MR0.50	-0.4491	El5%:Cruniform:MR0.50	-0.3743	El1:Crsingle:MR0.70	0.0088
El5%:Crsingle:MR0.70	0.0652	El1:Cruniform:MR0.70	-0.6753	El5%:Cruniform:MR0.70	-0.6388
El1:Crsingle:MR0.90	-0.0280	El5%:Crsingle:MR0.90	0.0242	El1:Cruniform:MR0.90	-0.6452
El5%:Cruniform:MR0.90	-0.6228	El1:CP0.70:MR0.05	0.0165	El5%:CP0.70:MR0.05	0.0143
El1:CP0.90:MR0.05	0.0292	El5%:CP0.90:MR0.05	0.0473	El1:CP0.70:MR0.10	0.0802
El5%:CP0.70:MR0.10	0.0289	El1:CP0.90:MR0.10	0.0749	El5%:CP0.90:MR0.10	0.0452
El1:CP0.70:MR0.20	0.0481	El5%:CP0.70:MR0.20	-0.0095	El1:CP0.90:MR0.20	0.0287
El5%:CP0.90:MR0.20	-0.0009	El1:CP0.70:MR0.50	0.0829	El5%:CP0.70:MR0.50	0.0641
El1:CP0.90:MR0.50	0.1313	El5%:CP0.90:MR0.50	0.0874	El1:CP0.70:MR0.70	0.0774
El5%:CP0.70:MR0.70	0.0323	El1:CP0.90:MR0.70	0.0408	El5%:CP0.90:MR0.70	0.0381
El1:CP0.70:MR0.90	0.0863	El5%:CP0.70:MR0.90	0.0609	El1:CP0.90:MR0.90	0.1339
El5%:CP0.90:MR0.90	0.1608	Crsingle:CP0.70:MR0.05	-0.1256	Cruniform:CP0.70:MR0.05	-0.1131
Crsingle:CP0.90:MR0.05	-0.2432	Cruniform:CP0.90:MR0.05	-0.1800	Crsingle:CP0.70:MR0.10	-0.1476
Cruniform:CP0.70:MR0.10	-0.1198	Crsingle:CP0.90:MR0.10	-0.2511	Cruniform:CP0.90:MR0.10	-0.2606
Crsingle:CP0.70:MR0.20	-0.2024	Cruniform:CP0.70:MR0.20	-0.1900	Crsingle:CP0.90:MR0.20	-0.3663
Cruniform:CP0.90:MR0.20	-0.3097	Crsingle:CP0.70:MR0.50	-0.1998	Cruniform:CP0.70:MR0.50	-0.1905
Crsingle:CP0.90:MR0.50	-0.4305	Cruniform:CP0.90:MR0.50	-0.3369	Crsingle:CP0.70:MR0.70	-0.2567
Cruniform:CP0.70:MR0.70	-0.2276	Crsingle:CP0.90:MR0.70	-0.4971	Cruniform:CP0.90:MR0.70	-0.3439
Crsingle:CP0.70:MR0.90	-0.2350	Cruniform:CP0.70:MR0.90	-0.1893	Crsingle:CP0.90:MR0.90	-0.4278

Table 37: Coefficients for the time model for group-number GCAs on Iris2. NA = coefficient not defined because of singularity.

A.4 Order-based and Ruspini2

Table 38 provides the coefficients for the order-based correctness model for Ruspini2. The residual deviance of 926.8 on 1239 degrees of freedom indicates a good fit, and this is supported by the histograms in Figure 44).

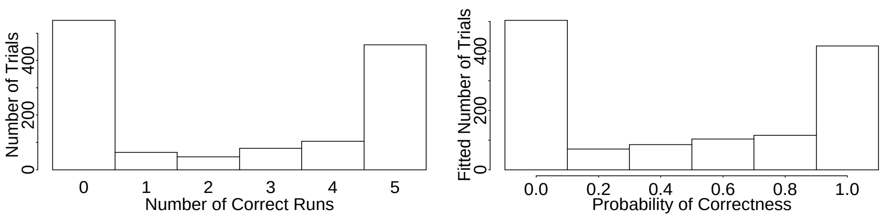


Figure 44: Comparison of experimental and fitted correctness for order-based GCAs on Ruspini2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	-2.4715	PS100	-5.6902	PS200	-3.6438
Trscale2	-0.1199	Trscale4	0.1205	El5%	0.5463
Credge	0.1860	CrPMX	-11.0055	CP0.70	1.6155
CP0.90	2.3811	MR0.20	2.0589	MR0.70	3.4281
MR0.90	2.8056	Credge:CP0.70	-0.4342	CrPMX:CP0.70	0.6507
Credge:CP0.90	-0.4922	CrPMX:CP0.90	-1.9798	Credge:MR0.20	0.8726
CrPMX:MR0.20	-1.9710	Credge:MR0.70	3.8386	CrPMX:MR0.70	3.9004
Credge:MR0.90	4.6169	CrPMX:MR0.90	5.7564	CP0.70:MR0.20	-0.4930
CP0.90:MR0.20	0.3465	CP0.70:MR0.70	0.0653	CP0.90:MR0.70	0.7733
CP0.70:MR0.90	-0.3047	CP0.90:MR0.90	0.1536	PS100:MR0.20	-1.3884
PS200:MR0.20	-2.1380	PS100:MR0.70	-2.6810	PS200:MR0.70	-4.6449
PS100:MR0.90	-2.3393	PS200:MR0.90	-4.2227	Trscale2:Credge	3.4483
Trscale4:Credge	2.8195	Trscale2:CrPMX	-8.5958	Trscale4:CrPMX	-8.8362
El5%:Credge	-0.6386	El5%:CrPMX	1.7717	PS100:Crborrow:CP0.50	6.7534
PS200:Crborrow:CP0.50	2.4441	PS100:Credge:CP0.50	7.5695	PS200:Credge:CP0.50	5.7067
PS100:CrPMX:CP0.50	0.6626	PS200:CrPMX:CP0.50	0.6532	PS100:Crborrow:CP0.70	6.6132
PS200:Crborrow:CP0.70	3.0023	PS100:Credge:CP0.70	7.7948	PS200:Credge:CP0.70	6.2593
PS100:CrPMX:CP0.70	-1.3722	PS200:CrPMX:CP0.70	-1.3368	PS100:Crborrow:CP0.90	6.3623
PS200:Crborrow:CP0.90	3.5917	PS100:Credge:CP0.90	7.0082	PS200:Credge:CP0.90	6.3237

Table 38: Coefficients for the correctness model for order-based GCAs on Ruspini2. NA = coefficient not defined because of singularity.

The coefficients for the corresponding time model are listed in Table 39. This model fits the data poorly with a residual deviance of 1619.7 on 566 degrees of freedom (see Figure 45). The residual plot (Figure 46) does not indicate any violation of model assumptions.

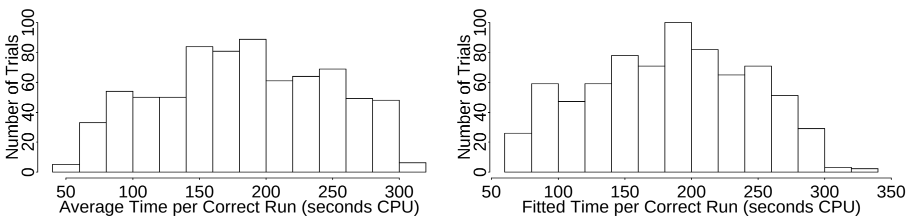


Figure 45: Comparison of experimental and fitted time for order-based GCAs on Ruspini2

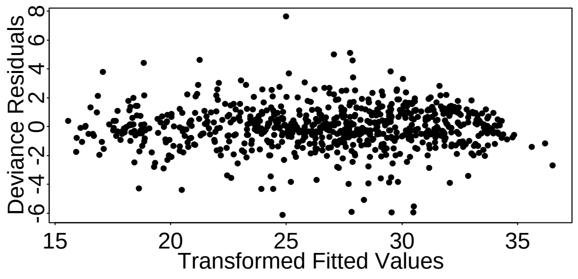


Figure 46: Residual plot for time model for order-based GCAs on Ruspini2

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	5.3846	PS100	0.2882	PS200	0.4571
Trscale2	0.1877	Trscale4	0.0924	El5%	-0.1339
Credge	0.0826	CP0.70	0.1013	CP0.90	-0.0281
Muuniform	0.0168	MR0.20	0.2572	MR0.70	0.0624
MR0.90	-0.0536	PS100:Trscale2	-0.3220	PS200:Trscale2	-0.2186
PS100:Trscale4	-0.3344	PS200:Trscale4	-0.2708	PS100:El5%	0.0942
PS200:El5%	0.0045	PS100:Credge	-0.3362	PS200:Credge	-0.4728
PS100:CP0.70	-0.1876	PS200:CP0.70	-0.2354	PS100:CP0.90	-0.1895
PS200:CP0.90	-0.2032	PS100:Muuniform	0.0075	PS200:Muuniform	-0.0410
PS100:MR0.20	-0.3119	PS200:MR0.20	-0.2307	PS100:MR0.70	-0.1807
PS200:MR0.70	-0.0497	PS100:MR0.90	-0.1035	PS200:MR0.90	0.1065
Trscale2:El5%	0.1018	Trscale4:El5%	0.1668	Trscale2:Credge	-0.3942
Trscale4:Credge	-0.2931	Trscale2:CP0.70	-0.2292	Trscale4:CP0.70	-0.1947
Trscale2:CP0.90	-0.2880	Trscale4:CP0.90	-0.1943	Trscale2:Muuniform	-0.0330
Trscale4:Muuniform	0.0427	Trscale2:MR0.20	-0.1623	Trscale4:MR0.20	-0.1482
Trscale2:MR0.70	0.0041	Trscale4:MR0.70	0.0248	Trscale2:MR0.90	0.0639
Trscale4:MR0.90	0.0617	El5%:Credge	0.0520	El5%:CP0.70	0.0515
El5%:CP0.90	0.1585	El5%:Muuniform	0.0528	El5%:MR0.20	0.0285
El5%:MR0.70	0.0346	El5%:MR0.90	0.0797	Credge:CP0.70	-0.0555
Credge:CP0.90	-0.1625	Credge:Muuniform	-0.0572	Credge:MR0.20	-0.1397
Credge:MR0.70	-0.4932	Credge:MR0.90	-0.4841	CP0.70:Muuniform	0.0956
CP0.90:Muuniform	0.0734	CP0.70:MR0.20	-0.2513	CP0.90:MR0.20	-0.2465
CP0.70:MR0.70	-0.2840	CP0.90:MR0.70	-0.2626	CP0.70:MR0.90	-0.1746
CP0.90:MR0.90	-0.1901	Muuniform:MR0.20	-0.1154	Muuniform:MR0.70	-0.1259
Muuniform:MR0.90	-0.0956	PS100:Trscale2:El5%	-0.0511	PS200:Trscale2:El5%	-0.0807
PS100:Trscale4:El5%	-0.0471	PS200:Trscale4:El5%	-0.0626	PS100:Trscale2:Credge	0.0520
PS200:Trscale2:Credge	0.3437	PS100:Trscale4:Credge	0.0170	PS200:Trscale4:Credge	0.2910
PS100:Trscale2:CP0.70	0.1132	PS200:Trscale2:CP0.70	0.2367	PS100:Trscale4:CP0.70	0.1327
PS200:Trscale4:CP0.70	0.3292	PS100:Trscale2:CP0.90	0.1242	PS200:Trscale2:CP0.90	0.3311
PS100:Trscale4:CP0.90	0.1917	PS200:Trscale4:CP0.90	0.3831	PS100:Trscale2:MR0.20	0.1882
PS200:Trscale2:MR0.20	0.0687	PS100:Trscale4:MR0.20	0.2037	PS200:Trscale4:MR0.20	0.0869
PS100:Trscale2:MR0.70	0.3373	PS200:Trscale2:MR0.70	0.1034	PS100:Trscale4:MR0.70	0.2725
PS200:Trscale4:MR0.70	0.0771	PS100:Trscale2:MR0.90	0.2269	PS200:Trscale2:MR0.90	0.0966
PS100:Trscale4:MR0.90	0.2699	PS200:Trscale4:MR0.90	0.0913	PS100:El5%:Muuniform	-0.0492
PS200:El5%:Muuniform	0.0120	PS100:Credge:MR0.20	0.1531	PS200:Credge:MR0.20	0.1941
PS100:Credge:MR0.70	0.2827	PS200:Credge:MR0.70	0.4298	PS100:Credge:MR0.90	0.2949
PS200:Credge:MR0.90	0.3912	PS100:CP0.70:Muuniform	-0.0778	PS200:CP0.70:Muuniform	-0.0547
PS100:CP0.90:Muuniform	-0.0178	PS200:CP0.90:Muuniform	0.0201	PS100:CP0.70:MR0.20	0.2132
PS200:CP0.70:MR0.20	0.2018	PS100:CP0.90:MR0.20	0.1823	PS200:CP0.90:MR0.20	0.1884
PS100:CP0.70:MR0.70	0.2592	PS200:CP0.70:MR0.70	0.2385	PS100:CP0.90:MR0.70	0.1729
PS200:CP0.90:MR0.70	0.1799	PS100:CP0.70:MR0.90	0.2064	PS200:CP0.70:MR0.90	0.1266
PS100:CP0.90:MR0.90	0.1059	PS200:CP0.90:MR0.90	0.0810	PS100:Muuniform:MR0.20	0.1312
PS200:Muuniform:MR0.20	0.0813	PS100:Muuniform:MR0.70	0.0738	PS200:Muuniform:MR0.70	0.0868
PS100:Muuniform:MR0.90	0.0451	PS200:Muuniform:MR0.90	0.0405	Trscale2:El5%:CP0.70	-0.0796
Trscale4:El5%:CP0.70	-0.0180	Trscale2:El5%:CP0.90	-0.0651	Trscale4:El5%:CP0.90	-0.1316
Trscale2:El5%:Muuniform	0.0574	Trscale4:El5%:Muuniform	-0.0060	Trscale2:El5%:MR0.20	-0.0415
cont.					

cont.					
Covariate	Effect	Covariate	Effect	Covariate	Effect
Trscale4:El5%:MR0.20	-0.0848	Trscale2:El5%:MR0.70	-0.1590	Trscale4:El5%:MR0.70	-0.1707
Trscale2:El5%:MR0.90	-0.1368	Trscale4:El5%:MR0.90	-0.1728	Trscale2:Credge:CP0.70	-0.0744
Trscale4:Credge:CP0.70	-0.2073	Trscale2:Credge:CP0.90	-0.0770	Trscale4:Credge:CP0.90	-0.1549
Trscale2:CP0.70:Muuniform	0.0261	Trscale4:CP0.70:Muuniform	-0.0445	Trscale2:CP0.90:Muuniform	-0.0517
Trscale4:CP0.90:Muuniform	-0.0841	Trscale2:CP0.70:MR0.20	0.1006	Trscale4:CP0.70:MR0.20	0.1194
Trscale2:CP0.90:MR0.20	0.1208	Trscale4:CP0.90:MR0.20	0.1123	Trscale2:CP0.70:MR0.70	0.1324
Trscale4:CP0.70:MR0.70	0.1347	Trscale2:CP0.90:MR0.70	0.2104	Trscale4:CP0.90:MR0.70	0.1566
Trscale2:CP0.70:MR0.90	0.1377	Trscale4:CP0.70:MR0.90	0.1783	Trscale2:CP0.90:MR0.90	0.2534
Trscale4:CP0.90:MR0.90	0.2239	El5%:Credge:MR0.20	-0.0304	El5%:Credge:MR0.70	0.0265
El5%:Credge:MR0.90	0.0622	El5%:CP0.70:Muuniform	-0.0856	El5%:CP0.90:Muuniform	-0.0816
El5%:CP0.70:MR0.20	0.0162	El5%:CP0.90:MR0.20	-0.0118	El5%:CP0.70:MR0.70	0.0216
El5%:CP0.90:MR0.70	-0.0422	El5%:CP0.70:MR0.90	-0.0697	El5%:CP0.90:MR0.90	-0.1225
Credge:Muuniform:MR0.20	0.0258	Credge:Muuniform:MR0.70	0.1021	Credge:Muuniform:MR0.90	0.0921

Table 39: Coefficients for the time model for order-based GCAs on Ruspini2.

A.5 Order-based and Towns2

The correctness model coefficients for the order-based GCAs on Towns2 are listed in Table 40. This model has a residual deviance of 386.1 on 1252 degrees of freedom and matches the data well (Figure 47).

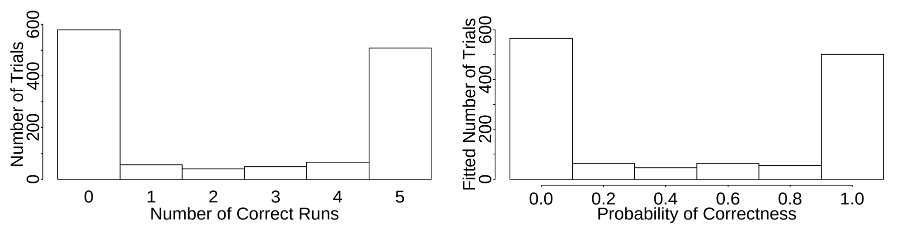


Figure 47: Comparison of experimental and fitted correctness for order-based GCAs on Towns2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	-0.4956	PS100	-0.7545	PS200	-3.4836
Trscale2	-0.7708	Trscale4	-0.6270	El5%	0.0095
Credge	3.4044	CrPMX	-8.9210	CP0.70	0.9022
CP0.90	1.3245	MR0.20	1.6766	MR0.70	4.0323
MR0.90	4.1953	PS100:El5%	-0.6381	PS200:El5%	0.2762
PS100:Credge	8.6976	PS200:Credge	4.5275	PS100:CrPMX	-0.7258
PS200:CrPMX	-4.6320	PS100:CP0.70	0.9303	PS200:CP0.70	0.5591
PS100:CP0.90	1.2453	PS200:CP0.90	2.1117	PS100:MR0.20	-0.5950
PS200:MR0.20	-1.6084	PS100:MR0.70	-1.3705	PS200:MR0.70	-3.3595
PS100:MR0.90	-1.8685	PS200:MR0.90	-3.7982	Trscale2:El5%	1.2432
Trscale4:El5%	1.3430	Trscale2:Credge	13.1436	Trscale4:Credge	13.3841
Trscale2:CrPMX	-5.2360	Trscale4:CrPMX	-5.0491	Trscale2:MR0.20	-1.0720
Trscale4:MR0.20	-1.5813	Trscale2:MR0.70	-6.2452	Trscale4:MR0.70	-6.7731
Trscale2:MR0.90	-6.4426	Trscale4:MR0.90	-6.7489	El5%:MR0.20	1.1547

Table 40: Coefficients for the correctness model for order-based GCAs on Towns2.

The coefficients for the final time model can be found in Table 41. This model has a residual deviance of 1182.3 on 591 degrees of freedom which indicates a relatively poor fit (notice the variations in the histograms in Figure 48). There are no obvious violations of model assumptions in the residual plot (Figure 49).

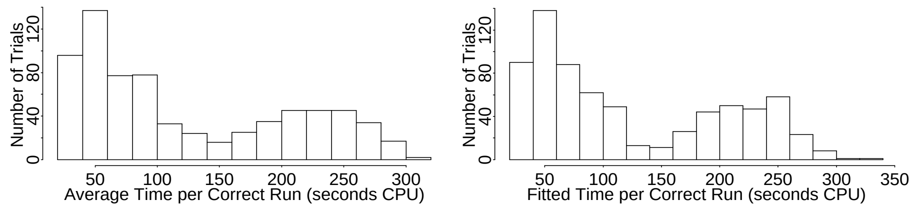


Figure 48: Comparison of experimental and fitted time for order-based GCAs on Towns2.

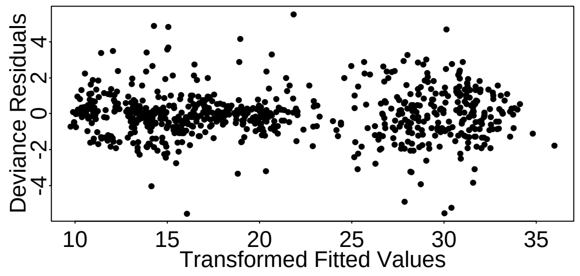


Figure 49: Residual plot for time model for order-based GCAs on Towns2.

Covariate	Effect	Covariate	Effect	Covariate	Effect
(Intercept)	5.5563	PS100	0.0562	PS200	0.1898
Trscale2	-0.1378	Trscale4	-0.0759	El%	-0.0553
Credge	-0.8701	CP0.70	-0.1992	CP0.90	-0.2204
Muuniform	-0.0148	MR0.20	-0.2486	MR0.70	-0.3531
MR0.90	-0.3615	PS100:Trscale2	0.0240	PS200:Trscale2	0.0399
PS100:Trscale4	-0.0308	PS200:Trscale4	0.0796	PS100:El%	0.1720
PS200:El%	-0.0074	PS100:Credge	-0.4012	PS200:Credge	-0.2251
PS100:MR0.20	0.1265	PS200:MR0.20	0.2036	PS100:MR0.70	0.1251
PS200:MR0.70	0.2733	PS100:MR0.90	0.1634	PS200:MR0.90	0.3477
Trscale2:El%	-0.0035	Trscale4:El%	0.0419	Trscale2:Credge	-0.6168
Trscale4:Credge	-0.8433	Trscale2:CP0.70	0.1317	Trscale4:CP0.70	0.2019
Trscale2:CP0.90	0.0594	Trscale4:CP0.90	0.0210	Trscale2:Muuniform	-0.0137
Trscale4:Muuniform	0.0529	Trscale2:MR0.20	0.1850	Trscale4:MR0.20	0.0371
Trscale2:MR0.70	0.4430	Trscale4:MR0.70	0.4368	Trscale2:MR0.90	0.6313
Trscale4:MR0.90	0.5074	El%::Credge	0.1655	El%:MR0.20	0.0461
El%:MR0.70	-0.1289	El%:MR0.90	-0.1345	Credge:CP0.70	-0.3522
Credge:CP0.90	-0.3942	Credge:MR0.20	0.0622	Credge:MR0.70	-0.2034
Credge:MR0.90	-0.2928	CP0.70:MR0.20	0.0830	CP0.90:MR0.20	0.0413
CP0.70:MR0.70	0.0729	CP0.90:MR0.70	0.0965	CP0.70:MR0.90	0.0689
CP0.90:MR0.90	0.0101	PS100:CP0.70	0.0115	PS200:CP0.70	0.0026
PS100:CP0.90	-0.0142	PS200:CP0.90	-0.0024	PS100:Trscale2:Credge	0.4354
PS200:Trscale2:Credge	0.7443	PS100:Trscale4:Credge	0.5537	PS200:Trscale4:Credge	0.7586
Trscale2:Credge:MR0.20	-0.0593	Trscale4:Credge:MR0.20	0.1046	Trscale2:Credge:MR0.70	-0.0405
Trscale4:Credge:MR0.70	0.0963	Trscale2:Credge:MR0.90	-0.0029	Trscale4:Credge:MR0.90	0.0879
PS100:Credge:MR0.20	-0.0484	PS200:Credge:MR0.20	-0.0461	PS100:Credge:MR0.70	0.2357
PS200:Credge:MR0.70	0.2744	PS100:Credge:MR0.90	0.3594	PS200:Credge:MR0.90	0.4159
PS100:Credge:CP0.70	0.1437	PS200:Credge:CP0.70	0.2552	PS100:Credge:CP0.90	0.1759
cont.					

cont. Covariate	Effect	Covariate	Effect	Covariate	Effect
PS200:Credge:CP0.90	0.2745	PS100:Trscale2:EI%	-0.0562	PS200:Trscale2:EI%	0.0823
PS100:Trscale4:EI%	-0.1320	PS200:Trscale4:EI%	0.0462	PS100:Trscale2:MR0.20	0.0466
PS200:Trscale2:MR0.20	-0.0761	PS100:Trscale4:MR0.20	0.1184	PS200:Trscale4:MR0.20	-0.0847
PS100:Trscale2:MR0.70	-0.0321	PS200:Trscale2:MR0.70	-0.2954	PS100:Trscale4:MR0.70	-0.0818
PS200:Trscale4:MR0.70	-0.3769	PS100:Trscale2:MR0.90	-0.2367	PS200:Trscale2:MR0.90	-0.5109
PS100:Trscale4:MR0.90	-0.1274	PS200:Trscale4:MR0.90	-0.5116	PS100:EI%:Credge	-0.1347
PS200:EI%:Credge	-0.2354	PS100:EI%:MR0.20	-0.1244	PS200:EI%:MR0.20	-0.0280
PS100:EI%:MR0.70	-0.0025	PS200:EI%:MR0.70	0.1395	PS100:EI%:MR0.90	-0.0079
PS200:EI%:MR0.90	0.1117	Trscale2:CP0.70:MR0.20	-0.1119	Trscale4:CP0.70:MR0.20	-0.2020
Trscale2:CP0.90:MR0.20	-0.1278	Trscale4:CP0.90:MR0.20	0.0395	Trscale2:CP0.70:MR0.70	-0.0482
Trscale4:CP0.70:MR0.70	-0.0953	Trscale2:CP0.90:MR0.70	0.0207	Trscale4:CP0.90:MR0.70	-0.0181
Trscale2:CP0.70:MR0.90	-0.0919	Trscale4:CP0.70:MR0.90	-0.0754	Trscale2:CP0.90:MR0.90	-0.0206

Table 41: Coefficients for the time model for order-based GCAs on Towns2.