UTRECHT UNIVERSITY

MASTER THESIS

Solving the clustering problem using greedy partitioning and linkage tree approaches

Author:
D. M. ZWEIJE BSc

Supervisor:
dr. ir. D. THIERENS

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Decision Support Systems
Department of Information and Computing Sciences

October 13, 2017

Abstract

Clustering is a process of dividing data into groups of similar objects. Each such group (a cluster) contains objects that are similar to each other and dissimilar to objects in other groups. The clustering problem is widespread in day-to-day life where data can be found, mined, or generated for most situations imaginable. To make managing or finding data easier, it is a useful practice to group, or cluster, data that is in some way similar. In this thesis, a genetic algorithms approach using a greedy partitioning crossover operator and a genetic algorithms approach using a linkage tree for recombination are studied.

Solutions are represented as arrays of n integer cluster labels, where n represents the number of data points. As a distance metric the sum of squared errors is used. Two different local search operators are studied. One operator looks at all elements in each solution once and places the element in the best cluster. The second version continues to improve the solution until an optimum is reached. Various data sets are used in the comparative experiments. In these data sets, the number of data points ranges from dozens to thousands, the number of variables ranges from two to dozens, and the number of clusters ranges from two to ten.

Data from the experiments shows that the greedy partitioning crossover approach and the linkage tree genetic algorithm approach can both solve instances of the clustering problem. However, the simpler multi-start local search approach solves the same instances to the same degree or better, in a more cost-effective way. We have gained valuable insight into the relation between these approaches and their performances.

Abbreviations

FOS	**F**amily **O**f **S**ubsets: set of subsets of the problem variables used to model dependencies between variables.
GA	**G**enetic **A**lgorithm: subset of evolutionary algorithms using a natural-evolution approach.
GGA	**G**rouping **G**enetic **A**lgorithm: class of evolutionary algorithms modified specifically to solve grouping problems.
GOMEA	**G**ene-pool **O**ptimal **M**ixing **E**volutionary **A**lgorithm: model-based evolutionary algorithm using the nodes of a FOS structure as crossover masks.
GPX	**G**reedy **P**artitioning **Cros**sover algorithm: GA using a greedy crossover selecting the fittest clusters in the recombination step first. Uses a LS operator that improves the solution for one iteration over all elements.
LS	**L**ocal **S**earch: algorithm that optimizes a solution for a combinatorial optimization problem by walking through the search space from one solution towards better neighbouring solutions.
LT	**L**inkage **T**ree: a FOS model representing a hierarchical clustering of the dependencies between variables. See FOS.
LTGA	**L**inkage **T**ree **G**enetic **A**lgorithm: gene-pool optimal mixing evolutionary algorithm incorporating the LT as its FOS structure. Uses a LS operator that improves the solution for one iteration over all elements.
MLS	**M**ulti-start **L**ocal **S**earch.
optGPX	**G**reedy **P**artitioning **Cros**sover algorithm using a LS operator that continues until a(n) (local) **opt**imum is found. See GPX.
optLTGA	**L**inkage **T**ree **G**enetic **A**lgorithm using a LS operator that continues until a(n) (local) **opt**imum is found. See LTGA.
RI	**R**and **I**ndex: measures similarity to known optimal solution. A supervised measure.
SSE	**S**um of **S**quared **E**rrors criterion: measures cohesion of clusters.

Contents

List of Figures

List of Tables

1 Introduction

The clustering problem is the problem of finding a way to divide data into groups of similar objects. It is widespread in day-to-day life where data can be found, mined, or generated for most situations imaginable. To make managing or finding data easier, it is a useful practice to group, or cluster, data that is in some way similar. As the number of possible data sets grows and the data sets become larger in both number of data points and variables, this automation of this process through clustering algorithms is increasingly important. Many different approaches have been proposed in the past decades, indicating that this problem is neither new nor solved. Chapter 2 gives an overview of many proposed approaches.

This thesis focuses on a genetic algorithm (GA) approach to solve the clustering problem. This research was inspired by the grouping genetic algorithm (GGA) proposed in (Agustın-Blas et al., 2012). Agustín-Blas et al. proposed a viable approach to solve the clustering algorithm. We will continue with the general outline of a GA such as proposed in (Agustın-Blas et al., 2012), but look at a greedier approach using a fairly simple recombination operator using knowledge about the fitness of individual clusters in a solution as opposed to using the fitness of the entire solution: the GPX approach. This approach is explained in Section 3.3. It has been used effectively in genetic algorithms for solving the graph colouring problem (Galinier and Hao, 1999; Glass and Prügel-Bennett, 2003). It is expected that a modified version of this operator can be used to solve the clustering problem.

Additionally, we have seen that the linkage tree genetic algorithm (LTGA) learns dependency between problem variables and solves many problems successfully (Thierens, 2010; Thierens and Bosman, 2011; Bosman and Thierens, 2012). We study the LTGA algorithm, using the same problem representation as the GGA proposed in (Agustın-Blas et al., 2012).

This then leads us to the focus of this thesis research. We investigate whether genetic algorithms, using the same problem representation as the GGA presented in (Agustın-Blas et al., 2012) are a viable approach to solve the clustering problem.

Our research questions can then be defined as:

Can we use a greedy partitioning crossover approach to solve the clustering problem?

Can we use a linkage tree genetic algorithm to solve the clustering problem?

We would like to improve the results found by Agustín-Blas et al., but are mostly interested in how these different approaches compare to each other. To this end, we compare the proposed GPX and LTGA approaches with each other and a simple multi-start local search (MLS) approach. The comparison to this last simple approach is to make sure the studied algorithms benefit from their specific ways of dealing with information about clusters and generating solutions based on that information, as opposed to just starting from random solutions. This thus leads us to the following research question which will also be used to answer the aforementioned questions:

> *How do the performances of the GPX, LTGA, and MLS approaches in solving the clustering problem compare?*

In the rest of this chapter, we will give an overview of the clustering problem and the general concepts it entails. In Section 1.1, we will define clustering. In Section 1.2, we will describe some basic vocabulary that is used in literature on the clustering problem and approaches. Then, we will give an overview of the clustering process in Section 1.3. Following this, we will indicate some of the different data types used in clustering in Section 1.4. An overview on some similarity measures is given in Section 1.5. Finally, the structure of this thesis is presented in Section 1.6.

1.1 Definition

Clustering is a process of dividing data into groups of similar objects. Each such group (a cluster) contains objects that are similar to each other and dissimilar to objects in other groups. It is also known as data clustering, clustering analysis, segmentation analysis, taxonomy analysis, or unsupervised classification (Gan, Ma, and Wu, 2007). It is important to note the difference between supervised learning and unsupervised learning processes. Supervised learning uses *a priori* labelled data to classify newly encountered data, whereas in unsupervised learning, no *a priori* labelled data is available and the cluster structure must be inferred from the data alone (Goebel and Gruenwald, 1999). Looking at this from a machine learning perspective, resulting the resulting clustering system represents a data concept. Thus, clustering can be seen as the unsupervised learning of a hidden data concept (Berkhin, 2006; Dalal and Harale, 2011).

1.2 Vocabulary

A single data item may be called a data point, pattern, observation, object, item, datum or feature vector. It is mostly represented as a vector of d features $X =$

$(x_1, ..., x_d)$. Thus, the dimensionality of a feature vector is described by d. Individual scalar components x_i of a pattern X in high-dimensional spaces may be denoted as variable, attribute, or feature (Jain, Murty, and Flynn, 1999; Gan, Ma, and Wu, 2007). Cluster, group, and class are used interchangeably in clustering literature. The number of clusters is typically denoted by k.

Distance measures quantify the similarity or dissimilarity of clusters. Distance measures are specialisations of proximity measures (Jain, Murty, and Flynn, 1999). They are metrics or quasi-metrics on the features space and play an important role in clustering (Anderberg, 1973; Jain and Dubes, 1988). Every clustering algorithm is based on an index of this. When no such distance measure exists, cluster analysis will have no meaningful results (Gan, Ma, and Wu, 2007).

1.3 Clustering Process

Clustering algorithms typically include the following four steps (Buhmann, 1995; Jain and Dubes, 1988):

1. Representation of the data
2. Definition of model and proximity measure
3. Clustering
4. Validation of the output

In the representation step, the structure of the clusters is determined. This includes, for example, the number of clusters to be found, the size of the data set, dimensionality of the data, and details on the features such as type and scale. Optionally, feature selection or feature extraction may be used here to obtain an appropriate set of features to use in the clustering step. In the definition step, cluster structure and criteria that separate clusters are defined. Also, a proximity measure is defined that is used in the next step. Different approaches to the clustering step will be explained in Chapter 2.

It may occur that values are missing from the data set. Missing data can be divided into three categories (Fujikawa and Ho, 2002): (1) in some attributes, (2) in a number of patterns, and (3) randomly. If one attribute or pattern misses all values, that attribute or pattern should be removed from the data set (Rousseeuw and Kaufman, 1990). It the number of missing values is limited, there are two ways to deal with missing values (Fujikawa and Ho, 2002): (1) replace the missing values before the clustering starts, or (2) deal with missing values during clustering. Thus, there may be a preprocessing step before the aforementioned steps if many values are missing in the data set.

1.4 Data Types

Data-clustering algorithms depend on the data types that need to be handled by them. A data type can be defined as the degree of quantization in the data (Anderberg, 1973; Jain and Dubes, 1988). Attributes can be categorized as being discrete or continuous. Discrete attributes have a finite number of possible values, while continuous attributes have an infinite number of possible values.

Attributes can be defined as either quantitative or qualitative. Quantitative attributes are associated with numerical data, while qualitative attributes are associated with categorical data. The following subdivision can be made (Gowda and Diday, 1992):

1. quantitative features:
   (a) continuous values (e.g., weight)
   (b) discrete values (e.g., the number of computers)
   (c) interval values (e.g., the duration of an event)
2. qualitative features:
   (a) nominal or unordered (e.g., colour)
   (b) ordinal (e.g., military rank or qualitative evaluations of temperature ("cool" or "hot"))

A special categorical type of attributes is the binary attribute. Binary attributes have exactly two values. Examples include *true or false*, *male or female*, and *inclusive or exclusive*.

In real life applications, various more complex data types exist, for example image data or spatial data. In addition, attributes of a single data point may be of different data types. For such data sets, the chosen similarity or dissimilarity measures need special thought.

1.5 Similarity Measures

In this section, we describe some similarity measures as used in clustering algorithms. Much of this information comes from (Gan, Ma, and Wu, 2007). How similar (or dissimilar) two data points or clusters are is normally measured using a distance function. Similarity is also called proximity or distance. A similarity coefficient indicates how strong the relationship between two data points is (Everitt, Landau, and Leese, 1993). Similarity between two data points is usually symmetric, i.e., $s(x, y) = s(y, x)$. Such measures are key to clustering algorithms.

Four strict rules define whether some measure is a metric or not: nonnegativity, reflexivity, commutativity, and triangle inequality (Anderberg, 1973; Zhang and Srihari, 2003).

1. nonnegativity: $f(x, y) \geq 0$
2. reflexivity: $f(x, y) = 0 \Leftrightarrow x = y$
3. commutativity: $f(x, y) = f(y, x)$
4. triangle inequality: $f(x, y) \leq f(x, z) + f(y, z)$

where x, y, and z are arbitrary data points in the set.

Functions that are like metrics but do not have all properties are pseudo-metrics. The specific properties of such a function are the following (Rousseeuw and Kaufman, 1990):

1. $0 \leq s(x, y) \leq 1$
2. $s(x, x) = 1$
3. $s(x, y) = s(y, x)$

where x and y are arbitrary data points in the set.

1.5.1 Numerical Data

The Minkowsky distance metric is a measure for numerical data with many different uses:

$$d_p(x_i, x_j) = \sum_{k=1}^{d} (|x_{i,k} - x_{j,k}|^p)^{1/p} = ||x_i - x_j||_p$$

Well-known and often-used versions of the Minkowsky distance are the Euclidean distance ($p = 2$), the Manhattan distance ($p = 1$), and the maximum distance ($p = \infty$). The Euclidean distance is an intuitive measure and is often used to measure distance of objects in two or three-dimensional space. The Minkowsky distance has a drawback: if some feature is much larger-scaled than the other features, this dominates the distance. This problem can, for example, be solved by normalising the numerical features to a common range, such as $[0, 1]$.

1.5.2 Categorical Data

Measures for categorical data have not received as much attention as measures for numerical data. One simple and well-known measure for categorical data is the simple matching distance (Rousseeuw and Kaufman, 1990; Huang, 1997; Huang, 1998). It is defined as:

$$\delta(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$$

Let X and Y be two categorical objects described by d attributes. The dissimilarity between X and Y is defined by:

$$d_{sim}(X,Y) = \sum_{j=1}^{d} \delta(x_j, y_j)$$

If we then take the frequencies of categories in the data set into account, we can define the dissimilarity measure as:

$$d_{simf}(X,Y) = \sum_{j=1}^{d} \frac{n_{x_j} + n_{y_j}}{n_{x_j} n_{y_j}} \delta(x_j, y_j)$$

where n_{x_j} and n_{y_j} are the amounts of objects in the data set that have categories x_j and y_j, respectively.

1.5.3 Measures between Clusters

It may be required that not the distance between data points is evaluated, but the distance between clusters. Some measures include:

- Mean-based Distance
  This is a popular measure to evaluate dissimilarity between clusters for numerical data. In this measure, the distance between the means of two clusters is used.
- Nearest Neighbour Distance
  This measure looks at the minimal distance between pairs of data points of two clusters and takes this as a measure (Williams and Lambert, 1966).
- Farthest Neighbour Distance
  This measure looks at the maximal distance between pairs of data points of two clusters and takes this as a measure (Duran and Odell, 1974).

In the end, which similarity or dissimilarity measures are appropriate depends on the types of variables and their measurement levels. Additional information on similarity and dissimilarity measures can be found in (Cattell, 1949; Sokal and Sneath, 1963; Goodall, 1966; Hartigan, 1967; Green and Rao, 1969; Sneath and Sokal, 1973; Hubalek, 1982; Gower and Legendre, 1986; Baulieu, 1989).

1.6 Structure

This section details the structure of the thesis. In Chapter 1, the clustering problem is defined and the research focus is detailed. Chapter 2 describes various techniques previously and currently used in the field in the form of a literature study. The various algorithms used in the experiments are explained in Chapter 3. Then, Chapter 4 notes the scope of the experiments and details the experiments performed in the study. The results of the experiments are stated in Chapter 5. Following the results, conclusions and a discussion are presented in Chapter 6.

2 Clustering Techniques: Literature Study

Clustering algorithms can be categorized in many ways. There is no straightforward and/or canonical way to do this. Such groups can also overlap. Common subdivisions include (Jain, Murty, and Flynn, 1999):

- Hierarchical and partitioning: Hierarchical clustering algorithms build clusters gradually over multiple levels, while partitioning clustering algorithms create a one-level clustering.
- Agglomerative and divisive: Agglomerative methods work bottom-up, starting with one cluster for each object and merging those until a stopping criterion is met. Divisive methods work top-down, starting with one cluster of all data points and splitting until a stopping criterion is met.
- Monothetic and polythetic: Polythetic methods consider all vector features at once; most clustering algorithms take this approach. Monothetic methods look at the vector features sequentially.
- Hard and fuzzy: In hard clustering, all objects are assigned to exactly one cluster. Such approaches find strict partitions and thus result in disjoint clusters. In fuzzy clustering, all objects are assigned degrees of membership in several clusters. In practice, fuzzy clusters often make sense: each object can then belong to multiple clusters with some probability. A membership function is used to assign this probability (Zadeh, 1965). The clusters fuzzy clustering algorithms have as output are not partitions. A book discussing the fuzzy clustering problem was written by (Höppner, 1999).
- Deterministic and stochastic: Traditional techniques use a deterministic method to cluster the data, but random search of the state space of all labellings may be used as a stochastic method.
- Incremental and non-incremental: If constraints on execution time and memory space are a problem, incremental methods may be employed to solve this problem. If not, non-incremental methods suffice.

In this chapter, we present a categorization of clustering algorithms based on a literature study. It is of importance to note that in our research, we focus on some specific approaches, namely Genetic Algorithms (GAs). These algorithms fall in the probability-based approaches category, presented in Section 2.3.

This chapter is structured as follows. In Section 2.1, hierarchical clustering approaches are discussed. In Section 2.2, centre-based approaches are presented. Then, we will discuss different probability-based approaches in Section 2.3. Following this, we will present some density-based clustering approaches in Section 2.4. Finally, in Section 2.5, we will review subspace clustering approaches.

2.1 Hierarchical Clustering

In the hierarchical clustering approach, a cluster hierarchy, or dendrogram, is built. A horizontal slice in the hierarchy represents a certain partitioning. This kind of approach enables different granularity levels to be investigated. Hierarchical clustering methods are divided into two categories: agglomerative and divisive (Jain and Dubes, 1988; Rousseeuw and Kaufman, 1990). Agglomerative hierarchical clustering works bottom-up: it starts with one cluster for each object and recursively merges the two closest clusters according to a chosen similarity measure until a stopping criterion is met. Divisive hierarchical clustering works top-down: it starts with one cluster containing all objects and recursively splits the most appropriate clusters until a stopping criterion is met.

Advantages of hierarchical clustering include (Berkhin, 2006; Dalal and Harale, 2011):

- embedded flexibility regarding the level of granularity
- ease of handling of any forms of similarity or distance
- good result visualizations are integrated into the methods
- applicability to any attribute type

Disadvantages of hierarchical clustering are related to (Berkhin, 2006; Gan, Ma, and Wu, 2007):

- vagueness of termination criteria
- most hierarchical algorithms do not revisit once (possibly incorrectly) constructed clusters
- different similarity measures for measuring similarity between clusters may lead to different results
- the computational complexity is generally higher than that of partitional algorithms

Finding the appropriate clusters to merge or split is done depending on the similarity or dissimilarity of objects in the clusters. So, as a certain similarity between objects in a cluster is assumed, the (dis)similarity between clusters can be used rather than (dis)similarity between individual objects. This generalization is called a linkage metric. The way in which this linkage metric is derived affects closeness and connectivity. Important inter-cluster linkage metrics include single-link, average-link, and complete-link, which are calculated for all pairs of objects in the different sets (Murtagh, 1985; Olson, 1995). Such methods are called graph methods. Other linkage metric methods are geometric methods. In geometric methods, a cluster is represented by a central point instead of individual objects. Geometric linkage metrics include centroid, median, and minimum variance. All linkage metrics for hierarchical clustering have, under reasonable assumptions, a time complexity of $O(n^2)$ (Olson, 1995).

We will now give a non-exhaustive list of hierarchical clustering algorithms (Gan, Ma, and Wu, 2007):

- BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is an agglomerative algorithm (Murtagh, 1983). It was designed to cluster very large numerical data sets in Euclidean space.
- CHAMELEON is an agglomerative algorithm (Karypis, Han, and Kumar, 1999). It utilizes dynamic modelling in cluster aggregation. It merges two clusters only if interconnectivity and closeness between them are high enough, relative to internal interconnectivity and closeness.
- CLINK is based on a compact representation of a dendrogram (Defays, 1977).
- COBWEB is an algorithm for categorical data (Fisher, 1987). It utilizes incremental learning and also belongs to conceptual or model-based learning. The dendrogram created is called a classification tree. COBWEB does reconsider clustering decisions.
- CURE (Clustering Using REpresentatives) is an agglomerative algorithm that is capable of identifying non-spherical shaped clusters in large databases and with wide variances in size (Guha, Rastogi, and Shim, 1998). It is robust to outliers.
- DIANA (DIvisive ANAlysis) can be applied to all data sets that van be clustered by means of the agglomerative hierarchical algorithms (Rousseeuw and Kaufman, 1990).
- DISMEA is a divisive algorithm that uses the k-means algorithm to subdivide a cluster into two (Späth, 1980).
- Edwards and Cavalli-Sforza Method is a divisive algorithm, based on splitting clusters into two new clusters to maximize the inter-cluster sum of squares (Edwards and Cavalli-Sforza, 1965).

- ROCK is an agglomerative algorithm for categorical attributes (Guha, Rastogi, and Shim, 1999). It is based on the number of links between records, not on any distance function.
- Single-link Algorithms Based on Minimum Spanning Trees is a single-link algorithm that can be carried out by using only the information contained in the minimum spanning tree (Gower and Ross, 1969).
- SLINK is a single-link algorithm that can be carried out using arbitrary dissimilarity coefficients (Sibson, 1973).

Some papers discussing hierarchical clustering methods and algorithms can be found in (Murtagh, 1983; Gordon, 1987; Willett, 1988; Rousseeuw and Kaufman, 1990; Gordon, 1996).

2.2 Centre-based Clustering

Centre-based clustering algorithms can be divided into two approach types: centroid and medoid (Dalal and Harale, 2011). In centroid approaches, clusters are represented by the gravity centre of the data points in that cluster. In medoid approaches, clusters are represented by the means of the data points closest to the gravity centre. Centre-based clustering algorithms are very efficient for large and high-dimensional databases (Gan, Ma, and Wu, 2007). These algorithms find convex-shaped clusters (Anderberg, 1973). Thus, when clusters of arbitrary shapes have to be found, the centre-based approach is likely not the best choice.

The k-means algorithm is the most well-known centre-based centroid clustering algorithm (MacQueen, 1967; Wagstaff et al., 2001). Even though it was proposed over 50 years ago, the k-means algorithm is still one of the most popular clustering algorithms (Jain, 2010). It is simple and straightforward. Beside the efficiency of centre-based algorithms in general, k-means often terminates at a local optimum (Anderberg, 1973; Selim and Ismail, 1984) and is sensitive to outliers (Dalal and Harale, 2011). It is not as effective for high-dimensional data as some other algorithms (Hinneburg and Keim, 1999). As the k-means algorithm takes the centroid approach to represent clusters, it does not work well with categorical attributes, while it does work well with numerical attributes (Berkhin, 2006). Determining the right k number of clusters is no trivial task.

Fuzzy k-means is an extension of the traditional k-means algorithm for fuzzy clustering (Bezdek, 1973). Many other extensions or improvements for the k-means algorithm can be found in (Faber, 1994; Bradley and Fayyad, 1998; Kanungo et al., 2002; Bottou and Bengio, 1995).

Some other centre-based centroid clustering algorithms include (Berkhin, 2006; Gan, Ma, and Wu, 2007):

- k-modes is derived from the k-means algorithm and was proposed to cluster categorical data (Ye, 2003). First, k initial modes are selected, after which all data points are assigned to their respective nearest mode. A parametric version (Huang, 1997; Huang, 1998) and a non-parametric version (Chaturvedi, Green, and Caroll, 2001) were proposed.
- Fuzzy k-modes is an extension of the traditional k-modes algorithm for fuzzy clustering (Huang and Ng, 1999).
- k-prototypes originates from k-modes and k-means, designed for the clustering of data sets with mixed attribute types (Huang, 1998; Jain and Dubes, 1988).
- k-probabilities extends the k-modes algorithm and was, like k-prototypes, designed for the clustering of data sets with mixed attribute types (Wishart, 2003).
- FCM (Fuzzy C-Means) is a fuzzy centroid based clustering algorithm (Bobrowski and Bezdek, 1991; Wagstaff et al., 2001; Bezdek, 2013).

Some centre-based medoid clustering algorithms include (Berkhin, 2006; Nagpal, Jatain, and Gaur, 2013):

- k-medoids methods represent clusters by one data point. This enables the algorithm to deal with all attribute types. Selecting medoids, clusters are defined as groups of points close to said medoids.
- PAM (Partitioning Around Medoids) is an iterative algorithm, improving found clusters with each step (Rousseeuw and Kaufman, 1990). Medoids are first created by determining a representative data point per cluster. Dissimilarity to all non-centre data points is then calculated, upon which clustering is then based. Medoids can be re-assigned when an improvement is found.
- CLARA (Clustering LARge Applications) is an algorithm that uses sampling to select a subset of data points and then uses PAM to select medoids (Ng and Han, 2002). It uses multiple sample subsets and presents the best clusters found from the sample sets.
- CLARANS (Clustering Large Applications based on RANdomized Search) combines PAM and CLARA into a more efficient and effective algorithm (Sarle, 1991). The drawback is that if the data set is large enough that not all data points can be stored in main memory, the run time increases.

2.3 Probability-based Clustering

Most clustering techniques described in the previous sections are deterministic. Algorithms based on those techniques guarantee a local optimal solution. In contrast, stochastic techniques cannot guarantee an optimal solution. However, they generate near-optimal solutions quickly. Also, convergence to optimal solutions is guaranteed asymptotically (Jain, Murty, and Flynn, 1999). Stochastic approaches allow for perturbations in directions that are non-optimal (locally) with non-zero probabilities. We mention three different (possibly overlapping) points of view for probability-based clustering in the following sections. Section 2.3.1 describes search-based clustering, Section 2.3.2 pertains evolution-based clustering and finally, Section 2.3.3 details model-based clustering.

2.3.1 Search-based Clustering

A well-known search technique is simulated annealing. This technique has been used to solve clustering problems (Klein and Dubes, 1989; McErlean, Bell, and McClean, 1990; Bell et al., 1990; Selim and Alsultan, 1991; Hua, Lang, and Lee, 1994). The perturbation operator in simulated annealing is similar to the k-means scheme: it relocates a data point from its current to a randomly chosen other cluster (Berkhin, 2006). For example, SARS (Simulated Annealing using Random Sampling) takes the simulated annealing approach, based on decomposition (Hua, Lang, and Lee, 1994). For this algorithm to work, the clustering problem is transformed into a graph partitioning problem. SARS explicitly addresses excessive disc access problems during annealing (Gan, Ma, and Wu, 2007).

Tabu search is similar to simulated annealing (Al-Sultan, 1995). Tabu search is a common heuristic algorithm used to solve combinatorial optimization problems (Glover, 1989; Glover, 1990; Glover and Taillard, 1993). Tabu search uses steepest descent to improve solutions. After finding a locally optimal solution, some perturbations are done. A number of recent solutions are recorded in the tabu list. These solutions are not allowed to be visited for a number of iterations. This allows tabu search to escape local optima. Adaptation of tabu search for the clustering problem were proposed in (Al-Sultan, 1995; Sung and Jin, 2000). Some adaptations were also proposed for solving the fuzzy clustering problem (Xu, Wang, and Li, 2002; Ng and Wong, 2002). Also, a tabu search technique was combined with the fuzzy c-means algorithm (Delgado, Skármeta, and Barberá, 1997).

Many search-based clustering algorithms are stochastic algorithms, but some deterministic algorithms exist (Gan, Ma, and Wu, 2007):

- VNS (Variable Neighbourhood Search for Clustering) is a metaheuristic which uses a systematic change of neighbourhood within a local search algorithm (Hansen and Mladenović, 2001).
- Global k-means combines the standard k-means algorithm and a local search procedure (Likas, Vlassis, and Verbeek, 2003). The clustering problem is solved incrementally: intermediate clustering problems with $1, ..., k-1$ clusters are solved sequentially.

2.3.2 Evolution-based Clustering

Evolution-based clustering approaches are inspired by natural evolution. Different evolutionary approaches exist, including (Jain, Murty, and Flynn, 1999): genetic algorithms (GAs) (Holland, 1975; Goldberg, 1989), evolutionary strategies (ESs) (Schwefel, 1981), and evolutionary programming (EP) (Fogel, Owens, and Walsh, 1965). These approaches see the clustering problem as a minimisation of the squared error criterion, described in Section 3.1.3. GAs have been applied for clustering the most out of the evolution-based clustering approaches (Jiang and De Ma, 1996; Maulik and Bandyopadhyay, 2000; Cheng, Lee, and Wong, 2002; Greene, 2003).

GAs use evolutionary operators, such as selection, crossover, and mutation, on a population of solutions. Generally, GAs consist of the following (Gan, Ma, and Wu, 2007): problem encoding, initialisation, selection operator, crossover operator, and mutation operator. The problem encoding is problem dependent. In addition, the evaluation function used to determine the fitness of a particular solution is also problem dependent. However, even for the same problem, different encodings or evaluation functions may be suitable.

The initialisation phase takes care of the (random) construction of the first population. Typically, GAs then iteratively create new populations using the genetic operators. The number of generations is specified by the user. The crossover operator takes parent solutions and combines these into new child solutions. The mutation operator takes a solution and modifies it slightly with a certain probability. The selection operator selects a number of new solutions based on survival of the fittest.

The best found solution is stored separately from the population and will be the output at the end of all iterations. The sensitivity to the selection of parameters is a major problem in the use of GAs (Jain, Murty, and Flynn, 1999). A survey of GAs can be found in (Ribeiro Filho, Treleaven, and Alippi, 1994).

Multiple hybrid approaches were proposed in literature (Jain, Murty, and Flynn, 1999; Gan, Ma, and Wu, 2007):

- A GA was used to find good initial cluster centres, after which the k-means algorithm was used for the final clustering (Babu and Murty, 1993). This hybrid approach outperformed the pure GA.
- GKA finds the globally optimal partition of data set into a given number of clusters (Krishna and Murty, 1999). It is proven to converge to optimality using the finite Markov chain theory.
- GKMODE (Genetic k-MODEs) is similar to GKA, but uses the k-modes operator and allows for illegal strings (Gan, Yang, and Wu, 2005).

- Genetic Fuzzy k-modes is the fuzzy version of GKMODE. It uses a one-step fuzzy k-modes algorithm instead of the crossover operator to speed up convergence.

2.3.3 Model-based Clustering

Model-based clustering algorithms assume that the data points can be classified using a mixture of probability distributions, where each such distribution corresponds to a different cluster. *Model* is often used to represent the type of constraints and geometric properties of the covariance matrices (Martinez et al., 2010). Model-based clustering algorithms attempt to optimize the fit between models and data. This means that the more the data conforms to the model, the better model-based clustering algorithms perform.

Some common model-based clustering techniques are described below (Gan, Ma, and Wu, 2007; Nagpal, Jatain, and Gaur, 2013):

- EM (Expectation Maximization) is a general statistical algorithm (McLachlan and Basford, 1988; Krishnan and McLachlan, 1997). It can be used in the presence of incomplete data for maximum likelihood estimation. It can be seen as an extension to k-means. EM is simple, stable, and robust to noise.
- Gaussian Mixture Models are a classical and powerful approach to clustering analysis (Banfield and Raftery, 1993).
- COBWEB creates a hierarchical clustering through simple incremental conceptual learning. The number of classes is automatically adjusted.
- CLASSIT extends COBWEB for continuous data clustering.
- Auto Class estimates the optimal number of clusters through Bayesian statistical analysis.
- COOLCAT clusters categorical attributes using entropy (Barbará, Li, and Couto, 2002). In an initialization step, a sample from the data set is used to find a set of cluster, after which all other data points are assigned to these clusters in an incremental step.
- STUCCO uses newly defined contrast-sets to find meaningfully different groups. A subset is selected from significant contrast-sets, selected from all possible combinations of attribute values using a tree searching method.
- SOM (Soft-Organizing Map) uses an incremental approach to map high-dimensional data points to a 2-dimensional or 3-dimensional space, preserving distance and proximity where possible (Kohonen, 1990). The visualisation of a SOM is straightforward because of this mapping.

- Artificial Neural Networks (ANNs) are based on their biological counterparts (Hertz, Krogh, and Palmer, 1991). ANNs only process numerical vectors, are inherently parallel, and may learn interconnection weights adaptively (Oja, 1992; Jain, Mao, and Mohiuddin, 1996; Mao and Jain, 1996). Learning systems, however, suffer from issues in balancing plasticity and stability.

As the EM algorithm is so popular, a framework for model-based clustering was based on it. This framework consists of three steps (Martinez et al., 2010):

1. Initialize EM using the partitions found by model-based agglomerative hierarchical clustering.
2. Estimate parameters using EM.
3. Choose the model and the number of clusters according to the BIC.

2.4 Density-based Clustering

In this section, we describe density-based clustering techniques. We first discuss general density-based clustering in Section 2.4.1. Then, we look at grid-based clustering in Section 2.4.2.

2.4.1 General Density-based Clustering

Density-based clustering algorithms are based on the idea that density of data within a cluster is higher than density of data outside a cluster. Clusters grow in any direction, based on density alone. Thus, density-based clustering algorithms are able to find clusters of arbitrary shapes. Outliers also do not disturb density-based algorithms. Knowing the number of clusters beforehand is not necessary, since density-based clustering algorithms can find the natural number of clusters automatically (El-Sonbaty, Ismail, and Farouk, 2004). In general, scalability is very good, but interpretability is worse than for other clustering approaches (Berkhin, 2006). Choosing the density threshold well is of high importance, and a difficult task. Also, a metric space is required, so spatial data clustering is the main application (Han, Kamber, and Tung, 2001; Kolatch, 2001).

Two major approaches for density-based clustering algorithms can be identified (Berkhin, 2006). In the first approach, density is pinned to training data points. In the second approach, density is pinned to a point in the attribute space.

Algorithms for the first approach include:

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is one of the most commonly known density-based algorithms (Ester et al., 1996).

It targets low-dimensional data and for the proximity of data, the Euclidean distance is used.

- OPTICS (Ordering Points To Identify the Clustering Structure) extends DBSCAN by automatically adjusting its parameters to different parts of the data in the clustering process (Ankerst et al., 1999).
- DBCLASD (Distribution-Based Clustering of LArge Spatial Databases) assumes that data points inside a cluster are uniformly distributed (Xu et al., 1998). It is a non-parametric approach that dynamically detects appropriate cluster count and shapes. As the name suggests, the algorithm targets larger databases.
- BRIDGE is a hybrid clustering algorithm, combining k-means and DBSCAN (Dash, Liu, and Xu, 2001). This approach helps deal with larger databases and deals with noise.

Algorithms for the second approach include DENCLUE (DENsity-based CLUstEring) (Hinneburg and Keim, 1998). DENCLUE focuses on its density function's local maxima and uses gradient hill-climbing to find these local maxima. The density function used in DENCLUE is a superposition of multiple influence functions (Berkhin, 2006). The algorithm is very robust regarding noise.

2.4.2 Grid-based Clustering

Grid-based clustering algorithms are also based on density (Berkhin, 2006; Gan, Ma, and Wu, 2007; Mann and Kaur, 2013). Grid-based clustering pertains to the value space surrounding the data instead of the data points themselves. Such algorithms use a multi-resolution grid data structure and dense grids for cluster formation. Thus, the infinite amounts of data are first quantized into a finite number of grid cells. All operations can then be performed on this finite number of cells instead of the data points themselves, making the run-time independent of the number of data points in the original data set and thus achieving a fast processing time, even for large data sets. This reduction in computational complexity is especially apparent when clustering very large data sets. Choosing grid size or density thresholds is a difficulty of these algorithms. Techniques of adaptive grids, automatically determining grid size based on data distribution, are proposed to help solve this problem (Nagesh, Goil, and Choudhary, 2001). Grid-based clustering algorithms are independent of data ordering.

Some common grid-based clustering algorithms include (Berkhin, 2006; Gan, Ma, and Wu, 2007; Nagpal, Jatain, and Gaur, 2013):

- STING (STatistical INformation Grid approach) was developed for spatial database clustering (Wang, Yang, and Muntz, 1997). Various levels of detail can be investigated and this results in a very scalable algorithm.

- WaveCluster applies wavelet transforms on the feature space to filter data (Sheikholeslami, Chatterjee, and Zhang, 1998). While it is able to deal with noise, able to detect clusters of arbitrary shapes, and efficient for large databases, it is not suitable for very high-dimensional data sets (Andritsos, 2002). It works on numerical data.
- CLIQUE is a grid-based and density-based algorithm developed to cluster high-dimensional data (Agrawal et al., 1998).
- BANG-clustering is a hierarchical clustering algorithm (Schikuta and Erhart, 1997). The algorithm works with grid-based segments, stored in a so-called special BANG-structure. From this structure, a dendrogram can be calculated.
- FC (Fractal Clustering) is an algorithm for numeric attributes (Barbará and Chen, 2000). It uses an incremental structure, which is beneficial for memory, but is data order dependent.
- GDILC (Grid-based Density-IsoLine Clustering) uses the concept of density-isoline figures to depict the distribution of data samples (Zhao and Song, 2001).
- GRIDCLUS is a hierarchical algorithm developed for large data set clustering (Schikuta, 1996).

2.5 Subspace Clustering

The high dimensionality of modern-world data sets sometimes makes it infeasible to identify clusters in the entire data space as conventional clustering algorithms do. Difficulties encountered by conventional clustering algorithms because of high-dimensional data include: more limitations on distance discrimination as distances even out the more dimensions there are (Beyer et al., 1999), and clusters may exist in different subspaces of the data space (Agrawal et al., 1998). Subspace, or projected, clustering finds clusters and their associated attributes from the data set (Gan, Ma, and Wu, 2007).

We can divide subspace clustering approaches into two categories (Parsons, Haque, and Liu, 2004):

- Top-down algorithms that look for an initial clustering in the full data space and then evaluates cluster subspaces.
- Bottom-up algorithms that look for dense regions in low-dimensional spaces and then combine them into clusters.

We will now describe some algorithms in the two categories mentioned above. First, some examples of top-down subspace clustering algorithms are (Gan, Ma, and Wu, 2007):

- PART (Projective Adaptive Resonance Theory) is a neural network architecture proposed to find projected clusters for data sets in high-dimensional spaces (Cao and Wu, 2002).

- PROCLUS (PROjected CLUSting) is a variation of the k-medoid algorithm (Rousseeuw and Kaufman, 1990) in subspace clustering (Aggarwal et al., 1999).

- ORCLUS (arbitrarily ORiented projected CLUSter generation) is an extension of PROCLUS. It diagonalizes the covariance matrix of each cluster and finds information about projection subspaces from the diagonalisation of the covariance matrix (Aggarwal and Yu, 2000).

- FINDIT (a Fast and INtelligent subspace clustering algorithm using DImension voTing) is a subspace clustering algorithm that uses a dimension-oriented distance measure and a dimension voting policy to determine the correlated dimensions for each cluster (Woo et al., 2004).

Some examples of bottom-up subspace clustering algorithms are (Gan, Ma, and Wu, 2007):

- CLIQUE (Clustering In QUEst) was the first subspace clustering algorithm. It is able to identify dense clusters in subspaces of maximum dimensionality (Agrawal et al., 1998).

- ENCLUS (ENtropy-based CLUStering) is an entropy-based subspace clustering algorithm for clustering numerical data. It can find arbitrarily shaped clusters embedded in the subspaces of the original data space (Cheng, Fu, and Zhang, 1999).

- MAFIA (Merging of Adaptive Finite Intervals) is a parallel subspace clustering algorithm using adaptive computation of the finite intervals in each dimension that are merged to explore clusters embedded in subspaces of a high dimensional data set. It is also a density- and grid-based clustering algorithm (Goil, Nagesh, and Choudhary, 1999; Nagesh, Goil, and Choudhary, 2000).

- CLTree (CLustering based on decision Trees) is a clustering algorithm for numerical data based on a supervised learning technique called decision tree construction (Liu, Xia, and Yu, 2000).

- DOC (Density-based Optimal projective Clustering) is a Monte Carlo–based algorithm that computes a good approximation of an optimal projective cluster (Procopiuc et al., 2002).

More information on subspace clustering algorithms can be found in (Krishnapuram and Freg, 1991; Narahashi and Suzuki, 2002; Aggarwal and Yu, 2002; Har-Peled and Varadarajan, 2002; Sarafis, Trinder, and Zalzala, 2003; Amir et al., 2003; Agarwal and Mustafa, 2004; Parsons, Haque, and Liu, 2004; Ke and Kanade, 2004; Kailing, Kriegel, and Kröger, 2004; Wang et al., 2004).

3 Algorithms

The algorithms we study to solve the clustering problem can be categorized as centre-based and probability-based clustering algorithms. In the algorithms, we look for a partitioning clustering and we look at instances with numerical data. Some instances may have categorical data transformed into numerical data, but we treat this as numerical data nonetheless. In our algorithms, we require the number of clusters as an input parameter. This makes sure the researcher has most control over the number of clusters that is found by the algorithms and does not prevent potentially sub-optimal but interesting numbers of clusters from being ruled out by the algorithm automatically.

As stated in the introduction of Chapter 1, the approaches presented in this thesis are mainly inspired by the grouping genetic algorithm (GGA) presented in (Agustın-Blas et al., 2012). This GGA has the following characteristics:

- Problem encoding
  The GGA uses a variable-length encoding. The elements and the groups of a solution are kept track of separately. The encoding length is n (for the number of elements) + k (for the number of groups). Each element gets assigned a number $1, ..., k$.

- Selection operator
  A rank-based wheel selection mechanism was used. The parents are selected for crossover with this operator. It is performed with replacement: individuals can be selected as parents multiple times, but the two parents must be different in the same crossover operation.

- Crossover operator
  Two parents generate one offspring. Two individuals are selected, where two crossing points are chosen in their group part. The elements belonging to the selected groups from the first individual are inserted into the offspring, after which the elements from the selected group of the second individual are inserted into the offspring, if unassigned by the first. Then, unassigned elements are randomly assigned and empty clusters are removed. The labels are then renamed 1 to k. The probability of crossover is high at first and moderate in the final stages of the algorithm.

- Mutation operator
  Two mutation operators are used, both with low probability. The first mutation operator is cluster splitting. A selected cluster is split into two new clusters. The elements are assigned with equal probability. One cluster will keep the old label, the other will get the label $k + 1$. The second mutation operator is cluster merging. This selects two random clusters and merges them into one. For both mutations, the probability to select a cluster depends on the cluster's size. Both operators are applied with low, independent probabilities. The probability of applying a mutation is low at first and moderate in the final stages of the algorithm to gain opportunities of escaping local minima.

- Elitism
  The algorithm always automatically passes the best solution to the next generation directly.

- Local search
  Local search was used to try to find local optima in a close neighbourhood of a solution. It works based on best neighbourhood search. As it is time-consuming, it is done with small probability.

- Island model
  The algorithm uses the island model to improve the performance. An elitist island model was used in which only the best solution of each island migrates and replaces a randomly chosen solution on another island.

In the algorithms discussed in this chapter, we differ from this approach in the following ways:

- We do not require the number of groups in the problem encoding
- The selection operator is different for both algorithms
- The crossover operator is different for both algorithms
- The mutation operator does not exist in either algorithm because its effects are lost when performing local search
- The local search operator is based on first-improvement instead of best-improvement, which reduces the time required to perform it and as such is done each generation
- No island model is used in either of the algorithms

The rest of this chapter is structured as follows. In Section 3.1, the framework upon which both algorithms are based is discussed. Then, the different local search approaches are presented in Section 3.2. Section 3.3 details the greedy partitioning crossover approach. Finally, the linkage tree genetic algorithm is explained in Section 3.4.

3.1 Algorithms Framework

All algorithms used in this study, besides the basic local search algorithm, are based on the same framework. Using this framework for all algorithms enables a clean comparison to be made where difference in implementation could make a difference otherwise.

The following input parameters are used in the algorithm:

- Maximum number of Evaluations
  This value indicates the maximum number of evaluations that are allowed by the algorithm before termination.
- Time Limit
  This value indicates the time limit in milliseconds the algorithm is allowed to take before termination.
- Problem
  This value indicated the problem instance the algorithm should solve.
- Goal Clusters
  This values indicates the number of clusters the algorithm should aim to cluster the data set in.

In addition, the application running the algorithm also takes the number of iterations the algorithm should run to termination and the specific algorithm that should be used to solve the problem instance.

3.1.1 Solution Representation

A solution is represented as an array of n integer cluster labels, where n represents the number of data points. More information is stored about solutions to reduce recalculation cost, but the integer array is the essential part of the representation.

3.1.2 Framework Outline

In the framework, a pool of populations of solutions is used. As seen in Algorithm 1, the algorithm framework provides a multi-population structure, where the largest population size exponentially increases over the generations. In this way, we do not have to provide the algorithms based upon this framework with a population size when running an experiment. Instead, the first population size is set to 1 and the maximum number of populations in the population pool is set to 25. This thus means that the maximal population size is 2^{24} for the 25^{th} population pool is created in the 25^{th} generation. This is a huge population size. However, it is unlikely that this full number of population sizes the algorithm can explore will actually be explored, as the termination criterion makes sure the algorithm is stopped when necessary. The best solution encountered is saved independently from the pool of populations. Every generation, every solution in the population pool is generated and improved as in Algorithm 2 or Algorithm 3. At the end of every generation, the best solution may be updated and another population is added to the population pool with doubled population size.

Algorithm 1 Algorithm Framework

```
1: function RUN($problem, seed, maxEvaluations, timeLimit$)
2:     Initialize given parameters
3:     $maximumNumberOfPopulations \leftarrow 25$
4:     while $notterminated$ do
5:         if $numberOfPopulations < maximumNumberOfPopulations$ then
6:             InitializeNewPopulation()
7:         GenerationalStepAllPopulations()
8:         $numberOfPopulations++$
9: function INITIALIZENEWPOPULATION
10:    if $numberOfPopulations = 0$ then
11:        create first population of size $1$
12:    else
13:        create new population of size $2 * populations[numberOfPopulations - 1]$
14:    $numberOfPopulations++$
15: function GENERATIONALSTEPALLPOPULATIONS
16:    for all nonterminated populations do
17:        GenerationalStepRecursion($smallestIndex, largestIndex$)
18: function GENERATIONALSTEPRECURSION($smallestIndex, largestIndex$)
19:    for $i = 0$ to $3$ do
20:        for $index = smallestIndex$ to $largestIndex$ do
21:            if $notterminated[index]$ then
22:                $terminated[index] \leftarrow$ CheckTerminationPopulation(index)
23:            if $notterminated[index] and index \geq minimumPopulationIndex$ then
24:                MakeOffspring(index)
25:                SelectSurvivors(index)
26:                PerformLocalSearch(index)
27: function CHECKTERMINATIONPOPULATION($index$)
28:    for $i = 0$ to $numberOfPopulations$ do
29:        if avg $population[i]$ fitter is better than avg $population[index]$ fitness then
30:            $minimumPopulationIndex \leftarrow index + 1$ return $true$
31:    for $i = 1$ to $populationSizes[index]$ do
32:        for $j = 0$ to $numberOfDatapoints$ do
33:            if $populations[index][i][j] \neq populations[index][0][j]$ then return $false$
    return $true$
34: function SELECTSURVIVORS($index$)
35:    for $i = 0$ to $populationSizes[index]$ do
36:        $populations[index][i] \leftarrow offsprings[index][i]$
```

The *SelectSurvivors* function copies the offspring to the population directly, so the generated offspring becomes the new population.

The *PerformLocalSearch* function has two different versions that are used. The first version of the algorithm, seen in Algorithm 2, looks at all elements in each solution once and places the element in the best cluster. The orders in which the elements and the clusters are examined are randomized as to avoid any bias. The second version, seen in Algorithm 3, works in the same way but continues to improve the solution until no more improvements are possible, so until an optimum is reached. This is a local optimum, which may or may not be the global optimum. Both LS approaches are based on first-improvement neighbourhood search instead of best-improvement neighbourhood search. In preliminary comparisons between first-improvement and best-improvement, it was found that best-improvement search takes an infeasible amount of time to complete, while first-improvement search takes a reasonable amount of time and has good results.

Algorithm 2 PerformLocalSearch once for every element

```
1: function PERFORMLOCALSEARCH($index$)
2:     for $i = 0$ to $populationSizes[index]$ do
3:         $elementOrder \leftarrow$ random ordering of size $numberOfDatapoints$
4:         for $j = 0$ to $numberOfDatapoints$ do
5:             $clusterOrder \leftarrow$ random ordering of size $numberOfClusters$
6:             for $k = 0$ to $numberOfClusters$ do
7:                 $currentCluster \leftarrow populations[index][i][elementOrder[j]]$
8:                 if $currentCluster \neq clusterOrder[k]$ then
9:                     $populations[index][i][elementOrder[j]] \leftarrow clusterOrder[k]$
10:                    Evaluate the updated solution
11:                    if new solution fitter than old solution then
12:                        keep the change
13:                    else
14:                        revert the change
```

Algorithm 3 PerformLocalSearch to optimum

```
function PERFORMLOCALSEARCH($index$)
    for $i = 0$ to $populationSizes[index]$ do
        $improved \leftarrow true$
        while $improved$ do
            $improved \leftarrow false$
            $elementOrder \leftarrow$ random ordering of size $numberOfDatapoints$
            for $j = 0$ to $numberOfDatapoints$ do
                $clusterOrder \leftarrow$ random ordering of size $numberOfClusters$
                for $k = 0$ to $numberOfClusters$ do
                    $currentCluster \leftarrow populations[index][i][elementOrder[j]]$
                    if $currentCluster \neq clusterOrder[k]$ then
                        $populations[index][i][elementOrder[j]] \leftarrow clusterOrder[k]$
                        Evaluate the updated solution
                        if new solution fitter than old solution then
                            keep the change
                            $improved \leftarrow true$
                        else
                            revert the change
```

The *MakeOffspring* function is the defining function for the different algorithms based on this framework. It decides how the algorithm generate new solutions for the population.

3.1.3 Evaluation

As a distance metric for this study we will take the sum of squared errors (**SSE**), where the distance is the Euclidean distance. The Euclidean distance is a specific version of the Minkowsky distance, explained in Section 1.5. Both the SSE approach and the Euclidean distance are very straightforward and popular in the evaluation of clustering (Agustın-Blas et al., 2012). The SSE represents cohesion of clusters. The SSE is used as an unsupervised measure, so it evaluates the quality of a solution without having information about the perfect solution. It is calculated as follows:

$$SSE(U) = \sum_{i=1}^{k} \sum_{x=C_i} d^2(x, \mu_i)$$

where U stands for a solution, k stand for the number of clusters and $d(x, \mu_i)$ represents the distance from element x to the cluster centre of cluster C_i, denoted by μ_i.

Fitness of a solution is represented in the form of an objective value and a constraint value. The objective value pertains to the fitness is the SSE for that solution. The constraint value pertains to how well the number of clusters in the solution corresponds to the desired number of clusters in the algorithm. If the constraint value is equal to zero, the solution is feasible given the required number of clusters. The fitness value and the objective value of a solution are used interchangeably as the constraint value is supposed to be equal to zero for solutions encountered when an algorithm terminates. The complete fitness can thus be represented by a tuple:

$$F(U) = \langle SSE(U), |k_{req} - k_U| \rangle$$

where k_{req} stands for the number of clusters required by the input parameter and k_U represents the number of clusters in solution U.

Algorithm 4 Solution evaluation

```
1: $objective \leftarrow 0$
2: $constraint \leftarrow 0$
3: $clustercenters \leftarrow newdouble[k][d]$
4: $clustercenters \leftarrow initializeeverycenterwith0$
5: for all cluster $c$ in clusters do
6:     for all element $e$ in solution do
7:         if $e.cluster = c$ then
8:             $clustercenters[c] \leftarrow$ add all $d$ attributes
9:     $clustercenters \leftarrow$ divide by the number of elements in $c$
10: for all cluster $c$ in clusters do
11:     for all element $e$ in solution do
12:         if $e.cluster = c$ then
13:             $objective \mathrel{+}= d^2(e, clustercenter[c])$
14: if there is an empty cluster then
15:     $constraint \leftarrow value > 0$
```

Additionally, we use the Rand Index (RI) to calculate the similarity of a found solution (U) and the known optimal solution (O) (Rand, 1971). As such, it measures how correct the solution is compared to the perfect solution. The RI is calculated as follows:

$$RI(U) = \frac{TP + TN}{TP + FP + TN + FN}$$

where U represents the solution that is measured. O represents the known optimal solution. TP (true positives) describes the number of elements U and O agree on being in the same cluster. TN (true negatives) describes the number of elements U and O agree on being in different clusters. FP (false positives) and FN (false negatives) refer to the numbers of elements U and O disagree on being in the same and in different clusters, respectively. RI is in the interval $[0, 1]$, where values closer to 0 indicate worse solution quality and values closer to 1 indicate better solution quality.

3.1.4 Termination

Termination of the algorithm happens based on the elapsed time or the number of evaluations. After every generation, these values are checked and the algorithm determines whether it should terminate. These values are passed to the algorithm as input parameters.

3.2 Local Search Algorithms

In order to test whether the GPX algorithm and the LTGA benefit from their specific ways of dealing with information about clusters and generating solutions based on that information we need to compare these algorithms to a simple algorithm starting from random solutions. So we can test whether these approaches are better than a multi-start local search (MLS) approach, we implement MLS to keep the difference only in terms of how new solutions are generated when comparing results.

For this, the framework algorithm is adjusted in such a way that *SelectSurvivors* is omitted. The *MakeOffspring* function generates new random solutions and the *PerformLocalSearch* function improves each newly generated solution until an optimum is found. This therefore effectively becomes a multi-start local search algorithm. In addition, populations are not terminated as this is not necessary due to there being no recombination operator in this algorithm.

3.3 Greedy Partitioning Crossover Algorithms

Two variants of the GPX approach are implemented. The GPX algorithm uses a LS operator that improves the solution through one iteration over all elements. Thus, all elements are moved to a different cluster at most once. This is shown in Algorithm 2. The optGPX algorithm uses a LS operator that keeps improving the solution until a(n) (local) optimum is found. This is shown in Algorithm 3. Thus, the LS operator is the only difference between the GPX and optGPX algorithms.

Both algorithms use a simple tournament selection of size 2 to select the solution parent pairs to use in the crossover. The same solution can be selected twice in a pair, resulting in the offspring being equal to either parent. As the population size increases, selecting the same solution from the population as both parents becomes less likely.

Algorithm 5 shows how both versions of the GPX approach handle selection and crossover. The greedy partitioning crossover operator (Galinier and Hao, 1999), performs the crossover of solutions based on the cardinality of the groups within them. We argue that the greedy partitioning crossover approach as proposed by (Galinier and Hao, 1999) also works well for clustering, as the graph colouring problem can be viewed as a partitioning problem. In our approach, we perform crossover based on cluster fitness of the clusters within a solution instead. This is more in line with our evaluation function, how fit a part of the solution is, than cardinality would be; clusters containing more elements are not necessarily better clusters than smaller ones.

Algorithm 5 GPX and optGPX MakeOffspring function

```
1: function MAKEOFFSPRING($index$)
2:     $populationSize \leftarrow populationSizes[index]$
3:     $indices \leftarrow$ new integer array of size $populationSize * 2$
4:     $perm1 \leftarrow$ random permutation of solution indices
5:     $perm2 \leftarrow$ random permutation of solution indices
6:     for $i = 0$ to $populationSize$ do
7:         if $populations[index][perm1[i]]$ fitter than $populations[index][perm2[i]]$ then
8:             $indices[i] \leftarrow perm1[i]$
9:         else
10:            $indices[i] \leftarrow perm2[i]$
11:    $perm1 \leftarrow$ random permutation of solution indices
12:    $perm2 \leftarrow$ random permutation of solution indices
13:    for $i = 0$ to $populationSize$ do
14:        if $populations[index][perm1[i]]$ fitter than $populations[index][perm2[i]]$ then
15:            $indices[i + populationSize] \leftarrow perm1[i]$
16:        else
17:            $indices[i + populationSize] \leftarrow perm2[i]$
18:    for $i = 0$ to $populationSize$ do
19:        $parentA \leftarrow populations[index][indices[i]]$
20:        $parentB \leftarrow populations[index][indices[i + populationSize]]$
21:        $child \leftarrow$ initialize empty solution
22:        Sort clusters of $parentA$ and $parentB$ based on their fitness
23:        while there are unassigned elements in $child$ do
24:            $cluster \leftarrow$ fittest remaining cluster from $parentA$ and $parentB$
25:            for all element $e$ in $cluster$ do
26:                if $e$ unassigned in $child$ then
27:                    Assign $e$ to $cluster$ in $child$
28:        $offsprings[index][i] \leftarrow child$
29:        Evaluate $offsprings[index][i]$
```

The algorithm considers both parents and uses the fittest cluster from either parent. This cluster is then inserted into the offspring. The elements in this cluster are then further ignored in both parent solutions. This is repeated until all clusters from the parent solutions are assigned to the offspring.

The evaluation of the fitness of a single cluster is simply a run of SSE for only one cluster:

$$SSE(C) = \sum_{x \in C} d^2(x, \mu_C)$$

where C stands for a cluster, the cluster centre of C is represented as μ_C and $d(x, \mu_C)$ represents the distance from element x to μ_C.

3.4 Linkage Tree Genetic Algorithms

Two variants of the LTGA approach are implemented. The LTGA algorithm uses a LS operator that improves the solution through one iteration over all elements. Thus, all elements are moved to a different cluster at most once. This is shown in Algorithm 2. The optLTGA algorithm uses a LS operator that keeps improving the solution until a(n) (local) optimum is found. This is shown in Algorithm 3. Thus, the LS operator is the only difference between the LTGA and optLTGA algorithms.

Algorithm 6 shows both versions of the LTGA approach. LTGA can be described as a gene-pool optimal mixing evolutionary algorithm (GOMEA) with its family of subsets (FOS) model being a linkage tree (LT).

A FOS is a dependency structure between the problem variables. It is a set of subsets of the set of all problem variables.

A LT is an hierarchical structure. It forms an hierarchical clustering of the problem variables (here the data points in the clustering instance). The root node is the group of all variables and each node at a lower level is the subset of the parent node split into two mutually exclusive subsets. The LT has n leaf nodes and $n - 1$ internal nodes.

Building the LT is done bottom-up, starting with a univariate structure and clustering hierarchically. Variables in a subset are considered to be dependent, but independent in different child subsets. The similarity measure between two variables is the mutual information, based on joint entropy. The similarity measure between two groups is the average pairwise linkage clustering. This is the unweighted pair group method with arithmetic mean (UPGMA). This is done in the **LearnLT** function and this is done each generation.

GOMEA (and thus LTGA) selects a new donor for each FOS subset. New solutions are generated by traversing the tree, starting at the top. Nodes in the LT are used as crossover masks. A random donor solution gets selected and its values at the crossover mask replace the variable values in the selected solution. Then, the new solution is evaluated and if it is better or equal, is accepted. If it is worse, the change is rejected. More specifics about the LTGA algorithm can be found, e.g. in (Thierens, 2010; Thierens and Bosman, 2011; Bosman and Thierens, 2012).

Algorithm 6 LTGA and optLTGA MakeOffspring function

```
1: function MakeOffspring($index$)
2:     LearnLT($index$)
3:     GenerateNewSolutions($index$)
4: function LearnLT($index$)
5:     Compute mutual information matrix using joint entropy
6:     Initialize LT to the initial mutual information matrix
7:     Initialize similarity matrix
8:     Calculate the LT
9: function GenerateNewSolutions($index$)
10:    for $i = 0$ to $populationSizes[index]$ do
11:        $result \leftarrow$ GenerateNewSolution($index, i$)
12:        $offsprings[index][i] \leftarrow result$
13: function GenerateNewSolution($index, parentIndex$)
14:    $result \leftarrow populations[index][parentIndex]$
15:    $backup \leftarrow result$
16:    ShuffleFOS($index$)
17:    for $i = 0$ to $fossLength[index]$ do
18:        if $fossNumberOfIndices[index][i] = numberOfDatapoints$ then
19:            continue
20:        $donor \leftarrow$ random donor from $populations[index]$
21:        for $j = 0$ to $fossNumberOfIndices[index][i]$ do
22:            $result[foss[index][i][j]] \leftarrow donor[foss[index][i][j]]$
23:        if $result$ changed then
24:            if change is improvement then
25:                $backup \leftarrow result$
26:            else
27:                $result \leftarrow backup$
28:        if $result$ not changed or no improvement for some time then
29:            ShuffleFOS($index$)
30:            for $i = 0$ to $fossLength[index]$ do
31:                $donor \leftarrow$ elitist solution
32:                for $j = 0$ to $fossNumberOfIndices[index][i]$ do
33:                    $result[foss[index][i][j]] \leftarrow donor[foss[index][i][j]]$
34:                if $result$ changed then
35:                    if change is improvement then
36:                        $backup \leftarrow result$
37:                    else
38:                        $result \leftarrow backup$
39:            if $result$ was not changed and $elitistSolution$ fitter than $result$ then
40:                $result \leftarrow elitistSolution$
    return $result$
```

4 Experiments

In this chapter, we will give an overview of the experiments that are performed. In Section 4.1, we will define the data sets used for the experiments and describe how the artificial data sets were created. In Section 4.2, we will describe the structure of the experiments and details about the used termination criteria. Finally, the hardware used to run the experiments is stated in Section 4.3.

4.1 Data Sets

For the experiments we compare fitness values of the best solutions encountered during the runs of the algorithms. The data sets used for the experiments are well-known and often-used data sets in the field of clustering algorithms. Some data sets are given slightly modified names for clarity and brevity. The used data sets with problem instance names and references can be found in Table 4.1. In case of the *cardio* and *german* data sets, there are multiple problem instances, where the number of clusters differs. This, and descriptions for the data sets, can be found in Table 4.2.

TABLE 4.1: Data Set Naming

Data set name	Instance name	Reference
spheric	spheric	(Agustın-Blas et al., 2012)
structured	structured	(Agustın-Blas et al., 2012)
unbalanced	unbalanced	(Agustın-Blas et al., 2012)
iris	iris	(Fisher, 1936)
wine	wine	(Forina, 1991)
studentevaluation	studenteval	(Gunduz and Fokoue, 2013)
ecoli	ecoli	(Nakai and Kanehisa, 1991)
pima indians diabetes	diabetes	(Sigillito, 1990)
banknote	banknote	(Doerksen, 2012)
cardiotocography	cardio-_	(Campos et al., 2000)
german towns	german-_	(Späth, 1980)

In order to compare the results of the algorithms with results achieved in (Agustın-Blas et al., 2012), we run experiments on comparable data sets. These data sets are *spheric*, *structured*, and *unbalanced*. These data sets are not exactly the same as in (Agustın-Blas et al., 2012), but rather generated in exactly the same way. The other data sets were chosen to be generally known data sets, such as in (Siddiqi and Sait, 2017).

TABLE 4.2: Data Set Descriptions

Instance	Data points n	Attributes d	Clusters k
spheric	300	2	8
structured	400	2	3
unbalanced	200	2	9
iris	150	4	3
wine	178	13	3
studenteval	5820	32	3
diabetes	768	8	2
cardio-_	2126	21	3 or 10
banknote	1372	4	2
ecoli	336	7	8
german-_	89	3	2 to 10

All three artificial data sets (*spheric, structured, unbalanced*) represent 2-dimensional clustering problems. The *spheric* data set consists of, as is shown in Table 4.2, 300 data points. These data points are randomly generated using a Gaussian distribution from 8 equiprobable classes. These classes have the following means: $\mu_1 = (-1, 0)$, $\mu_2 = (-1, -1)$, $\mu_3 = (-1, -3)$, $\mu_4 = (3, -1)$, $\mu_5 = (-1, 1)$, $\mu_6 = (2, -2)$, $\mu_7 = (1, 2)$, and $\mu_8 = (3, 1)$. All classes have the following covariance matrix:

$$\Sigma = \begin{bmatrix} 0.35^2 & 0 \\ 0 & 0.35^2 \end{bmatrix}$$

The resulting *spheric* instance used in the experiments is shown in Figure 4.1.

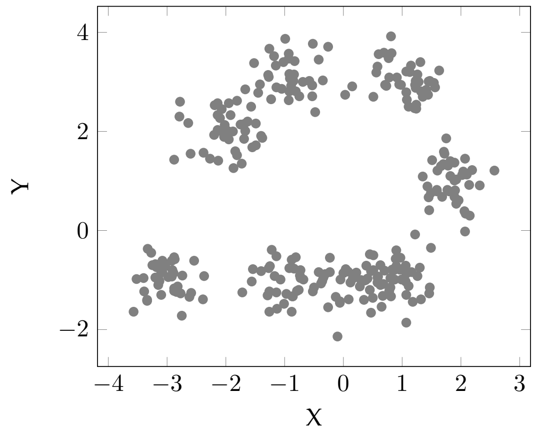

FIGURE 4.1: Spheric instance representation

The *structured* data set consists of, as is shown in Table 4.2, 400 data points. These data points are randomly generated using a Gaussian distribution from 3 classes with probabilities: $p_1 = 0.5$, $p_2 = 0.33$, and $p_3 = 0.17$. The means of the classes are: $\mu_1 = (-1, -1)$, $\mu_2 = (2, -1)$, and $\mu_3 = (0, 2)$. They have the following covariance matrices:

$$\Sigma_1 = \begin{bmatrix} 1^2 & 0 \\ 0 & 0.8^2 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 0.6^2 & 0 \\ 0 & 0.4^2 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 0.3^2 & 0 \\ 0 & 0.5^2 \end{bmatrix}$$

The resulting *structured* instance used in the experiments is shown in Figure 4.2.

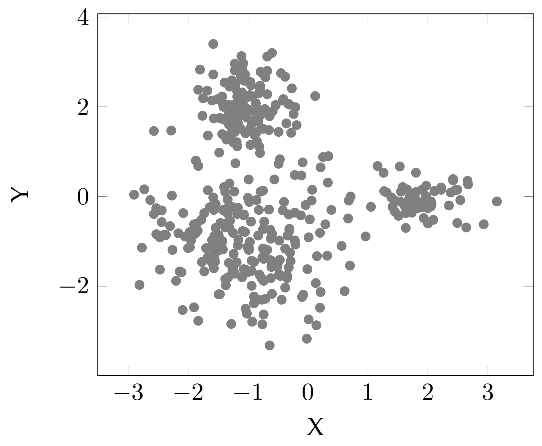

FIGURE 4.2: Structured instance representation

The *unbalanced* data set consists of, as is shown in Table 4.2, 200 data points. These data points are randomly generated using a Gaussian distribution from 9 equiprobable classes. The means of the classes are: $\mu_1 = (1, -1)$, $\mu_2 = (-1.5, 0)$, $\mu_3 = (0, 1)$, $\mu_4 = (-1, 1)$, $\mu_5 = (2, -1)$, $\mu_6 = (-2, -1)$, $\mu_7 = (-0.5, 2)$, $\mu_8 = (-1, -1)$, and $\mu_9 = (1.5, 0)$. All classes have the following covariance matrix:

$$\Sigma = \begin{bmatrix} 0.2^2 & 0 \\ 0 & 0.2^2 \end{bmatrix}$$

The resulting *unbalanced* instance used in the experiments is shown in Figure 4.3.

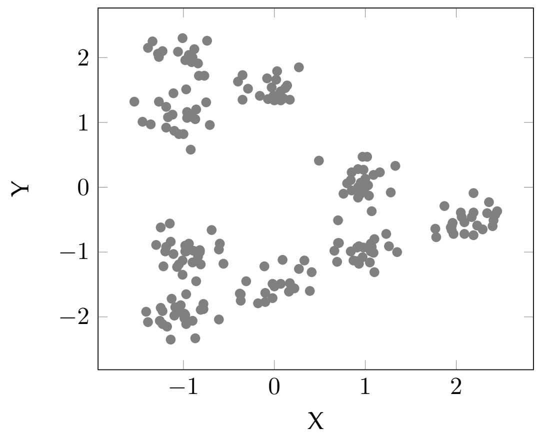

FIGURE 4.3: Unbalanced instance representation

4.2 Experiments

As we study the performance of the GPX and the LTGA algorithms, we need to conduct experiments pertaining one difference at a time. Described in Section 3.2, we have two different ways to perform the local search operator. We need to distinguish between these approaches and only compare the GPX and LTGA algorithms using the same local search approach. However, we also want to see which local search approach is more successful. This thus leaves us with the following experiments:

- Comparing GPX and optGPX (comparing the LS approaches)
- Comparing LTGA and optLTGA (comparing the LS approaches)
- Comparing GPX and LTGA (comparing the algorithms)
- Comparing optGPX and optLTGA (comparing the algorithms)
- Comparing optGPX and optLTGA to MLS (2 experiments, comparing the algorithms to a simple restarting LS approach)

The comparisons will be done based on the fitness values of the best-found solutions. A student's t-test will determine statistical significance in experiment results.

In the experiments, we run most experiments 100 times and take the means of the found elite solution fitnesses to compare. We also calculate the standard deviations. The number of runs for each problem instance can be found in Table 4.3. A balanced between time required to run the experiments and effect on statistical significance of the results needed to be found. We found that running each experiment 100 times gave us an acceptable required time to run all experiments and ensures that low sample sizes are not a problem for statistical significance. The exception is the *studenteval* instance, which takes a very long time to run. Because of the lengthy run time for *studenteval*, only 10 runs were performed for each experiment.

TABLE 4.3: Number of runs per problem instance

Instance	Runs
spheric	100
structured	100
unbalanced	100
iris	100
wine	100
studenteval	10
ecoli	100
diabetes	100
banknote	100
cardio-_	100
german-_	100

Such experiments can be performed with different termination criteria. Because choosing a different criterion can influence the results of the experiments, we compare the algorithms using both evaluation-based and time-based termination criteria. The specific values for the chosen termination criteria can be found in Table 4.4. These values were determined by running a single local search to an optimum. The run times and numbers of evaluations were averaged over 100 runs and rounded.

Furthermore, additional experiments are done with an extended time limit to investigate how the algorithms perform when less constrained. This gives the algorithms the opportunity to show increased performance because of the more extensive learning that can take place, if applicable.

TABLE 4.4: Termination Criteria

Instance	Run Time (ms)	Evaluations
spheric	54	15068
structured	34	4456
unbalanced	24	9604
iris	9	2317
wine	21	1593
studenteval	119206	128734
ecoli	171	23416
diabetes	348	7187
banknote	508	10707
cardio-3	12039	52979
cardio-10	44741	535774
german-2	1	465
german-3	2	1116
german-4	6	2668
german-5	9	4724
german-6	7	3962
german-7	8	4784
german-8	9	5612
german-9	15	10012
german-10	15	10294

The framework used in all algorithms checks the termination criterion each generation. The termination criteria shown in Table 4.4 are, in many runs, higher than the values at the generation which has the value that is actually closest to the value of the termination criterion. As such, the run time or number of evaluations used by the algorithms is usually 2 to 3 times that of the relevant termination criterion value. This thus also allows the algorithms to reach evaluate more generations.

As an example: the termination criterion is set to 150 milliseconds. After some generations, the algorithm has used 140 milliseconds to reach this point. Because each generation contains more solutions to process, the next generation will take, e.g. 170 milliseconds. The number of used milliseconds used by the algorithm on termination then becomes $140 + 170 = 310$ milliseconds. This is more than twice the number of milliseconds set as the termination value. This happens often during the experiments.

Additionally, we will give an analysis of the Rand Index as opposed to the fitness value based on the SSE used internally by the algorithms. We get the data for this analysis through the experiments mentioned above. In this analysis, we can see how well the used evaluation function fits the problems. Because this is not indicative of how well the different approaches perform, it will not be extensive. The analysis will be based on the correlation between internally used fitness and the Rand Index.

We will also give an analysis of memory usage by the different approaches, giving some indication about the scalability of the approaches, memory-wise.

4.3 System specifications

Because some of the experiment runs are limited by time rather than number of calculations or evaluations, the hardware that was used when running the experiments becomes an important factor. In this case, the same hardware was used for all experiments, but the specifications are listed in Table 4.5 so that all relevant data is noted.

TABLE 4.5: Specifications of the hardware used for the experiments

Component	Model
CPU	Intel Core i7-4790K (4.00GHz)
GPU	NVIDIA GeForce GTX 1070
RAM	16GB DDR4

5 Results

In this chapter, we look at the results from the experiments. This chapter is here to show the main experiment results, for which the conclusions and discussion following the results are presented in Chapter 6. However, for some evaluation It must be made clear that when an optimum or optima are described, this will mean a local optimum or local optima. Such a local optimum may or may not be the global optimum, but should this be the case, this will be stated specifically. As mentioned in 3.1.3, fitness of a solution and the objective value of the fitness of a solution are used interchangeably.

This chapter is structured as follows. In Section 5.1, the results from the experiments pertaining the different local search operators are described. Comparisons between the greedy partitioning crossover algorithm, the linkage tree genetic algorithm and, the multi-start local search algorithm are made in Section 5.2. The results of the experiments where the time limit was extended are described in Section 5.3. A comparison is made between the Rand Index and the internally used fitness value in Section 5.4. In Section 5.5, the memory usage of the different approaches is discussed. Finally, a summary of the comparisons between the algorithms is presented in Section 5.6.

5.1 Local Search Variants

In this section, we will look at how the different local search approaches, discussed in Section 3.1 (Algorithm 2 and Algorithm 3), perform in GPX and LTGA. We expect the local search to optimality to outperform the local search that looks at every element once per generation. This would be at the cost of the number of generations that can be generated. We will first look at how the local search operators converge to an optimum in Section 5.1.1. After this, we will see how the local search operators relate in the next sections.

5.1.1 Optimum Convergence

We look at how the local search algorithms converge to an optimum. In Figure 5.1 we see how the fitness of the best solutions in each iteration of the local search operator converges to the fitness of the optimum that is found. The fitness values, shown on the y-axes of the figures, are explained in 3.1.3. The x-axes represent the number of iterations the local search performs over all the elements in the solution(s). It is important to note that Algorithm 2 performs only one iteration over all elements, so the improvement of the fitness stops at iteration 1. In contrast, Algorithm 3 performs all iterations shown in the figures. Because the local search operator is not deterministic, these figures are only an indication for how the fitness value can converge in different runs of the local search value. The figures were generated starting with a random solution, so the first fitness value (at iteration 0), is worse than the fitness value would be for a starting solution in later generations in either of the GA algorithms studied.

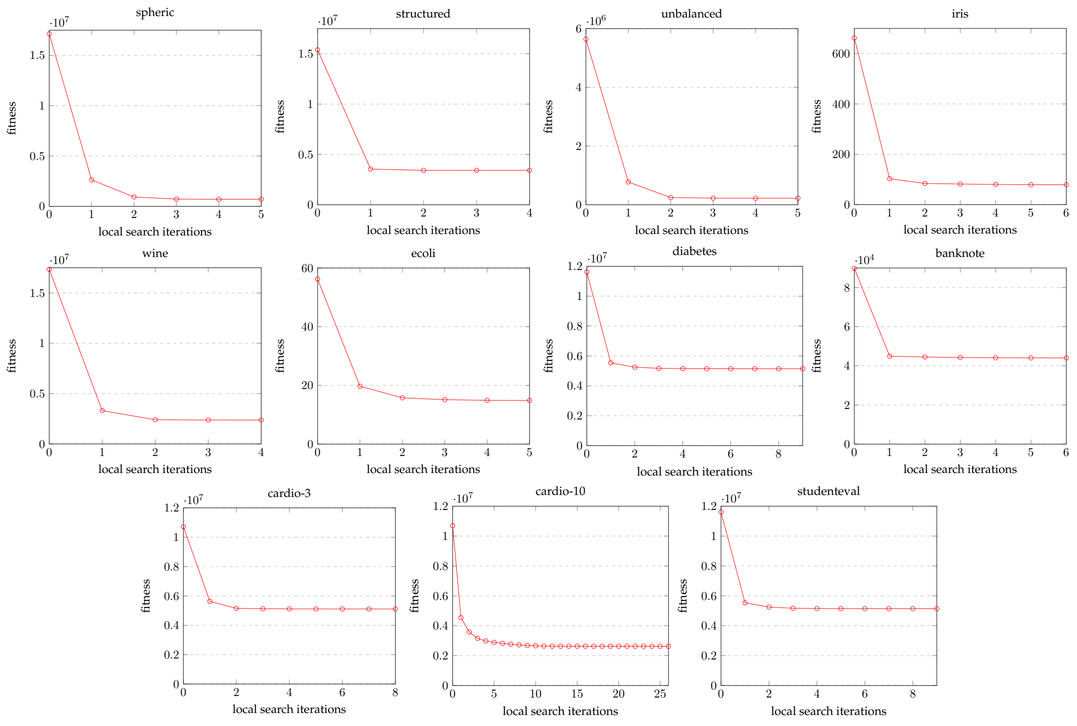

FIGURE 5.1: Fitness plots for all non-*german-_* instances

We can see that the first few iterations yield the highest improvement in fitness value. Most improvements start being only marginal from iteration 2 on, except for the *cardio-10* instance, where substantial gains are also made thereafter. This can be explained by the fact that the *cardio-10* instance requires many more iterations to get to an optimum. These figures give us a clear idea of what happens when applying either of the local search operator(s).

5.1.2 GPX with equal evaluation limit

In this section, we compare the GPX and the optGPX algorithms. The termination criterion for the experiments run here is based on the maximum number of evaluations. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.1.

TABLE 5.1: Fitness values from SSE and p-value from t-test comparing GPX and optGPX with equal evaluation limit

Instance	GPX $\bar{x}$	GPX s	optGPX $\bar{x}$	optGPX s	p
spheric	1.86E+06	2.06E+05	**6.85E+05**	5.73E+04	<0.001
structured	3.54E+06	6.76E+04	**3.42E+06**	3.28E-09	<0.001
unbalanced	6.35E+05	3.61E+04	**2.07E+05**	5.25E+04	<0.001
iris	9.25E+01	2.46E+00	**7.89E+01**	1.00E-13	<0.001
wine	3.08E+06	1.15E+05	**2.37E+06**	5.62E-09	<0.001
studenteval	1.93E+05	1.15E+03	**1.77E+05**	3.07E-11	<0.001
ecoli	1.76E+01	5.69E-01	**1.42E+01**	3.59E-01	<0.001
diabetes	5.50E+06	1.72E+04	**5.14E+06**	8.42E-09	<0.001
banknote	4.49E+04	4.00E+01	**4.40E+04**	2.19E-11	<0.001
cardio-3	5.38E+06	4.79E+04	**5.10E+06**	2.53E+04	<0.001
cardio-10	4.07E+06	1.33E+05	**2.63E+06**	3.61E+04	<0.001
german-2	1.54E+12	3.28E+10	**6.03E+11**	7.36E-04	<0.001
german-3	1.22E+12	7.93E+10	**3.64E+11**	4.91E-04	<0.001
german-4	9.40E+11	1.54E+11	**1.04E+11**	1.53E-05	<0.001
german-5	7.52E+11	1.84E+11	**5.98E+10**	9.20E-05	<0.001
german-6	5.94E+11	1.74E+11	**4.58E+10**	6.13E-05	<0.001
german-7	4.30E+11	1.36E+11	**3.72E+10**	6.90E-05	<0.001
german-8	3.68E+11	1.20E+11	**3.47E+10**	1.20E+09	<0.001
german-9	3.07E+11	8.53E+10	**3.03E+10**	3.77E+08	<0.001
german-10	2.67E+11	7.93E+10	**2.91E+10**	4.19E+07	<0.001

As becomes clear from the results, optGPX dominates GPX for every instance.

5.1.3 LTGA with equal evaluation limit

In this section, we compare the LTGA and the optLTGA algorithms. The termination criterion for the experiments run here is based on the maximum number of evaluations. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.2.

TABLE 5.2: Fitness values from SSE and p-value from t-test comparing LTGA and optLTGA with equal evaluation limit

Instance	LTGA $\bar{x}$	LTGA s	optLTGA $\bar{x}$	optLTGA s	p
spheric	8.82E+05	2.32E+05	**7.57E+05**	1.61E+05	<0.001
structured	3.53E+06	4.73E+05	3.50E+06	5.97E+05	0.69
unbalanced	2.93E+05	6.20E+04	**2.39E+05**	6.10E+04	<0.001
iris	7.90E+01	1.32E-01	**7.89E+01**	9.74E-14	<0.001
wine	2.40E+06	3.96E+04	**2.37E+06**	5.62E-09	<0.001
studenteval	1.88E+05	5.89E+03	**1.77E+05**	5.13E-10	<0.001
ecoli	1.46E+01	5.05E-01	**1.41E+01**	4.48E-01	<0.001
diabetes	5.24E+06	2.48E+03	**5.14E+06**	1.03E-08	<0.001
banknote	4.45E+04	6.87E+01	**4.40E+04**	5.85E-11	<0.001
cardio-3	5.13E+06	1.53E+04	**5.11E+06**	8.17E+03	<0.001
cardio-10	2.78E+06	4.19E+04	**2.64E+06**	3.83E+04	<0.001
german-2	9.58E+11	1.33E+11	**6.03E+11**	7.36E-04	<0.001
german-3	3.75E+11	1.45E+09	**3.64E+11**	4.91E-04	<0.001
german-4	3.01E+11	2.66E+10	**1.04E+11**	1.53E-05	<0.001
german-5	2.30E+11	3.98E+10	**5.98E+10**	9.20E-05	<0.001
german-6	1.65E+11	5.05E+10	**4.58E+10**	6.13E-05	<0.001
german-7	1.29E+11	6.17E+10	**3.72E+10**	6.90E-05	<0.001
german-8	8.75E+10	5.99E+10	**3.48E+10**	1.07E+09	<0.001
german-9	6.40E+10	5.22E+10	**3.04E+10**	7.31E+08	<0.001
german-10	5.29E+10	4.29E+10	**2.91E+10**	3.94E+07	<0.001

As becomes clear from the results, optLTGA dominates LTGA for every instance but the *structured* instance.

5.1.4 GPX with equal time limit

In this section, we compare the GPX and the optGPX algorithms. The termination criterion for the experiments run here is based on the maximum number of milliseconds. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.3.

TABLE 5.3: Fitness values from SSE and p-value from t-test comparing GPX and optGPX with equal time limit

Instance	GPX $\bar{x}$	GPX s	optGPX $\bar{x}$	optGPX s	p
spheric	2.05E+06	2.47E+05	**6.90E+05**	7.00E+04	<0.001
structured	3.56E+06	7.95E+04	**3.42E+06**	3.28E-09	<0.001
unbalanced	6.35E+05	4.45E+04	**2.04E+05**	4.76E+04	<0.001
iris	9.42E+01	3.33E+00	**7.89E+01**	1.00E-13	<0.001
wine	3.08E+06	1.36E+05	**2.37E+06**	5.62E-09	<0.001
studenteval	1.94E+05	1.04E+03	**1.78E+05**	2.49E+03	<0.001
ecoli	1.81E+01	6.20E-01	**1.42E+01**	3.71E-01	<0.001
diabetes	5.50E+06	1.66E+04	**5.14E+06**	8.42E-09	<0.001
banknote	4.49E+04	4.40E+01	**4.40E+04**	2.19E-11	<0.001
cardio-3	5.38E+06	4.01E+04	**5.09E+06**	2.67E+04	<0.001
cardio-10	4.14E+06	1.40E+05	**2.63E+06**	3.74E+04	<0.001
german-2	1.54E+12	7.11E+10	**6.03E+11**	7.36E-04	<0.001
german-3	1.22E+12	9.33E+10	**3.64E+11**	4.91E-04	<0.001
german-4	9.55E+11	1.39E+11	**1.04E+11**	1.53E-05	<0.001
german-5	7.26E+11	1.87E+11	**5.98E+10**	9.20E-05	<0.001
german-6	7.79E+11	1.59E+11	**4.58E+10**	6.13E-05	<0.001
german-7	5.96E+11	1.77E+11	**3.72E+10**	6.90E-05	<0.001
german-8	4.60E+11	1.62E+11	**3.46E+10**	1.22E+09	<0.001
german-9	3.11E+11	7.13E+10	**3.03E+10**	3.77E+08	<0.001
german-10	2.77E+11	9.47E+10	**2.91E+10**	4.17E+07	<0.001

As becomes clear from the results, optGPX dominates GPX for every instance.

5.1.5 LTGA with equal time limit

In this section, we compare the LTGA and the optLTGA algorithms. The termination criterion for the experiments run here is based on the maximum number of milliseconds. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.2.

TABLE 5.4: Fitness values from SSE and p-value from t-test comparing LTGA and optLTGA with equal time limit

Instance	LTGA $\bar{x}$	LTGA s	optLTGA $\bar{x}$	optLTGA s	p
spheric	2.63E+06	5.07E+05	**7.37E+05**	1.35E+05	<0.001
structured	4.63E+06	1.50E+06	**3.55E+06**	7.28E+05	<0.001
unbalanced	6.95E+05	6.51E+04	**2.40E+05**	6.76E+04	<0.001
iris	1.01E+02	7.93E+00	**7.89E+01**	9.05E-14	<0.001
wine	3.23E+06	2.39E+05	**2.37E+06**	5.62E-09	<0.001
studenteval	1.99E+05	2.38E+03	**1.77E+05**	4.78E-10	<0.001
ecoli	1.78E+01	8.92E-01	**1.41E+01**	3.58E-01	<0.001
diabetes	5.55E+06	1.98E+04	**5.14E+06**	1.03E-08	<0.001
banknote	4.50E+04	7.32E+02	**4.40E+04**	5.85E-11	<0.001
cardio-3	5.51E+06	7.56E+04	**5.11E+06**	8.17E+03	<0.001
cardio-10	4.04E+06	1.26E+05	**2.64E+06**	3.97E+04	<0.001
german-2	1.59E+12	8.14E+09	**6.18E+11**	1.57E+11	<0.001
german-3	1.38E+12	1.42E+11	**3.64E+11**	4.91E-04	<0.001
german-4	1.07E+12	2.83E+11	**1.04E+11**	1.53E-05	<0.001
german-5	8.27E+11	3.09E+11	**5.98E+10**	9.20E-05	<0.001
german-6	6.00E+11	3.08E+11	**4.58E+10**	7.69E+07	<0.001
german-7	4.65E+11	2.08E+11	**3.72E+10**	6.90E-05	<0.001
german-8	3.86E+11	1.19E+11	**3.48E+10**	1.04E+09	<0.001
german-9	3.62E+11	1.28E+11	**3.05E+10**	7.33E+08	<0.001
german-10	3.25E+11	8.99E+10	**2.91E+10**	6.84E+07	<0.001

As becomes clear from the results, optLTGA dominates LTGA for every instance.

5.2 Algorithms

In this section, we will look at how the different algorithm approaches, discussed in Section 3.2, Section 3.3, and Section 3.4 perform. We expect the LTGA and optLTGA to learn structure and use this to outperform GPX and optGPX, respectively. We expect GPX and optGPX to greedily make use of the selection of the fittest clusters to outperform MLS, which makes no use of problem structure or learned models at all. The next sections describe the found results.

5.2.1 GPX and LTGA with equal evaluation limit

In this section, we compare the GPX and the LTGA algorithms and the optGPX and the optLTGA algorithms. The termination criterion for the experiments run here is based on the maximum number of evaluations. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.5 and Table 5.6.

TABLE 5.5: Fitness values from SSE and p-value from t-test comparing GPX and LTGA with equal evaluation limit

Instance	GPX $\bar{x}$	GPX s	LTGA $\bar{x}$	LTGA s	p
spheric	1.86E+06	2.06E+05	**8.82E+05**	2.32E+05	<0.001
structured	3.54E+06	6.76E+04	3.53E+06	4.73E+05	0.81
unbalanced	6.35E+05	3.61E+04	**2.93E+05**	6.20E+04	<0.001
iris	9.25E+01	2.46E+00	**7.90E+01**	1.32E-01	<0.001
wine	3.08E+06	1.15E+05	**2.40E+06**	3.96E+04	<0.001
studenteval	1.93E+05	1.15E+03	**1.88E+05**	5.89E+03	0.0203
ecoli	1.76E+01	5.69E-01	**1.46E+01**	5.05E-01	<0.001
diabetes	5.50E+06	1.72E+04	**5.24E+06**	2.48E+03	<0.001
banknote	4.49E+04	4.00E+01	**4.45E+04**	6.87E+01	<0.001
cardio-3	5.38E+06	4.79E+04	**5.13E+06**	1.53E+04	<0.001
cardio-10	4.07E+06	1.33E+05	**2.78E+06**	4.19E+04	<0.001
german-2	1.54E+12	3.28E+10	**9.58E+11**	1.33E+11	<0.001
german-3	1.22E+12	7.93E+10	**3.75E+11**	1.45E+09	<0.001
german-4	9.40E+11	1.54E+11	**3.01E+11**	2.66E+10	<0.001
german-5	7.52E+11	1.84E+11	**2.30E+11**	3.98E+10	<0.001
german-6	5.94E+11	1.74E+11	**1.65E+11**	5.05E+10	<0.001
german-7	4.30E+11	1.36E+11	**1.29E+11**	6.17E+10	<0.001
german-8	3.68E+11	1.20E+11	**8.75E+10**	5.99E+10	<0.001
german-9	3.07E+11	8.53E+10	**6.40E+10**	5.22E+10	<0.001
german-10	2.67E+11	7.93E+10	**5.29E+10**	4.29E+10	<0.001

As becomes clear from the results, LTGA dominates GPX for every instance but the *structured* instance.

TABLE 5.6: Fitness values from SSE and p-value from t-test comparing optGPX and optLTGA with equal evaluation limit

Instance	optGPX $\bar{x}$	optGPX s	optLTGA $\bar{x}$	optLTGA s	p
spheric	**6.85E+05**	5.73E+04	7.57E+05	1.61E+05	<0.001
structured	3.42E+06	3.28E-09	3.50E+06	5.97E+05	0.16
unbalanced	**2.07E+05**	5.25E+04	2.39E+05	6.10E+04	<0.001
iris	7.89E+01	1.00E-13	7.89E+01	9.74E-14	1.00
wine	2.37E+06	5.62E-09	2.37E+06	5.62E-09	1.00
studenteval	1.77E+05	3.07E-11	**1.77E+05**	5.13E-10	<0.001
ecoli	1.42E+01	3.59E-01	1.41E+01	4.48E-01	0.36
diabetes	5.14E+06	8.42E-09	**5.14E+06**	1.03E-08	<0.001
banknote	4.40E+04	2.19E-11	4.40E+04	5.85E-11	1.00
cardio-3	**5.10E+06**	2.53E+04	5.11E+06	8.17E+03	<0.001
cardio-10	2.63E+06	3.61E+04	2.64E+06	3.83E+04	0.13
german-2	6.03E+11	7.36E-04	6.03E+11	7.36E-04	1.00
german-3	3.64E+11	4.91E-04	3.64E+11	4.91E-04	1.00
german-4	1.04E+11	1.53E-05	1.04E+11	1.53E-05	1.00
german-5	5.98E+10	9.20E-05	5.98E+10	9.20E-05	1.00
german-6	4.58E+10	6.13E-05	4.58E+10	6.13E-05	1.00
german-7	3.72E+10	6.90E-05	3.72E+10	6.90E-05	1.00
german-8	3.47E+10	1.20E+09	3.48E+10	1.07E+09	0.36
german-9	3.03E+10	3.77E+08	3.04E+10	7.31E+08	0.21
german-10	**2.91E+10**	4.19E+07	2.91E+10	3.94E+07	0.00511

Between optGPX and optLTGA, there is no clearly better performing algorithm. The optGPX algorithm seems to perform slightly better with 4 instances having significantly better results than optLTGA, versus 2 instances having better results for optLTGA. However, most instances show no statistical difference between the two approaches.

5.2.2 MLS with equal evaluation limit

In this section, we compare the optGPX and the optLTGA algorithms with the MLS algorithm. The termination criterion for the experiments run here is based on the maximum number of evaluations. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.7 and Table 5.8.

TABLE 5.7: Fitness values from SSE and p-value from t-test comparing optGPX and MLS with equal evaluation limit

Instance	optGPX $\bar{x}$	optGPX s	MLS $\bar{x}$	MLS s	p
spheric	6.85E+05	5.73E+04	**6.73E+05**	4.39E+04	0.04
structured	3.42E+06	3.28E-09	3.42E+06	3.28E-09	1.00
unbalanced	2.07E+05	5.25E+04	**1.90E+05**	4.14E+04	0.00598
iris	7.89E+01	1.00E-13	7.89E+01	1.00E-13	1.00
wine	2.37E+06	5.62E-09	2.37E+06	5.62E-09	1.00
studenteval	1.77E+05	3.07E-11	1.77E+05	3.07E-11	1.00
ecoli	1.42E+01	3.59E-01	**1.40E+01**	2.96E-01	<0.001
diabetes	5.14E+06	8.42E-09	5.14E+06	8.42E-09	1.00
banknote	4.40E+04	2.19E-11	4.40E+04	2.19E-11	1.00
cardio-3	5.10E+06	2.53E+04	5.09E+06	2.65E+04	0.53
cardio-10	2.63E+06	3.61E+04	**2.62E+06**	3.37E+04	0.0165
german-2	6.03E+11	7.36E-04	6.03E+11	7.36E-04	1.00
german-3	3.64E+11	4.91E-04	3.64E+11	4.91E-04	1.00
german-4	1.04E+11	1.53E-05	1.04E+11	1.53E-05	1.00
german-5	5.98E+10	9.20E-05	5.98E+10	9.20E-05	1.00
german-6	4.58E+10	6.13E-05	4.58E+10	6.13E-05	1.00
german-7	3.72E+10	6.90E-05	3.72E+10	6.90E-05	1.00
german-8	3.47E+10	1.20E+09	**3.42E+10**	1.42E+09	0.0124
german-9	3.03E+10	3.77E+08	3.03E+10	3.71E+08	0.90
german-10	2.91E+10	4.19E+07	**2.91E+10**	3.95E+07	0.0149

We can see that the MLS algorithm performs significantly better in 6 of the instances. For most instances however, there is no statistical difference between optGPX and MLS. The optGPX algorithm never performs significantly better than the MLS algorithm.

TABLE 5.8: Fitness values from SSE and p-value from t-test comparing optLTGA and MLS with equal evaluation limit

Instance	optLTGA $\bar{x}$	optLTGA s	MLS $\bar{x}$	MLS s	p
spheric	7.57E+05	1.61E+05	**6.73E+05**	4.39E+04	<0.001
structured	3.50E+06	5.97E+05	3.42E+06	3.28E-09	0.16
unbalanced	2.39E+05	6.10E+04	**1.90E+05**	4.14E+04	<0.001
iris	7.89E+01	9.74E-14	7.89E+01	1.00E-13	1.00
wine	2.37E+06	5.62E-09	2.37E+06	5.62E-09	1.00
studenteval	**1.77E+05**	5.13E-10	1.77E+05	3.07E-11	<0.001
ecoli	1.41E+01	4.48E-01	**1.40E+01**	2.96E-01	0.0415
diabetes	**5.14E+06**	1.03E-08	5.14E+06	8.42E-09	<0.001
banknote	4.40E+04	5.85E-11	4.40E+04	2.19E-11	1.00
cardio-3	5.11E+06	8.17E+03	**5.09E+06**	2.65E+04	<0.001
cardio-10	2.64E+06	3.83E+04	**2.62E+06**	3.37E+04	<0.001
german-2	6.03E+11	7.36E-04	6.03E+11	7.36E-04	1.00
german-3	3.64E+11	4.91E-04	3.64E+11	4.91E-04	1.00
german-4	1.04E+11	1.53E-05	1.04E+11	1.53E-05	1.00
german-5	5.98E+10	9.20E-05	5.98E+10	9.20E-05	1.00
german-6	4.58E+10	6.13E-05	4.58E+10	6.13E-05	1.00
german-7	3.72E+10	6.90E-05	3.72E+10	6.90E-05	1.00
german-8	3.48E+10	1.07E+09	**3.42E+10**	1.42E+09	<0.001
german-9	3.04E+10	7.31E+08	3.03E+10	3.71E+08	0.18
german-10	2.91E+10	3.94E+07	**2.91E+10**	3.95E+07	<0.001

We can see that the MLS algorithm performs significantly better in 7 of the instances. The optLTGA algorithm performs significantly better than the MLS algorithm in 2 instances. For about half of the instances however, there is no statistical difference between the algorithms.

5.2.3 GPX and LTGA with equal time limit

In this section, we compare the GPX and the LTGA algorithms and the optGPX and the optLTGA algorithms. The termination criterion for the experiments run here is based on the maximum number of milliseconds. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.9 and Table 5.10.

TABLE 5.9: Fitness values from SSE and p-value from t-test comparing GPX and LTGA with equal time limit

Instance	GPX $\bar{x}$	GPX s	LTGA $\bar{x}$	LTGA s	p
spheric	**2.05E+06**	2.47E+05	2.63E+06	5.07E+05	<0.001
structured	**3.56E+06**	7.95E+04	4.63E+06	1.50E+06	<0.001
unbalanced	**6.35E+05**	4.45E+04	6.95E+05	6.51E+04	<0.001
iris	**9.42E+01**	3.33E+00	1.01E+02	7.93E+00	<0.001
wine	**3.08E+06**	1.36E+05	3.23E+06	2.39E+05	<0.001
studenteval	**1.94E+05**	1.04E+03	1.99E+05	2.38E+03	<0.001
ecoli	1.81E+01	6.20E-01	**1.78E+01**	8.92E-01	0.00233
diabetes	**5.50E+06**	1.66E+04	5.55E+06	1.98E+04	<0.001
banknote	4.49E+04	4.40E+01	4.50E+04	7.32E+02	0.32
cardio-3	**5.38E+06**	4.01E+04	5.51E+06	7.56E+04	<0.001
cardio-10	4.14E+06	1.40E+05	**4.04E+06**	1.26E+05	<0.001
german-2	**1.54E+12**	7.11E+10	1.59E+12	8.14E+09	<0.001
german-3	**1.22E+12**	9.33E+10	1.38E+12	1.42E+11	<0.001
german-4	**9.55E+11**	1.39E+11	1.07E+12	2.83E+11	<0.001
german-5	**7.26E+11**	1.87E+11	8.27E+11	3.09E+11	0.00291
german-6	7.79E+11	1.59E+11	**6.00E+11**	3.08E+11	<0.001
german-7	5.96E+11	1.77E+11	**4.65E+11**	2.08E+11	<0.001
german-8	4.60E+11	1.62E+11	**3.86E+11**	1.19E+11	<0.001
german-9	**3.11E+11**	7.13E+10	3.62E+11	1.28E+11	<0.001
german-10	**2.77E+11**	9.47E+10	3.25E+11	8.99E+10	<0.001

We can see that the GPX algorithm performs significantly better in 14 of the instances. The LTGA algorithm performs significantly better than the GPX algorithm in 5 instances. For one instance there is no statistical difference between the algorithms.

TABLE 5.10: Fitness values from SSE and p-value from t-test comparing optGPX and optLTGA with equal time limit

Instance	optGPX $\bar{x}$	optGPX s	optLTGA $\bar{x}$	optLTGA s	p
spheric	**6.90E+05**	7.00E+04	7.37E+05	1.35E+05	0.00116
structured	**3.42E+06**	3.28E-09	3.55E+06	7.28E+05	0.0416
unbalanced	**2.04E+05**	4.76E+04	2.40E+05	6.76E+04	<0.001
iris	7.89E+01	1.00E-13	7.89E+01	9.05E-14	1
wine	2.37E+06	5.62E-09	2.37E+06	5.62E-09	1
studenteval	1.78E+05	2.49E+03	**1.77E+05**	4.78E-10	0.00104
ecoli	1.42E+01	3.71E-01	**1.41E+01**	3.58E-01	0.0105
diabetes	5.14E+06	8.42E-09	**5.14E+06**	1.03E-08	<0.001
banknote	4.40E+04	2.19E-11	4.40E+04	5.85E-11	1
cardio-3	**5.09E+06**	2.67E+04	5.11E+06	8.17E+03	<0.001
cardio-10	**2.63E+06**	3.74E+04	2.64E+06	3.97E+04	0.0369
german-2	6.03E+11	7.36E-04	6.18E+11	1.57E+11	0.32
german-3	3.64E+11	4.91E-04	3.64E+11	4.91E-04	1
german-4	1.04E+11	1.53E-05	1.04E+11	1.53E-05	1
german-5	5.98E+10	9.20E-05	5.98E+10	9.20E-05	1
german-6	4.58E+10	6.13E-05	4.58E+10	7.69E+07	0.32
german-7	3.72E+10	6.90E-05	3.72E+10	6.90E-05	1
german-8	3.46E+10	1.22E+09	3.48E+10	1.04E+09	0.19
german-9	3.03E+10	3.77E+08	3.05E+10	7.33E+08	0.18
german-10	**2.91E+10**	4.17E+07	2.91E+10	6.84E+07	0.0312

We can see that the optGPX algorithm performs significantly better in 6 of the instances. The LTGA algorithm performs significantly better than the GPX algorithm in 3 instances. For the remaining 11 instances, there is no statistical difference between the algorithms.

5.2.4 MLS with equal time limit

In this section, we compare the optGPX and the optLTGA algorithms with the MLS algorithm. The termination criterion for the experiments run here is based on the maximum number of milliseconds. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.11 and Table 5.12.

TABLE 5.11: Fitness values from SSE and p-value from t-test comparing optGPX and MLS with equal time limit

Instance	optGPX $\bar{x}$	optGPX s	MLS $\bar{x}$	MLS s	p
spheric	6.90E+05	7.00E+04	**6.69E+05**	2.05E+04	<0.001
structured	3.42E+06	3.28E-09	3.42E+06	3.28E-09	1
unbalanced	2.04E+05	4.76E+04	**1.92E+05**	4.93E+04	0.0324
iris	7.89E+01	1.00E-13	7.89E+01	1.00E-13	1
wine	2.37E+06	5.62E-09	2.37E+06	5.62E-09	1
studenteval	1.78E+05	2.49E+03	1.77E+05	3.07E-11	0.34
ecoli	1.42E+01	3.71E-01	**1.41E+01**	3.18E-01	0.00887
diabetes	5.14E+06	8.42E-09	5.14E+06	8.42E-09	1
banknote	4.40E+04	2.19E-11	4.40E+04	2.19E-11	1
cardio-3	5.09E+06	2.67E+04	5.09E+06	2.70E+04	0.88
cardio-10	2.63E+06	3.74E+04	**2.62E+06**	3.12E+04	<0.001
german-2	6.03E+11	7.36E-04	6.03E+11	7.36E-04	1
german-3	3.64E+11	4.91E-04	3.64E+11	4.91E-04	1
german-4	1.04E+11	1.53E-05	1.04E+11	1.53E-05	1
german-5	5.98E+10	9.20E-05	5.98E+10	9.20E-05	1
german-6	4.58E+10	6.13E-05	4.58E+10	6.13E-05	1
german-7	3.72E+10	6.90E-05	3.72E+10	6.90E-05	1
german-8	3.46E+10	1.22E+09	3.44E+10	1.36E+09	0.19
german-9	3.03E+10	3.77E+08	3.03E+10	7.67E-06	0.25
german-10	2.91E+10	4.17E+07	**2.91E+10**	3.94E+07	0.00105

We can see that the MLS algorithm performs significantly better in 5 of the instances. The optGPX algorithm does not perform significantly better for any instance. For the remaining 15 instances, there is no statistical difference between the algorithms.

TABLE 5.12: Fitness values from SSE and p-value from t-test comparing optLTGA and MLS with equal time limit

Instance	optLTGA $\bar{x}$	optLTGA s	MLS $\bar{x}$	MLS s	p
spheric	7.37E+05	1.35E+05	**6.69E+05**	2.05E+04	<0.001
structured	3.55E+06	7.28E+05	**3.42E+06**	3.28E-09	0.0416
unbalanced	2.40E+05	6.76E+04	**1.92E+05**	4.93E+04	<0.001
iris	7.89E+01	9.05E-14	7.89E+01	1.00E-13	1
wine	2.37E+06	5.62E-09	2.37E+06	5.62E-09	1
studenteval	**1.77E+05**	4.78E-10	1.77E+05	3.07E-11	<0.001
ecoli	1.41E+01	3.58E-01	1.41E+01	3.18E-01	0.95
diabetes	**5.14E+06**	1.03E-08	5.14E+06	8.42E-09	<0.001
banknote	4.40E+04	5.85E-11	4.40E+04	2.19E-11	1
cardio-3	5.11E+06	8.17E+03	**5.09E+06**	2.70E+04	<0.001
cardio-10	2.64E+06	3.97E+04	**2.62E+06**	3.12E+04	<0.001
german-2	6.18E+11	1.57E+11	6.03E+11	7.36E-04	0.32
german-3	3.64E+11	4.91E-04	3.64E+11	4.91E-04	1
german-4	1.04E+11	1.53E-05	1.04E+11	1.53E-05	1
german-5	5.98E+10	9.20E-05	5.98E+10	9.20E-05	1
german-6	4.58E+10	7.69E+07	4.58E+10	6.13E-05	0.32
german-7	3.72E+10	6.90E-05	3.72E+10	6.90E-05	1
german-8	3.48E+10	1.04E+09	**3.44E+10**	1.36E+09	0.00462
german-9	3.05E+10	7.33E+08	**3.03E+10**	7.67E-06	0.0182
german-10	2.91E+10	6.84E+07	**2.91E+10**	3.94E+07	<0.001

We can see that the MLS algorithm performs significantly better in 8 of the instances. The optGPX algorithm performs significantly better for 2 instances. For the remaining 10 instances, there is no statistical difference between the algorithms.

5.3 Extended Time Limit

In this section, we look at the same algorithms but give them more time to run. The time limits are multiplied by 20 for all non-*german* instances. The *studenteval* instance is not studied here as the run time became too long to run a decent number of experiments for. The time limits are multiplied by 100 for the *german* instances. To balance the increased time limit, every instance was only ran 20 times.

We compare the optGPX, optLTGA, and the MLS algorithms. The termination criterion for the experiments run here is based on the maximum number of milliseconds. We look at the solution fitness to compare the performance. For each algorithm, the mean fitness value ($\bar{x}$) and standard deviation s is given. These values are generated by the evaluation function described in Section 3.1.3; they represent a measure of distance. The p-value resulting from a students t-test is given. This is generally calculated one-tailed, but whenever inconclusive, the test is done two-tailed to determine significant inequality instead. Significantly better results are displayed as **bold** text. The results can be found in Table 5.13, Table 5.14 and Table 5.15.

TABLE 5.13: Fitness values from SSE and p-value from t-test comparing optGPX and optLTGA with equal extended time limit

Instance	optGPX $\bar{x}$	optGPX s	optLTGA $\bar{x}$	optLTGA s	p
spheric	**6.58E+05**	1.19E-10	7.81E+05	1.83E+05	0.0035
structured	3.42E+06	1.43E-09	3.42E+06	1.43E-09	1
unbalanced	**1.48E+05**	0.00E+00	2.18E+05	5.89E+04	<0.001
iris	7.89E+01	4.37E-14	7.89E+01	4.37E-14	1
wine	2.37E+06	4.78E-10	2.37E+06	4.78E-10	1
ecoli	**1.39E+01**	1.74E-02	1.42E+01	4.69E-01	0.00504
diabetes	5.14E+06	1.91E-09	5.14E+06	1.91E-09	1
banknote	4.40E+04	0.00E+00	4.40E+04	7.46E-12	1
cardio-3	**5.05E+06**	9.56E-10	5.11E+06	1.30E+04	<0.001
cardio-10	**2.60E+06**	1.37E+04	2.63E+06	3.58E+04	<0.001
german-2	6.03E+11	1.25E-04	6.03E+11	1.25E-04	1
german-3	3.64E+11	6.26E-05	3.64E+11	6.26E-05	1
german-4	1.04E+11	3.13E-05	1.04E+11	3.13E-05	1
german-5	5.98E+10	2.35E-05	5.98E+10	2.35E-05	1
german-6	4.58E+10	1.57E-05	4.58E+10	1.57E-05	1
german-7	3.72E+10	1.57E-05	3.72E+10	1.57E-05	1
german-8	**3.23E+10**	0.00E+00	3.32E+10	1.40E+09	<0.001
german-9	3.03E+10	3.91E-06	3.03E+10	3.91E-06	1
german-10	**2.91E+10**	0.00E+00	2.91E+10	3.45E+07	<0.001

We can see that the optGPX algorithm performs significantly better in 7 of the instances. The optLTGA algorithm does not perform significantly better for any of the instances. For the remaining 12 instances, there is no statistical difference between the algorithms.

TABLE 5.14: Fitness values from SSE and p-value from t-test comparing optGPX and MLS with equal extended time limit

Instance	optGPX $\bar{x}$	optGPX s	MLS $\bar{x}$	MLS s	p
spheric	6.58E+05	1.19E-10	6.58E+05	1.19E-10	1
structured	3.42E+06	1.43E-09	3.42E+06	1.43E-09	1
unbalanced	1.48E+05	0.00E+00	1.48E+05	0.00E+00	1
iris	7.89E+01	4.37E-14	7.89E+01	4.37E-14	1
wine	2.37E+06	4.78E-10	2.37E+06	4.78E-10	1
ecoli	1.39E+01	1.74E-02	1.39E+01	1.17E-02	0.34
diabetes	5.14E+06	1.91E-09	5.14E+06	1.91E-09	1
banknote	4.40E+04	0.00E+00	4.40E+04	0.00E+00	1
cardio-3	5.05E+06	9.56E-10	5.06E+06	1.79E+04	0.16
cardio-10	2.60E+06	1.37E+04	2.59E+06	1.58E+03	0.14
german-2	6.03E+11	1.25E-04	6.03E+11	1.25E-04	1
german-3	3.64E+11	6.26E-05	3.64E+11	6.26E-05	1
german-4	1.04E+11	3.13E-05	1.04E+11	3.13E-05	1
german-5	5.98E+10	2.35E-05	5.98E+10	2.35E-05	1
german-6	4.58E+10	1.57E-05	4.58E+10	1.57E-05	1
german-7	3.72E+10	1.57E-05	3.72E+10	1.57E-05	1
german-8	3.23E+10	0.00E+00	3.23E+10	0.00E+00	1
german-9	3.03E+10	3.91E-06	3.03E+10	3.91E-06	1
german-10	2.91E+10	0.00E+00	2.91E+10	0.00E+00	1

We can see that the MLS and optGPX algorithms do not have a significant difference in performance between them.

TABLE 5.15: Fitness values from SSE and p-value from t-test comparing optLTGA and MLS with equal extended time limit

Instance	optLTGA $\bar{x}$	optLTGA s	MLS $\bar{x}$	MLS s	p
spheric	7.81E+05	1.83E+05	**6.58E+05**	1.19E-10	0.0035
structured	3.42E+06	1.43E-09	3.42E+06	1.43E-09	1
unbalanced	2.18E+05	5.89E+04	**1.48E+05**	0.00E+00	<0.001
iris	7.89E+01	4.37E-14	7.89E+01	4.37E-14	1
wine	2.37E+06	4.78E-10	2.37E+06	4.78E-10	1
ecoli	1.42E+01	4.69E-01	**1.39E+01**	1.17E-02	0.00548
diabetes	5.14E+06	1.91E-09	5.14E+06	1.91E-09	1
banknote	4.40E+04	7.46E-12	4.40E+04	0.00E+00	1
cardio-3	5.11E+06	1.30E+04	**5.06E+06**	1.79E+04	<0.001
cardio-10	2.63E+06	3.58E+04	**2.59E+06**	1.58E+03	<0.001
german-2	6.03E+11	1.25E-04	6.03E+11	1.25E-04	1
german-3	3.64E+11	6.26E-05	3.64E+11	6.26E-05	1
german-4	1.04E+11	3.13E-05	1.04E+11	3.13E-05	1
german-5	5.98E+10	2.35E-05	5.98E+10	2.35E-05	1
german-6	4.58E+10	1.57E-05	4.58E+10	1.57E-05	1
german-7	3.72E+10	1.57E-05	3.72E+10	1.57E-05	1
german-8	3.32E+10	1.40E+09	**3.23E+10**	0.00E+00	<0.001
german-9	3.03E+10	3.91E-06	3.03E+10	3.91E-06	1
german-10	2.91E+10	3.45E+07	**2.91E+10**	0.00E+00	<0.001

We can see that the MLS algorithm performs significantly better in 7 of the instances. The optLTGA algorithm does not perform significantly better for any of the instances. For the remaining 12 instances, there is no statistical difference between the algorithms.

5.4 Rand Index

In Sections 5.1, 5.2 and 5.3, we looked at the fitness value as used internally by the algorithms. While this indicates how well the different algorithms optimize the fitness value (through the internal, unsupervised measure), it does not regard how good the found solutions are compared to the clustering that was aimed for. Therefore, we now look at the Rand Index (an external, supervised measure) for every algorithm and every non-*german-_* instance. We do not look at the *german-_* instances for this because we do not have a known perfect solution for all *german-_* instances and thus could not calculate the Rand Index for these instances.

TABLE 5.16: Rand Indices for all approaches for all non-*german-_* instances for the equal evaluation limit experiments

Instance	LS	GPX	LTGA	optGPX	optLTGA	MLS
spheric	0.9608	0.8962	0.9510	0.9706	0.9647	0.9744
structured	0.9226	0.9028	0.9190	0.9313	0.9270	0.9313
unbalanced	0.9692	0.8659	0.9509	0.9793	0.9692	0.9848
iris	0.8805	0.8386	0.8810	0.8805	0.8805	0.8805
wine	0.7202	0.7312	0.7232	0.7202	0.7202	0.7202
studenteval	0.5126	0.5152	0.5126	0.5126	0.5126	0.5126
ecoli	0.8036	0.8270	0.8332	0.8057	0.8176	0.8083
diabetes	0.5513	0.5264	0.5465	0.5513	0.5513	0.5513
banknote	0.5252	0.5171	0.5182	0.5252	0.5252	0.5252
cardio-3	0.4773	0.4621	0.4721	0.6427	0.4753	0.4813
cardio-10	0.7886	0.7389	0.7939	0.7875	0.7877	0.7871

The Rand Indices are shown in Table 5.16. The values here are from the equal evaluation limit experiments, but are comparable for the other experiments. We also show the fitness values for all non-*german-_* instances for the equal evaluation limit experiments in Table 5.17.

TABLE 5.17: Fitness values from SSE for all approaches for all non-*german-_* instances for the equal evaluation limit experiments

Instance	LS	GPX	LTGA	optGPX	optLTGA	MLS
spheric	7.77E+05	1.86E+06	8.82E+05	6.85E+05	7.57E+05	6.73E+05
structured	3.59E+06	3.54E+06	3.53E+06	3.42E+06	3.50E+06	3.42E+06
unbalanced	2.43E+05	6.35E+05	2.93E+05	2.07E+05	2.39E+05	1.90E+05
iris	7.89E+01	9.25E+01	7.90E+01	7.89E+01	7.89E+01	7.89E+01
wine	2.37E+06	3.08E+06	2.40E+06	2.37E+06	2.37E+06	2.37E+06
studenteval	1.77E+05	1.93E+05	1.88E+05	1.77E+05	1.77E+05	1.77E+05
ecoli	1.46E+01	1.76E+01	1.46E+01	1.42E+01	1.41E+01	1.40E+01
diabetes	5.14E+06	5.50E+06	5.24E+06	5.14E+06	5.14E+06	5.14E+06
banknote	4.40E+04	4.49E+04	4.45E+04	4.40E+04	4.40E+04	4.40E+04
cardio-3	5.10E+06	5.38E+06	5.13E+06	5.10E+06	5.11E+06	5.09E+06
cardio-10	2.65E+06	4.07E+06	2.78E+06	2.63E+06	2.64E+06	2.62E+06

Looking at Table 5.16, we can see that for the *spheric*, *structured*, *unbalanced*, and *iris* instances, the Rand Index is near or above 0.9, which means that the found solutions are much alike to the perfect solutions. On the other hand, we also see that *studenteval*, *diabetes*, *banknote*, and *cardio-3* have a Rand Index around 0.5. This indicates that the found solutions and the perfect, known solutions only agree on cluster membership for about half of the elements.

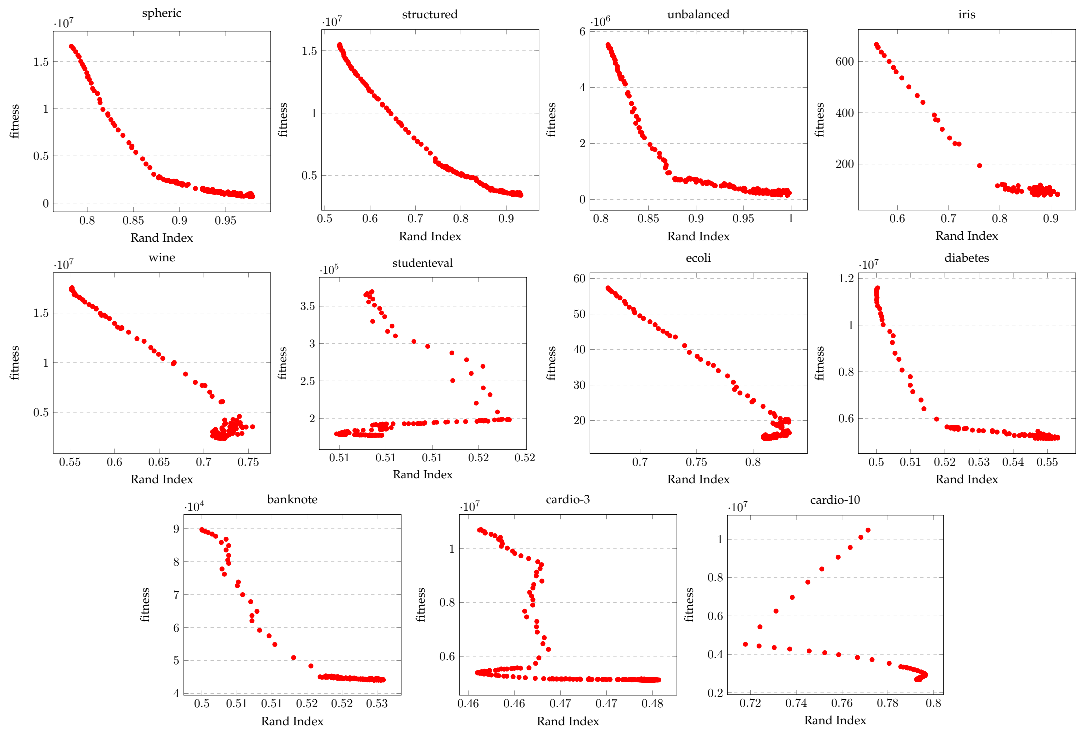

FIGURE 5.2: Correlation plots for all non-*german_* instances with 150-250 data points per plot

In the plots shown in Figure 5.2 and the correlation coefficients shown in Table 5.18, we see how fitness and Rand Index correlate for all non-*german_* instances. For about half of the instances (*spheric*, *structured*, *iris*, *wine*, *ecoli*) we can see a correlation that is roughly linear. Less so, but where one can still imagine a linear-like correlation are the *unbalanced*, *diabetes*, and *banknote* instances. Completely not resembling a linear correlation remain the *studenteval*, *cardio-3*, and *cardio-10* instances.

TABLE 5.18: Correlation coefficients (r) between fitness and Rand Index for all non-*german_* instances and the number of involved pairs of variables

Instance	r	n
spheric	-0.9075	14940
structured	-0.9725	4979
unbalanced	-0.8768	8159
iris	-0.9639	2776
wine	-0.9736	1492
studenteval	0.1508	96372
diabetes	-0.8887	7823
ecoli	-0.9569	16747
banknote	-0.9211	11499
cardio-3	-0.5988	39369
cardio-10	-0.6510	366648

The correlation coefficients between Rand Index and fitness are presented in Table 5.18. Many of the instances show a strong negative correlation. Some instances (*structured*, *iris*, *wine*, *ecoli*) even come close to a perfect negative correlation.

Finally, we compare our results with the results from the GGA from (Agustın-Blas et al., 2012). We compare our best found RI values with the best presented RI values for both GGA approaches proposed in (Agustın-Blas et al., 2012). The evaluation functions differ: in their algorithms, the Davies-Bouldin Index (DB) and the Silhouette coefficient (S) were used as internal measures. The Rand Indices are shown in Table 5.19.

TABLE 5.19: Overview of Rand Indices between GGA approaches and our presented approaches

Instance	GGA (DB)	GGA (S)	GPX	LTGA	optGPX	optLTGA	MLS
spheric	0.9814	0.9578	0.9093	0.9528	0.9792	0.9647	0.9792
structured	0.9177	0.9511	0.9159	0.9259	0.9314	0.9314	0.9314
unbalanced	1.0000	0.9936	0.8820	0.9653	0.9978	0.9742	0.9978

We can see that the Rand Indices of the solutions found through our experiments are quite close to the Rand Indices from (Agustın-Blas et al., 2012).

5.5 Memory Usage

In clustering algorithm comparison, focus lies often on quality of the produced solutions given a certain run time or number of evaluations. For completeness, we will also give some remarks about the memory usage of the studied algorithms. The memory requirements for solving small instances mostly make it so that there is no problem on modern machines. However, as instance size increases, different algorithms scale differently in their memory usage and this may cause one algorithm to become infeasible for larger problem instances, while another may still run fine.

TABLE 5.20: Memory Usage for all algorithms in MB

Instance	LS	GPX	LTGA	optGPX	optgomea	MLS
spheric	2.36	5.19	26.16	2.88	19.76	2.37
structured	2.42	8.13	52.10	4.20	45.83	2.44
unbalanced	2.29	6.32	14.77	3.55	11.64	2.30
iris	2.30	11.42	12.77	3.82	11.18	2.30
wine	2.52	6.27	12.88	3.69	11.56	2.53
studenteval	27.38	28.33	3202.11	28.37	3199.04	27.56
ecoli	2.59	4.63	24.47	2.98	16.88	2.61
diabetes	3.24	6.48	121.88	3.85	80.31	3.26
banknote	3.42	6.54	313.64	4.04	313.46	3.46
cardio-3	8.52	8.87	436.24	8.88	436.65	8.58
cardio-10	8.53	8.91	426.57	8.96	427.45	8.59
german-2	2.23	8.03	5.76	3.20	5.19	2.23
german-3	2.25	5.71	5.75	2.84	4.71	2.25
german-4	2.25	4.31	5.13	2.62	3.96	2.25
german-5	2.25	4.47	5.10	2.48	3.95	2.25
german-6	2.25	3.57	4.64	2.49	3.94	2.25
german-7	2.25	3.66	4.63	2.51	3.92	2.25
german-8	2.25	3.75	4.63	2.53	3.63	2.25
german-9	2.25	3.84	4.62	2.42	3.62	2.25
german-10	2.25	3.19	4.26	2.43	3.61	2.26

The memory requirements for the different algorithms are shown in Table 5.20. Statistical significance is not checked, as this is only an indication for the scalability memory-wise. As we can see, the memory requirements for LS and MLS are almost equal. GPX and LTGA require more memory than optGPX and optLTGA, respectively. The optGPX requires slightly more memory than both LS and MLS. LTGA and optLTGA require the most memory by far.

LS and MLS save the least information, as they keep the best encountered solution, and the investigated solution in memory. GPX and optGPX also store information about individual cluster fitnesses. LTGA and optLTGA store most information, as a LT has to be constructed and kept in memory.

We can see that LS, GPX, optGPX, and MLS scale up nicely with increased instance size. However, we also see that LTGA and optLTGA grow quickly in memory requirement. This makes LTGA and optLTGA infeasible much sooner than any of the other algorithms. For the *studenteval* instance, more than 3 GB of memory was used. On the system used (Table 4.5), this was not a problem yet, but much larger instances exist for which LTGA and optLTGA could not run on non-specialized hardware.

5.6 Experiment Summary

In this section, we will give a summary of the main experiments. The tables shown in this section summarize the outcomes of the experiments. We show the following data in the summary:

- Number of times the approach had significantly better results than the approach it was compared to, denoted as *win*
- Number of times the compared approaches had results that were not significantly different, denoted as *tie*
- Number of times the approach had significantly worse results than the approach it was compared to, denoted as *loss*

The format of the summary in the tables is as follows:

In each column (*win* / *tie* / *loss*) for the approach in the column versus the approaches in the row.

Likewise, in each row: (*loss* / *tie* / *win*) for the approach in the row versus the approaches in the column.

Example: In Table 5.21, we see in the LTGA column and the GPX row the following result: (19 / 1 / 0). This then means that the LTGA algorithm had significantly better results than the GPX algorithm for 19 instances, there was no significant difference for 1 instance, and the LTGA algorithm had significantly worse results than the GPX algorithm for 0 instances.

In the experiments where the number of evaluations was equal for the studied algorithms, we see the following:

TABLE 5.21: Summary of experiments with equal evaluations

Algorithms	GPX	LTGA	optGPX	optLTGA	MLS
GPX		(19 / 1 / 0)	(20 / 0 / 0)		
LTGA	(0 / 1 / 19)			(19 / 1 / 0)	
optGPX	(0 / 0 / 20)			(2 / 14 / 4)	(6 / 14 / 0)
optLTGA		(0 / 1 / 19)	(4 / 14 / 2)		(7 / 11 / 2)
MLS			(0 / 14 / 6)	(2 / 11 / 7)	

In the experiments where the time limit was the same for the studied algorithms, we see the following:

TABLE 5.22: Summary of experiments with equal time limits

Algorithms	GPX	LTGA	optGPX	optLTGA	MLS
GPX		(5 / 1 / 14)	(20 / 0 / 0)		
LTGA	(14 / 1 / 5)			(20 / 0 / 0)	
optGPX	(0 / 0 / 20)			(3 / 11 / 6)	(5 / 15 / 0)
optLTGA		(0 / 0 / 20)	(6 / 11 / 3)		(8 / 10 / 2)
MLS			(0 / 15 / 5)	(2 / 10 / 8)	

In the experiments where the time limit was increased, we see the following:

TABLE 5.23: Summary of experiments with extended equal time limits

Algorithms	optGPX	optLTGA	MLS
optGPX		(0 / 12 / 7)	(0 / 19 / 0)
optLTGA	(7 / 12 / 0)		(7 / 12 / 0)
MLS	(0 / 19 / 0)	(0 / 12 / 7)	

6 Conclusions

In this chapter, we will give our conclusions in Section 6.1. Furthermore, we will discuss interesting observations and give recommendations for future work in Section 6.2.

When an optimum or optima are described, this pertains a local optimum or local optima. Such a local optimum may or may not be the global optimum, but should this be the case, this will be stated specifically. As mentioned in 3.1.3, fitness of a solution and the objective value of the fitness of a solution are used interchangeably.

6.1 Conclusions

Local Search Variants

Regarding the local search variants, it becomes clear that the algorithms using local search to an optimum (Algorithm 3) outperform the algorithms using local search iterating over all elements once (Algorithm 2).

This can be explained by the following: Both GPX and LTGA start with random solutions, which most likely are not optima. Both algorithms proceed to recombine solutions to generate a new population. This process of recombination (using the greedy partitioning and the linkage tree approaches, respectively) is destructive for at least some of the clusters. From Section 5.1.1 and from the numerical results, it is clear that improving the cluster assignment of every element once after this destructive operation likely does not lead to an optimal solution. In contrast, optGPX and optLTGA always end up with solutions that are optima. The recombination process is as destructive as for GPX and LTGA, but in this case, it is guaranteed that the newly created population consists of only solutions that are optima. The fitness of a solution that is an optimum (be it local or global), will very likely be better than the fitness of a solution that is not. Thus, recombination for optGPX and optLTGA starts with fitter solutions with fitter individual clusters and also generates offspring that will be improved to an optimum again.

The fitness of optGPX and optLTGA solutions is higher for GPX and LTGA solutions when looking at it from a point of optima. However, this higher fitness comes at the cost of more evaluations and thus more time required. This gives GPX and LTGA more opportunity to learn good clusters than optGPX and optLTGA. As the results show however, the higher fitness given by the local search to an optimum outweighs the cost of evaluations and time that could be spent on learning or utilizing cluster structure. We can thus conclude that, for the algorithms considered, using the local search which only stops when an optimum was found is more effective than its counterpart which only improves each element at most once.

GPX and LTGA

Regarding the difference in GPX and the LTGA approaches, we see that when using the number of evaluations as termination criterion, LTGA outperforms GPX. In contrast, GPX outperforms LTGA when using the run time as the termination criterion, albeit not as decisively.

This can be explained by the following: The process of calculating the LTs and using the information found for recombination in the LTGA algorithm is more time-consuming than the process of using cluster fitness to decide how to recombine in the GPX algorithm. It is very likely that LTGA learns more about good clusterings than GPX, on average per evaluation. As such, the time spent per evaluation lies higher in the LTGA algorithm, but this results in a higher fitness value per evaluation overall. The GPX algorithm does not spend as much time per evaluation, which is why many more evaluations fit in the same time period than LTGA evaluations do. In the experiments where the time limit is the termination criterion, GPX is more time efficient per evaluation, meaning that more generations can be generated and therefore more time is spent on local search. The LTGA algorithm is less dependent on the effectiveness of the LS operator than the GPX algorithm by learning structure more explicitly, while the GPX algorithm emphasizes the success of the LS operator and uses the good results of the LS operator to build new solutions from there.

optGPX, optLTGA and MLS

Regarding the optGPX, optLTGA and the MLS algorithms, we see that in all experiments, performance is not significantly better for one or the other for most instances. This indicates that the local search operator has more influence than the recombination operator getting to fit solutions. The optGPX algorithm performs slightly better than the optLTGA algorithm in general. MLS performs better or equally well than both optGPX and opgLTGA. However, we see that when the run time is increased, the difference in performance between MLS and optGPX becomes negligible.

We have already established that the local search operator has a large influence. We have also established that spending extra time on learning structure via a LT gives fitter solutions per evaluation. What we look at now is, given the better local search function, does learning structure via a LT or using information about individual cluster fitness via GPX provide better results than a simple multi-start local search? The answer becomes clear from our results: MLS never performs significantly worse than either optGPX or optLTGA and in some cases performs significantly better. Thus, we can conclude that for our representation of the clustering problem and our evaluation function, the effort of creating a genetic algorithm does not provide any advantage over a rather simple search approach. Moreover, the MLS algorithm can be simplified by not using the framework described in Algorithm 1. This would also speed up the algorithm by removing (for MLS) unnecessary overhead. The simpler, sped up algorithm would then spend less time per evaluation and therefore would be even more efficient in solution fitness per evaluation.

Rand Index

Looking at the Rand Index gives us insight in how good the found solutions are compared to the clustering that was aimed for. This gives us information about the performance regarding solution quality. It is important to note that looking at the Rand Index values does not give any indication as to how the different studied approaches perform. Rather, this pertains to how well the evaluation method fits the problem instance.

From Tables 5.16 and 5.17, it becomes clear that better (lower) fitness values do not guarantee better (higher) Rand Index values. Looking at correlation plots and coefficients between these two variables gives us a clearer perspective. For about half of the instances (*spheric*, *structured*, *iris*, *wine*, *ecoli*) we can see a correlation that is roughly linear. Less so, but where one can still imagine a linear-like correlation are the *unbalanced*, *diabetes*, and *banknote* instances. Completely not resembling a linear correlation remain the *studenteval*, *cardio-3*, and *cardio-10* instances.

We have to keep the linearity of the plotted data in mind when looking at the correlation coefficients (Table 5.18). Many of the instances show a strong negative correlation. Some instances (*structured*, *iris*, *wine*, *ecoli*) come close to a perfect negative correlation. A strong negative correlation between fitness and Rand Index would be ideal: it means that as the fitness of a solution gets better (lower), the likeness of the solution to the perfect solution also gets better (higher).

That the correlation coefficient is close to 0 for the *studenteval* instance is not surprising given the plot. That the correlation coefficients are closer to $r = 1.0$ than to $r = 0$ for the *cardio-3* and *cardio-10* instances is more surprising. A possible explanation could be the impact of the number of elements in the regions of lower fitness values. The number of elements there is much higher, thus having a larger influence on the correlation coefficient than the elements in the regions of higher fitness values have. However, the plots show such non-linearity that the r-values do not reflect the correlation correctly either way.

Then, when comparing our Rand Indices with those found in (Agustın-Blas et al., 2012) (Table 5.19), we can see that even though our internal evaluation measure (SSE) is simpler than DB or S, the Rand Indices found from our experiments are quite close to the Rand Indices from (Agustın-Blas et al., 2012). It is of importance to note that we cannot determine which approach is better as the data sets are different, even when created using the same parameters. That the internal evaluation measures are different is another reason why no direct comparison can be made. The aforementioned correlation between internal measure value and external measure value is different for each problem instance and especially when using different internal measures. Still, the Rand Indices being close indicate that our higher-scoring approaches may work as well as the GGA approaches.

Research Questions

To return to our research questions mentioned in Chapter 1, we will first repeat them.

> *Can we use a greedy partitioning crossover approach to solve the clustering problem?*
>
> *Can we use a linkage tree genetic algorithm to solve the clustering problem?*

These questions are answered through one additional question.

> *How do the performances of the GPX, LTGA, and MLS approaches in solving the clustering problem compare?*

From the above, we can answer our research questions. The first and second questions have the same answer: yes. This answer comes with an additional commentary though. The clustering problem can be handled by these approaches, but these approaches are not necessarily the best and not the easiest approaches. This is where answer to the third question provides more nuance: the GPX and the LTGA approaches solve the presented problem instances to a satisfactory degree, but the easier MLS approach solves the presented problem instances as well. The MLS approach is either as good or better in solving the given problem instances. We can thus conclude that the more complicated approaches of GPX and LTGA do not return increased performance on the invested time to create these approaches.

6.2 Discussion and Future work

In this section, we discuss several observations and some additional thoughts on the conclusions. For the sake of conciseness, we mention GPX and LTGA approaches in this section, meaning the GA method, independent from the LS operator. So GPX here stands for both GPX and optGPX, while LTGA stands for both LTGA and optLTGA.

As mentioned, the results clearly show that MLS is the overall best-performing approach. This means that the best solution, can be found easily using a repeated hill-climbing algorithm. The main benefit one would expect from our GA approaches, namely finding better solutions than can be found by just hill-climbing, does not show. The question arises, why do we not see the influences from our crossover operators? Some possible explanations come to mind: the chosen problem instances are not complex or large enough for it to matter or the algorithms did not get enough time for the crossover operators' benefits to show.

When we look at the chosen problem instances, we can see that they are quite diverse in numbers of data points, clusters, and variables. They were chosen to represent common clustering problems encountered throughout the literature. It is possible that the chosen instances were not complex or large enough, but we deem this unlikely. When examining larger instances, the times required for local search to improve solutions to an optimum become larger. As both our studied algorithms still require LS to perform the improvements, the run times for our approaches would become infeasible to solve such a problem. If we could incorporate a different way to perform improvements, or limit the LS operator in some other way, it may be possible to use the GPX or LTGA approaches to solve instances of some orders of magnitude larger. An interesting area of research would be to look at the fitness landscape for the studied problem instances. This may very well provide additional insight into why the MLS approach dominates the other approaches.

Increasing the time GPX and LTGA can spend increases the effectiveness of the algorithms in solving the presented problem instances, but they do not get more effective than MLS. Relatively, they go from worse than MLS to more equal to MLS. Giving the algorithms more time will not increase performance for many of the problem instances, as MLS and optGPX, and to a lesser degree optLTGA, already find the same best solution for all runs for many of the instances. This was found in the experiments with extended time limits. This indicates that allowing more time to be spent will not improve results for most instances. Some instances will benefit from a higher time limit, but this will be the case for all algorithms.

Something else of considerable interest is the issue of increasing the Rand Index for all problems, especially the problems where the Rand Index was below 0.9. As for some instances the same optimum was reached in all runs of an experiment, we could assume that this is the global optimum. If this is so, we would like the corresponding Rand Index to be 1.0. However, this was never the case. The problem here thus lies with the evaluation function. If a different evaluation function would be used, it would be likely that most, if not all, optima would change. More optima could appear, or fewer optima could remain. Either way, from the correlations originating from our experiments and the simple observation that the Rand Indices were not always close to or exactly 1.0, we can see an important next step in the process of determining the value and performance of the GPX and LTGA approaches in the general clustering problem.

One would need to look at different evaluation methods to eventually find better final results (determined through the Rand Index, for example). Comparing how the studied algorithms interact with different evaluation methods would be an interesting research on its own. It is possible that the LS operator has an easy time improving solutions using the SSE evaluation, as used in this thesis. Other evaluation methods may be harder to use for the LS operator, resulting in the rest of a GA becoming more important. Even if the LS operator stays as effective, it may consume so much time that a search to an optimum could become infeasible. Studies into whether greedy partitioning or the structure learned through linkage trees become more effective compared to the MLS approach when using different evaluation functions seem to be a possible next area of investigation.

It is difficult to say what the influence of the representation is on the ability to solve problem instances. The chosen representation may in general, or even specifically for these approaches, not be the best one, as another representation based on the cluster centres exists and is used with success (e.g. Maulik and Bandyopadhyay, 2000; Siddiqi and Sait, 2017). It would be interesting to see how the studied GPX and LTGA approaches would perform when adapted to fit this different representation. Comparing the effectiveness of the algorithms using the different representations could also allow one to gain additional insight into why one representation possibly works better than the other.

Bibliography

Agarwal, Pankaj K and Nabil H Mustafa (2004). “k-Means projective clustering”. In: *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, pp. 155–165.

Aggarwal, Charu C and Philip S Yu (2000). *Finding generalized projected clusters in high dimensional spaces*. Vol. 29. 2. ACM.

Aggarwal, Charu C and Philip S. Yu (2002). “Redefining clustering for high-dimensional applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 14.2, pp. 210–225.

Aggarwal, Charu C et al. (1999). “Fast algorithms for projected clustering”. In: *ACM SIGMoD Record*. Vol. 28. 2. ACM, pp. 61–72.

Agrawal, Rakesh et al. (1998). *Automatic subspace clustering of high dimensional data for data mining applications*. Vol. 27. ACM.

Agustın-Blas, LE et al. (2012). “A new grouping genetic algorithm for clustering problems”. In: *Expert Systems with Applications* 39.10, pp. 9695–9703.

Al-Sultan, Khaled S (1995). “A tabu search approach to the clustering problem”. In: *Pattern Recognition* 28.9, pp. 1443–1451.

Amir, Amihood et al. (2003). “Analyzing high-dimensional data by subspace validity”. In: *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE, pp. 473–476.

Anderberg, Michael R (1973). *Cluster analysis for applications. Monographs and textbooks on probability and mathematical statistics*.

Andritsos, Periklis et al. (2002). “Data clustering techniques”. In: *Rapport technique, University of Toronto. Department of Computer Science*.

Ankerst, Mihael et al. (1999). “OPTICS: ordering points to identify the clustering structure”. In: *ACM Sigmod Record*. Vol. 28. ACM, pp. 49–60.

Babu, G Phanendra and M Narasimha Murty (1993). “A near-optimal initial seed value selection in k-means means algorithm using a genetic algorithm”. In: *Pattern Recognition Letters* 14.10, pp. 763–769.

Banfield, Jeffrey D and Adrian E Raftery (1993). “Model-based Gaussian and non-Gaussian clustering”. In: *Biometrics*, pp. 803–821.

Barbará, Daniel and Ping Chen (2000). “Using the fractal dimension to cluster datasets”. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 260–264.

Barbará, Daniel, Yi Li, and Julia Couto (2002). "COOLCAT: an entropy-based algorithm for categorical clustering". In: *Proceedings of the eleventh international conference on Information and knowledge management*. ACM, pp. 582–589.

Baulieu, FB (1989). "A classification of presence/absence based dissimilarity coefficients". In: *Journal of Classification* 6.1, pp. 233–246.

Bell, David A. et al. (1990). "Application of simulated annealing to clustering tuples in databases". In: *Journal of the American Society for Information Science* 41.2, p. 98.

Berkhin, Pavel (2006). "A survey of clustering data mining techniques". In: *Grouping multidimensional data*. Springer, pp. 25–71.

Beyer, Kevin et al. (1999). "When is "nearest neighbor" meaningful?" In: *International conference on database theory*. Springer, pp. 217–235.

Bezdek, James C (2013). *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media.

Bezdek, James Christian (1973). "Fuzzy mathematics in pattern classification". PhD. Cornell University, Ithaca, NY.

Bobrowski, Leon and James C Bezdek (1991). "c-means clustering with the l_l and l_∞ norms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.3, pp. 545–554.

Bosman, Peter AN and Dirk Thierens (2012). "On measures to build linkage trees in LTGA". In: *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 276–285.

Bottou, Leon, Yoshua Bengio, et al. (1995). "Convergence properties of the k-means algorithms". In: *Advances in neural information processing systems*, pp. 585–592.

Bradley, Paul S and Usama M Fayyad (1998). "Refining Initial Points for K-Means Clustering." In: *ICML*. Vol. 98. Citeseer, pp. 91–99.

Buhmann, Joachim (1995). "Data clustering and learning". In: *The Handbook of Brain Theory and Neural Networks*, pp. 278–281.

Campos, Diogo Ayres-de et al. (2000). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/banknote+authentication`.

Cao, Yongqiang and Jianhong Wu (2002). "Projective ART for clustering data sets in high dimensional spaces". In: *Neural networks* 15.1, pp. 105–120.

Cattell, Raymond B (1949). "Rp and other coefficients of pattern similarity". In: *Psychometrika* 14.4, pp. 279–298.

Chaturvedi, Anil, Paul E Green, and J Douglas Caroll (2001). "K-modes clustering". In: *Journal of Classification* 18.1, pp. 35–55.

Cheng, Chun-Hung, Ada Waichee Fu, and Yi Zhang (1999). "Entropy-based subspace clustering for mining numerical data". In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 84–93.

Cheng, Chun-Hung, Wing-Kin Lee, and Kam-Fai Wong (2002). "A genetic algorithm-based clustering approach for database partitioning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 32.3, pp. 215–230.

Dalal, M. A. and N. D. Harale (2011). "A Survey on Clustering in Data Mining". In: *Proceedings of the International Conference &#38; Workshop on Emerging Trends in*

Technology. ICWET '11. New York, NY, USA: ACM, pp. 559–562. ISBN: 978-1-4503-0449-8.

Dash, Manoranjan, Huan Liu, and Xiaowei Xu (2001). "'1+ 1> 2': merging distance and density based clustering". In: *Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on*. IEEE, pp. 32–39.

Defays, Daniel (1977). "An efficient algorithm for a complete link method". In: *The Computer Journal* 20.4, pp. 364–366.

Delgado, Miguel, Antonio Gómez Skármeta, and Humberto Martínez Barberá (1997). "A tabu search approach to the fuzzy clustering problem". In: *Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on*. Vol. 1. IEEE, pp. 125–130.

Doerksen, Helene (2012). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/banknote+authentication`.

Duran, Benamin S and Patrick L Odell (1974). *Cluster analysis, a survey, volume 100 of lectures notes in economics and mathematical systems*.

Edwards, Anthony WF and L Luka Cavalli-Sforza (1965). "A method for cluster analysis". In: *Biometrics*, pp. 362–375.

El-Sonbaty, Yasser, MA Ismail, and Mohamed Farouk (2004). "An efficient density based clustering algorithm for large databases". In: *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. IEEE, pp. 673–677.

Ester, Martin et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96, pp. 226–231.

Everitt, BS, S Landau, and M Leese (1993). "Cluster analysis. 1993". In: *Edward Arnold and Halsted Press,*

Faber, Vance (1994). "Clustering and the continuous k-means algorithm". In: *Los Alamos Science* 22.138144.21.

Fisher, Douglas H (1987). "Knowledge acquisition via incremental conceptual clustering". In: *Machine learning* 2.2, pp. 139–172.

Fisher, RA (1936). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/iris`.

Fogel, LJ, AJ Owens, and MJ Walsh (1965). "Intelligent decision-making through a simulation of evolution". In: *IEEE Transactions on Human Factors in Electronics* 1, pp. 13–23.

Forina, M (1991). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/Wine`.

Fujikawa, Yoshikazu and TuBao Ho (2002). "Cluster-based algorithms for dealing with missing values". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pp. 549–554.

Galinier, Philippe and Jin-Kao Hao (1999). "Hybrid evolutionary algorithms for graph coloring". In: *Journal of combinatorial optimization* 3.4, pp. 379–397.

Gan, Guojun, Chaoqun Ma, and Jianhong Wu (2007). *Data clustering: theory, algorithms, and applications*. Vol. 20. Siam.

Gan, Guojun, Zijiang Yang, and Jianhong Wu (2005). “A genetic k-modes algorithm for clustering categorical data”. In: *International Conference on Advanced Data Mining and Applications*. Springer, pp. 195–202.

Glass, Celia A and Adam Prügel-Bennett (2003). “Genetic algorithm for graph coloring: exploration of Galinier and Hao’s algorithm”. In: *Journal of Combinatorial Optimization* 7.3, pp. 229–236.

Glover, Fred (1989). “Tabu search—part I”. In: *ORSA Journal on computing* 1.3, pp. 190–206.

Glover, Fred (1990). “Tabu search—part II”. In: *ORSA Journal on computing* 2.1, pp. 4–32.

Glover, Fred and Eric Taillard (1993). “A user’s guide to tabu search”. In: *Annals of operations research* 41.1, pp. 1–28.

Goebel, Michael and Le Gruenwald (1999). “A survey of data mining and knowledge discovery software tools”. In: *ACM SIGKDD explorations newsletter* 1.1, pp. 20–33.

Goil, Sanjay, Harsha Nagesh, and Alok Choudhary (1999). “MAFIA: Efficient and scalable subspace clustering for very large data sets”. In: *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 443–452.

Goldberg, David (1989). “Genetic algorithms in optimization, search and machine learning”. In: *Reading: Addison-Wesley*.

Goodall, David W (1966). “A new similarity index based on probability”. In: *Biometrics*, pp. 882–907.

Gordon, Allan D (1987). “A review of hierarchical classification”. In: *Journal of the Royal Statistical Society. Series A (General)*, pp. 119–137.

Gordon, Allan D (1996). “Hierarchical classification”. In: *Clustering and classification*, pp. 65–121.

Gowda, K Chidananda and Edwin Diday (1992). “Symbolic clustering using a new similarity measure”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 22.2, pp. 368–378.

Gower, John C and Pierre Legendre (1986). “Metric and Euclidean properties of dissimilarity coefficients”. In: *Journal of classification* 3.1, pp. 5–48.

Gower, John C and GJS Ross (1969). “Minimum spanning trees and single linkage cluster analysis”. In: *Applied statistics*, pp. 54–64.

Green, Paul E and Vithala R Rao (1969). “A note on proximity measures and cluster analysis”. In: *Journal of Marketing Research* 6.3, pp. 359–364.

Greene, William A (2003). “Unsupervised hierarchical clustering via a genetic algorithm”. In: *Evolutionary Computation, 2003. CEC’03. The 2003 Congress on*. Vol. 2. IEEE, pp. 998–1005.

Guha, Sudipto, Rajeev Rastogi, and Kyuseok Shim (1998). “CURE: an efficient clustering algorithm for large databases”. In: *ACM SIGMOD Record*. Vol. 27. ACM, pp. 73–84.

Guha, Sudipto, Rajeev Rastogi, and Kyuseok Shim (1999). "ROCK: A robust clustering algorithm for categorical attributes". In: *Data Engineering, 1999. Proceedings., 15th International Conference on*. IEEE, pp. 512–521.

Gunduz, N and E Fokoue (2013). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/Turkiye+Student+Evaluation`.

Han, J, M Kamber, and AKH Tung (2001). *Spatial Clutering Methods in Data Mining: A Survey (2001)*.

Hansen, Pierre and Nenad Mladenović (2001). "Variable neighborhood search: Principles and applications". In: *European journal of operational research* 130.3, pp. 449–467.

Har-Peled, Sariel and Kasturi Varadarajan (2002). "Projective clustering in high dimensions using core-sets". In: *Proceedings of the eighteenth annual symposium on Computational geometry*. ACM, pp. 312–318.

Hartigan, John A (1967). "Representation of similarity matrices by trees". In: *Journal of the American Statistical Association* 62.320, pp. 1140–1158.

Hertz, John, Anders Krogh, and Richard G Palmer (1991). *Introduction to the theory of neural computation*. Vol. 1. Basic Books.

Hinneburg, Alexander and Daniel A Keim (1998). "An efficient approach to clustering in large multimedia databases with noise". In: *KDD*. Vol. 98, pp. 58–65.

Hinneburg, Alexander and Daniel A Keim (1999). "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering". In:

Holland, John H (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Höppner, Frank (1999). *Fuzzy cluster analysis: methods for classification, data analysis and image recognition*. John Wiley & Sons.

Hua, Kien A, Sheau-Dong Lang, and Wen K Lee (1994). "A decomposition-based simulated annealing technique for data clustering". In: *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, pp. 117–128.

Huang, Zhexue (1997). "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining." In: *DMKD*, p. 0.

Huang, Zhexue (1998). "Extensions to the k-means algorithm for clustering large data sets with categorical values". In: *Data mining and knowledge discovery* 2.3, pp. 283–304.

Huang, Zhexue and Michael K Ng (1999). "A fuzzy k-modes algorithm for clustering categorical data". In: *IEEE Transactions on Fuzzy Systems* 7.4, pp. 446–452.

Hubalek, Zdenek (1982). "Coefficients of association and similarity, based on binary (presence-absence) data: an evaluation". In: *Biological Reviews* 57.4, pp. 669–689.

Jain, A. K., M. N. Murty, and P. J. Flynn (1999). "Data Clustering: A Review". In: *ACM Comput. Surv.* 31.3, pp. 264–323. ISSN: 0360-0300.

Jain, Anil K (2010). "Data clustering: 50 years beyond K-means". In: *Pattern recognition letters* 31.8, pp. 651–666.

Jain, Anil K and Richard C Dubes (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

Jain, Anil K, Jianchang Mao, and K Moidin Mohiuddin (1996). "Artificial neural networks: A tutorial". In: *IEEE computer* 29.3, pp. 31–44.

Jiang, Tianzi and Song De Ma (1996). "Cluster analysis using genetic algorithms". In: *Signal Processing, 1996., 3rd International Conference on*. Vol. 2. IEEE, pp. 1277–1279.

Kailing, Karin, Hans-Peter Kriegel, and Peer Kröger (2004). "Density-connected subspace clustering for high-dimensional data". In: *Proc. SDM*. Vol. 4. SIAM.

Kanungo, Tapas et al. (2002). "An efficient k-means clustering algorithm: Analysis and implementation". In: *IEEE transactions on pattern analysis and machine intelligence* 24.7, pp. 881–892.

Karypis, George, Eui-Hong Han, and Vipin Kumar (1999). "Chameleon: Hierarchical clustering using dynamic modeling". In: *Computer* 32.8, pp. 68–75.

Ke, Qifa and Takeo Kanade (2004). "Robust subspace clustering by combined use of knnd metric and svd algorithm". In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 2. IEEE, pp. II–592.

Klein, Raymond W and Richard C Dubes (1989). "Experiments in projection and clustering by simulated annealing". In: *Pattern Recognition* 22.2, pp. 213–220.

Kohonen, Teuvo (1990). "The self-organizing map". In: *Proceedings of the IEEE* 78.9, pp. 1464–1480.

Kolatch, Erica et al. (2001). "Clustering algorithms for spatial databases: A survey". In: *PDF is available on the Web*.

Krishna, K and M Narasimha Murty (1999). "Genetic K-means algorithm". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.3, pp. 433–439.

Krishnan, Thriyambakam and G McLachlan (1997). "The EM algorithm and extensions". In: *Wiley* 1.997, pp. 58–60.

Krishnapuram, Raghu and C-P Freg (1991). "Fuzzy algorithms to find linear and planar clusters and their applications". In: *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*. IEEE, pp. 426–431.

Likas, Aristidis, Nikos Vlassis, and Jakob J Verbeek (2003). "The global k-means clustering algorithm". In: *Pattern recognition* 36.2, pp. 451–461.

Liu, Bing, Yiyuan Xia, and Philip S Yu (2000). "Clustering through decision tree construction". In: *Proceedings of the ninth international conference on Information and knowledge management*. ACM, pp. 20–29.

MacQueen, James et al. (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA., pp. 281–297.

Mann, Amandeep Kaur and Navneet Kaur (2013). "Review paper on clustering techniques". In: *Global Journal of Computer Science and Technology* 13.5.

Mao, Jianchang and Anil K Jain (1996). "A self-organizing network for hyperellipsoidal clustering (HEC)". In: *Ieee transactions on neural networks* 7.1, pp. 16–29.

Martinez, Wendy L et al. (2010). *Exploratory data analysis with MATLAB*. CRC Press.

Maulik, Ujjwal and Sanghamitra Bandyopadhyay (2000). "Genetic algorithm-based clustering technique". In: *Pattern recognition* 33.9, pp. 1455–1465.

McErlean, FJ, David A. Bell, and Sally I. McClean (1990). "The use of simulated annealing for clustering data in databases". In: *Information Systems* 15.2, pp. 233–245.

McLachlan, Geoffrey J and Kaye E Basford (1988). "Mixture models. Inference and applications to clustering". In: *Statistics: Textbooks and Monographs, New York: Dekker, 1988* 1.

Murtagh, Fionn (1983). "A survey of recent advances in hierarchical clustering algorithms". In: *The Computer Journal* 26.4, pp. 354–359.

Murtagh, Fionn (1985). "Multidimensional clustering algorithms". In: *Compstat Lectures, Vienna: Physika Verlag, 1985*.

Nagesh, Harsha S, Sanjay Goil, and Alok Choudhary (2000). "A scalable parallel subspace clustering algorithm for massive data sets". In: *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, pp. 477–484.

Nagesh, Harsha S, Sanjay Goil, and Alok N Choudhary (2001). "Adaptive Grids for Clustering Massive Data Sets." In: *SDM*. SIAM, pp. 1–17.

Nagpal, Arpita, Arnan Jatain, and Deepti Gaur (2013). "Review based on data clustering algorithms". In: *Information & Communication Technologies (ICT), 2013 IEEE Conference on*. IEEE, pp. 298–303.

Nakai, Kenta and Minoru Kanehisa (1991). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/ecoli`.

Narahashi, Masaki and Einoshin Suzuki (2002). "Subspace clustering based on compressibility". In: *International Conference on Discovery Science*. Springer, pp. 435–440.

Ng, Michael K and Joyce C Wong (2002). "Clustering categorical data sets using tabu search techniques". In: *Pattern Recognition* 35.12, pp. 2783–2790.

Ng, Raymond T. and Jiawei Han (2002). "CLARANS: A Method for Clustering Objects for Spatial Data Mining". In: *IEEE Trans. on Knowl. and Data Eng.* 14.5, pp. 1003–1016. ISSN: 1041-4347. DOI: `10.1109/TKDE.2002.1033770`. URL: `http://dx.doi.org/10.1109/TKDE.2002.1033770`.

Oja, Erkki (1992). "Principal components, minor components, and linear neural networks". In: *Neural networks* 5.6, pp. 927–935.

Olson, Clark F (1995). "Parallel algorithms for hierarchical clustering". In: *Parallel computing* 21.8, pp. 1313–1325.

Parsons, Lance, Ehtesham Haque, and Huan Liu (2004). "Subspace clustering for high dimensional data: a review". In: *ACM SIGKDD Explorations Newsletter* 6.1, pp. 90–105.

Parsons, Lance, Ehtesham Haque, Huan Liu, et al. (2004). "Evaluating subspace clustering algorithms". In: *Workshop on Clustering High Dimensional Data and its Applications, SIAM Int. Conf. on Data Mining*. Citeseer, pp. 48–56.

Procopiuc, Cecilia M et al. (2002). "A Monte Carlo algorithm for fast projective clustering". In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, pp. 418–427.

Rand, William M (1971). "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical association* 66.336, pp. 846–850.

Ribeiro Filho, José L, Philip C Treleaven, and Cesare Alippi (1994). "Genetic-algorithm programming environments". In: *Computer* 27.6, pp. 28–43.

Rousseeuw, Peter J and L Kaufman (1990). *Finding Groups in Data*. Wiley Online Library.

Sarafis, Ioannis A, Phil W Trinder, and Ali MS Zalzala (2003). "Towards effective subspace clustering with an evolutionary algorithm". In: *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. Vol. 2. IEEE, pp. 797–806.

Sarle, Warren S (1991). "Finding Groups in Data: An Introduction to Cluster Analysis". In: *Journal of the American Statistical Association* 86.415, pp. 830–833.

Schikuta, Erich (1996). "Grid-clustering: An efficient hierarchical clustering method for very large data sets". In: *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. Vol. 2. IEEE, pp. 101–105.

Schikuta, Erich and Martin Erhart (1997). "The BANG-clustering system: Grid-based data analysis". In: *International Symposium on Intelligent Data Analysis*. Springer, pp. 513–524.

Schwefel, Hans-Paul (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

Selim, Shokri Z and K1 Alsultan (1991). "A simulated annealing algorithm for the clustering problem". In: *Pattern recognition* 24.10, pp. 1003–1008.

Selim, Shokri Z and Mohamed A Ismail (1984). "K-means-type algorithms: a generalized convergence theorem and characterization of local optimality". In: *IEEE Transactions on pattern analysis and machine intelligence* 1, pp. 81–87.

Sheikholeslami, Gholamhosein, Surojit Chatterjee, and Aidong Zhang (1998). "Wavecluster: A multi-resolution clustering approach for very large spatial databases". In: *VLDB*. Vol. 98, pp. 428–439.

Sibson, Robin (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method". In: *The computer journal* 16.1, pp. 30–34.

Siddiqi, Umair F and Sadiq M Sait (2017). "A New Heuristic for the Data Clustering Problem". In: *IEEE Access*.

Sigillito, Vincent (1990). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes`.

Sneath, Peter HA, Robert R Sokal, et al. (1973). *Numerical taxonomy. The principles and practice of numerical classification.*

Sokal, Robert R, Peter HA Sneath, et al. (1963). "Principles of numerical taxonomy." In: *Principles of numerical taxonomy.*

Späth, Helmuth (1980). *Cluster analysis algorithms for data reduction and classification of objects*. Vol. 4. Halsted Pr.

Sung, Chang Sup and Hyun Woong Jin (2000). "A tabu-search-based heuristic for clustering". In: *Pattern Recognition* 33.5, pp. 849–858.

Thierens, Dirk (2010). "The linkage tree genetic algorithm". In: *Parallel Problem Solving from Nature, PPSN XI*, pp. 264–273.

Thierens, Dirk and Peter AN Bosman (2011). "Optimal mixing evolutionary algorithms". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, pp. 617–624.

Wagstaff, Kiri et al. (2001). "Constrained k-means clustering with background knowledge". In: *ICML*. Vol. 1, pp. 577–584.

Wang, Haixun et al. (2004). "A fast algorithm for subspace clustering by pattern similarity". In: *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE, pp. 51–60.

Wang, Wei, Jiong Yang, Richard Muntz, et al. (1997). "STING: A statistical information grid approach to spatial data mining". In: *VLDB*. Vol. 97, pp. 186–195.

Willett, Peter (1988). "Recent trends in hierarchic document clustering: a critical review". In: *Information Processing & Management* 24.5, pp. 577–597.

Williams, WT and JM t Lambert (1966). "Multivariate methods in plant ecology: V. similarity analyses and information-analysis". In: *The Journal of Ecology*, pp. 427–445.

Wishart, David (2003). "K-means clustering with outlier detection, mixed variables and missing values". In: *Exploratory Data Analysis in Empirical Research*. Springer, pp. 216–226.

Woo, Kyoung-Gu et al. (2004). "FINDIT: a fast and intelligent subspace clustering algorithm using dimension voting". In: *Information and Software Technology* 46.4, pp. 255–271.

Xu, Hong-Bing, Hou-Jun Wang, and Chun-Guang Li (2002). "Fuzzy tabu search method for the clustering problem". In: *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*. Vol. 2. IEEE, pp. 876–880.

Xu, X et al. (1998). "A nonparametric clustering algorithm for knowledge discovery in large spatial databases". In: *Proc. of the Intl. Conf. on Data Engineering (ICDE'98).*

Ye, Nong et al. (2003). *The handbook of data mining*. Vol. 24. Lawrence Erlbaum Associates, Publishers Mahwah, NJ/London.

Zadeh, Lotfi A (1965). "Fuzzy sets". In: *Information and control* 8.3, pp. 338–353.

Zhang, Bin and Sargur N Srihari (2003). "Properties of binary vector dissimilarity measures". In: *Proc. JCIS Int'l Conf. Computer Vision, Pattern Recognition, and Image Processing*. Vol. 1.

Zhao, Yanchang and Junde Song (2001). "GDILC: a grid-based density-isoline clustering algorithm". In: *Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on*. Vol. 3. IEEE, pp. 140–145.