Metaheuristics and machine learning for joint stratification and sample allocation in survey design

Mervyn O'Luing

BA, MSc (UCC), MA, Prof Dip in Official Statistics (UCD)

NATIONAL UNIVERSITY OF IRELAND, CORK

School of Mathematical Sciences

Department of Mathematics

Thesis submitted for the degree of
Doctor of Philosophy

January 2022

Head of Department: Prof. Utz Roedig

Supervisors: Dr. Steve Prestwich

Prof. S. Armagan Tarim

Research supported by Science Foundation Ireland

Contents

List of Tables

I, Mervyn O'Luing, certify that this thesis is my own work and I have not obtained a degree in this university or elsewhere on the basis of the work submitted in this thesis.

Mervyn O'Luing

To my wife Karla, my parents and my family

Acknowledgements

I would like to thank, my PhD supervisors, Dr. Steven Prestwich and Prof. S. Armagan for their direction, guidance, encouragement and feedback over the course of my research.

I am grateful to Dr. Michael Cronin and Prof. Finbarr O'Sullivan for their motivation and instruction in the MSc in Data Science & Analytics program which was a stepping stone to the PhD and the path which led, by and by, to the thesis I am presenting today.

I would also like to thank my supervisors and colleagues in the Administrative Data Centre, Household Budget Survey section, and Methodology Division in the Central Statistics Office for their interest in and regard for my research. I should mention my colleague Dr. Thierry Vallée who generously gave his time to discuss my research, in particular with respect to tweaking the *Rcpp* code outlined in chapter 10.

Acknowledgements are also due to my family (living and deceased (Go ndéana Dia trócaire orthu)) for their facilitation, encouragement and support while I was completing the PhD.

I would like to show gratitude to the staff in the Insight Centre for Data Analytics. I would also like to thank the training unit in the Central Statistics Office for supporting my research. In addition thanks are also due to Science Foundation Ireland which supported my research under grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund. My research was also supported in part by a grant from Science Foundation Ireland under grant number 16/RC/3918 which is co-funded under the European Regional Development Fund.

In respect of chapter 7, I wish to acknowledge Steven Riesz of the Economic Statistical Methods Division of the U.S. Census Bureau and Brian J. McElroy of the Economic Reimbursable Survey Division of the U.S Census Bureau, both of whom answered questions which were of assistance in choosing which US Census Bureau data to use.

I would also like to thank Giulio Barcaroli and Marco Ballin, the co-authors of (Ballin and Barcaroli, 2013), for independently testing our GGA. Furthermore, I am grateful to Professor Roberto Benedetti for sharing the Matlab code for the hierarchical divisive algorithm used by Benedetti et al. (2008) to partition basic strata.

Last but not least, I am extremely grateful to the editorial staff and reviewers of Survey Methodology for their constructive suggestions in the review process for this journal submission, especially their suggestions for future work.

In respect of chapter 8, I wish to acknowledge the editorial staff and reviewers of Survey Methodology for their constructive suggestions in the review process for the journal submission on which this chapter is based, in particular the suggestion to compare the SAA with the classical genetic algorithm in (Ballin and Barcaroli, 2020)

in the case of continuous strata.

The material in all chapters is based upon work supported by the Insight Centre for Data Analytics and Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund. Also, this publication has emanated from research supported in part by a grant from Science Foundation Ireland under Grant number 16/RC/3918 which is co-funded under the European Regional Development Fund.

Abstract

In this thesis, we propose a number of metaheuristics and machine learning techniques to solve the joint stratification and sample allocation problem. Finding the optimal solution to this problem is hard when the sampling frame is large, and the evaluation algorithm is computationally burdensome.

To advance the research in this area, we explore and evaluate different algorithmic methods of modelling and solving this problem. Firstly, we propose a new genetic algorithm approach using "grouping" genetic operators instead of traditional operators. Experiments show a significant improvement in solution quality for similar computational effort. Next, we combine the capability of a simulated annealing algorithm to escape from local minima with delta evaluation to exploit the similarity between consecutive solutions and thereby reduce evaluation time. Comparisons with two recent algorithms show the simulated annealing algorithm attaining comparable solution qualities in less computation time.

Then, we consider the combination of the k-means and clustering algorithms with a hill climbing algorithm in stages and report the solution costs, evaluation times and training times. The multi-stage combinations generally compare well with recent algorithms, and provide the survey designer with a greater choice of algorithms to choose from.

Finally, we combine the explorative properties of an estimation of distribution algorithm (EDA) to model the probabilities of an atomic stratum belonging to different strata with the exploitative search properties of a simulated annealing algorithm to create a hybrid estimation of distribution algorithm (HEDA).

Results of comparisons with the best solution qualities from our earlier experiments show that the HEDA finds better solution qualities, but requires a longer total execution time than alternative approaches we considered.

Terminologies and symbols

In the sections that follow, the implementation of metaheuristics and machine learning algorithms, to learn solutions to a complex survey design problem, builds on and intersperses existing research from two broad fields: computer science and statistics. Each field has its own set of terminologies and mathematical symbols, representing various values.

For example, in machine learning there are symbols used to indicate hyper-parameters (specified before the learning process), or parameters (weights/values learned during the learning process), and in survey design we encounter symbols representing values such as probability and coefficient of variation. However, within each domain there are also different applications of some of these, and there is also some cross-over between the domains.

To illustrate this point, Ballin and Barcaroli (2013) used the symbol K to denote the number of atomic strata in the set L, whereas subsequently in Ballin and Barcaroli (2020) they refer to K means clustering, which relates to the number of strata (i.e. clusters) into which the atomic strata are grouped. Although we are further developing the research undertaken by Ballin and Barcaroli (2013, 2020), to avoid confusion, we use different notation in chapters 7 and 8. There, we use L to describe the number (and not the set) of atomic strata, rather than K which we intend to refer to the number of clusters, such as in k-means.

Likewise clusters and strata are, unless otherwise stated (i.e. self-organising maps and neural gas in chapter 9), used interchangeably in this thesis - which differs from the use of clusters in survey methodology broadly speaking. The above example of the use of K, in relation to strata/clusters, is also indicative of the evolving methods to search for solutions to this problem. As, when Ballin and Barcaroli (2013) wrote their paper, K-means clustering had not yet been identified as an algorithm for finding initial (if not optimal) solutions for this problem. It was later suggested by Lisic et al. (2018).

We mainly use the original terminology and mathematical symbols as they appear in the referenced work - although in a minority cases we do not. For example, again referring to K, even though we cite (Bezdek et al., 1984) where fuzzy k-means clustering is referred to as fuzzy c-means clustering, we have used the former in order to be consistent.

In describing the stratification problem, we intend that population refers to the finite set of units from which a stratified sample is drawn, in order to make inferences

about that population. However, in describing the genetic algorithms (chapter 7), we intend that population refers to a set of candidate solutions, which are iteratively evolved through a process of fitness testing, ranking, recombination and mutation. This reflects the use of the term "population" in the relevant literature.

Likewise, the same mathematical symbols have differing representations in separate works. Thus, the same symbols appear throughout this thesis, except with different meanings. For example, in chapter 2 we use the symbol ρ to denote a correlation coefficient, whereas later in Chapter 8 ρ is used to represent probability. Similarly, in chapter 3, in the Bethel-Chromy algorithm the parameter λ represents a Lagrangian multiplier, whereas in chapter 8 it is used as controlling factor in hyperparameter optimisation, and again in chapter 9 λ is used a number of times in separate clustering algorithms, e.g. self-organising maps and neural gas. On a broader note, these are λ values which are learned in their respective algorithms - however, they have a different purpose in each case. We explain the meaning of each term and symbol for the algorithm or section that it arises.

Chapter 1

Introduction

(Kish et al., 1965) describes survey design as the interaction of survey objectives with sample design. Sample design comprises two aspects: the process of selecting a sample and the process for computing sample estimates of population values. A column of variables contains the population or sample measurements for a characteristic. From these we calculate our estimates of population values. The values for such characteristics are recorded as continuous or discrete variables.

Target population variables (or attributes) relating to units (or individuals), e.g. total income, turnover, crop coverage, smoker indicators, etc. are those for which we wish to obtain quantitative or qualitative values in a survey. There is often more than one characteristic of interest to the survey designer.

Similarly, the survey designer may have more than one purpose to consider in designing a sample. Gonzalez and Eltinge (2010) state that surveys are inherently multipurpose with both anticipated and unanticipated uses of the collected data. This has the implications of reducing costs and enabling greater analysis possibilities with the collected data. There may also be pressures to consolidate and use the same sample for multiple surveys. This may be particularly true in surveys sponsored by governments (Valliant and Gentle, 1997).

In addition to sample design there are a number of other important aspects in the design of surveys which collectively can be called survey objectives. These objectives comprise: definition of the survey variables, methods of observation, methods of analysis, utilization of survey results and the desired precision of survey results.

Survey objectives typically require an achieved sample (i.e. the data collected from respondents) that is sufficiently representative of the target population charac-

teristics. The achieved sample size is, in practice, less than 100% of the actual sample size. This is due to varying response rates for different cohorts within a given survey.

Examples of measures for dealing with non-response can be found in (Bethlehem, 2009; Cochran, 1977; Groves et al., 2011; Lohr, 2019; Särndal et al., 2003). Such measures comprise various types of mitigations made in the survey design phase or during and post data collection. In the context of the joint stratification and sample allocation method, an example of one of these measures is taking a larger sample size to mitigate for the expected non-response.

In this sense, once the minimum sample has been calculated, this may be considered as an estimation of the achieved sample size. We could then gross this by a factor of 1 divided by the expected response rate to get the required sample size. For example, if the sample size is $1,000$ and the expected response rate is 60% we multiply the achieved sample size by $1/0.6$ or 1.67, giving a required sample size of $1,670$ (Bethlehem, 2009). In stratified samples where there are unequal response rates across the strata, we simply adjust for the response rate in each stratum. Furthermore, Baillargeon and Rivest (2009) provides a formula that can take into account varying non-response rates when minimising the anticipated achieved sample size for univariate stratification, but it can return negative values and recommended for use in cases where it can return positive values. However, in this thesis we will assume a 100% response rate.

With the full response rate in mind, we focus on the sample selection, estimation of population values and desired precision aspects of survey design. We use *stratified random sampling* as a mechanism to encapsulate these three aspects.

There are a number of methodological approaches for stratified sampling. Basically, these entail the use of existing auxiliary and/or target information to determine a stratification before selecting a sample, and are discussed in chapter 2. With regards to the determination of strata, in this thesis, we focus on the stratification of basic strata (previously determined from such information) into groups or *strata*.

This is a partitioning problem in which the number of possible solutions grows with the set of basic strata. For larger sets, the problem cannot be solved to optimality, because the search for an optimum solution requires a prohibitive amount of computation time. This compels one to use metaheuristic and machine learning algorithms, which do not guarantee optimal solutions, but can provide approximately-optimal solutions in an acceptable time interval. This assumes that the survey designer has the relevant knowledge from the fields of statistics and computer

science.

Prior to our research, two algorithmic approaches had been developed to tackle this problem, i.e. the hierarchical algorithm proposed by (Benedetti et al., 2008), and the genetic algorithm proposed by (Ballin and Barcaroli, 2013). In this thesis, We expand on this research with a suite of evolutionary, local search and clustering algorithms - with the goal of finding solutions that are "good enough" in a computing time that is "small enough" (Sörensen and Glover, 2013). We present these algorithms in this thesis.

1.1 Outline

This thesis is organised as follows:

In chapter 2, we provide the background and related work to this problem. This comprises a discussion on stratified sampling and sample allocation. We provide details on three approaches for stratified random sampling and sample allocation. Included in these details are particulars on the problem of joint determination of stratification and sample allocation, which can be applied to univariate and multivariate survey design scenarios - and for which we have developed the algorithms, presented in this thesis, to search for solutions. We also introduce the basic (atomic or continuous) strata which we intend to partition into *strata*.

In chapter 3 we provide the problem description. This chapter includes details on the construction of atomic strata (which can also be applied to continuous strata). We also provide particulars on the objective function and variance (precision) constraints. This is followed by information on the allocation model, and subsequently the Bethel-Chromy algorithm which is used to find the sample size (solution quality) of each stratification.

In chapter 7, we present a grouping genetic algorithm (GGA). This uses grouping genetic operators instead of the traditional operators in the classical genetic algorithm (CGA). In empirical experiments, we show a significant improvement in solution quality for similar computational effort with the GGA over the CGA.

In chapter 8, we combine simulated annealing with delta evaluation to solve the problem. The simulated annealing algorithm (SAA) enables escape from local minima and uses delta evaluation to exploit the similarity between consecutive solutions and thereby reduce the evaluation time. We compare the SAA with the CGA and GGA. In both cases the SAA attains a solution of comparable quality in considerably less computation time.

In chapter 9, we compare the multi-stage combination of k-means and clustering algorithms with a hill-climbing algorithm - with the GGA and SAA, from chapters 7 and 8 respectively, and report the solution costs, evaluation times and training times. The multi-stage combinations generally compare well with the recent algorithms, both in the case of atomic and continuous strata.

In chapter 10, we combine an estimation of distribution algorithm with simulated annealing to enhance the exploitative properties of the former. The hybrid estimation of distribution algorithm (HEDA) attains the best results, when compared to the results from the same experiments in earlier chapters. However, the results also indicate that it requires longer computation times than the alternative algorithms we described.

1.2 Contributions

More generally, we have research and implemented diverse computer science techniques to solve a complex and computationally expensive stratification and sampling problem, and applied them to different instances of that problem.

- We advance the use of a grouping genetic algorithm, a simulated annealing algorithm, fuzzy clustering, expectation maximisation, self organising maps, neural gas, a hill climbing algorithm, and a hybrid estimation of distribution algorithm for the creation of starting solutions or near-optimal solutions in a novel combinatorial optimisation problem in statistics.
- For the grouping genetic algorithm, the use of grouping operators tackles symmetry and is less likely to break up good solutions. We have also added inversion to diversify the selection of groups when creating offspring. Accordingly, the algorithm produces substantial gains in processing efficiency over the classical genetic algorithm in the *SamplingStrata* package.
- The combination of delta evaluation with minor perturbation in a simulated annealing algorithm leads to faster evaluation times for solutions, thus enabling more solutions to be evaluated in less time than the classical genetic algorithm and grouping genetic algorithm. The simulated annealing algorithm also strikes a balance between exploring good solution neighbourhoods and exploiting them. As more solutions can be evaluated in less time, it enables a greater refinement of this balance. We have also added a feature to increase the number of basic strata moved from one group to another in each iteration in the first sequence, as it can help reduce the number of strata.

- As the evaluation algorithm has an upper limit of 200 iterations, it is possible that it may not have converged within this limit. However, the use of information from the previous solution as a starting point for evaluating the current solution, means it is starting with information closer to the optimum than randomly generated information is likely to be.
- The application of the fuzzy clustering, expectation maximisation, self organising maps, and neural gas clustering techniques in a multi-stage combination with hill climbing, offers alternatives for creating better quality starting solutions and local neighbourhood exploitation to converge on a local minima.
- We also have combined an estimation of distribution algorithm with simulated annealing and delta evaluation to explore the search space while also exploiting good quality solutions, as a hybrid estimation of distribution algorithm.

1.3 Publications

Chapter 7 is based on the following published paper which has been subject to peer review.

1. O'Luing, M., Prestwich, S. and Tarim, S.A. (2019). A grouping genetic algorithm for joint stratification and sample allocation designs. Survey Methodology, Statistics Canada, Catalogue No. 12-001-X, Vol. 45, No. 3. Paper available at http://www.statcan.gc.ca/pub/12-001-x/2019003/article/00007-eng.htm.

Chapter 8 is based on the following published paper which has been subject to peer review.

2. O'Luing, M., Prestwich, S. and Tarim, A. (2022). A simulated annealing algorithm for joint stratification and sample allocation. Survey Methodology, Statistics Canada, Catalogue No. 12-001-X, Vol. 48, No. 1. Paper available at http://www.statcan.gc.ca/pub/12-001-x/2022001/article/00010-eng.htm.

The following is a paper based on Chapter 9 which we hope will be published after it has been evaluated.

1. O'Luing, Mervyn, Steven Prestwich, and S. Armagan Tarim. "Combining clustering algorithms with hill climbing for joint stratification and sample allocation designs."

The following is a paper based on Chapter 10 which we hope will be published after it has been evaluated.

2. O'Luing, Mervyn, Steven Prestwich, and S. Armagan Tarim. "A hybrid estimation of distribution algorithm for joint stratification and sample allocation designs."

Chapter 2

Background and Related Work

2.1 Stratified random sampling

In stratified random sampling, simple random samples are extracted from non-overlapping subpopulations or strata in a population (Cochran, 1953). It is used to achieve savings in costs while obtaining information for each stratum and preserving the accuracy and precision of the target estimates. Samples taken from a stratification must return accurate (or unbiased) estimates and meet precision constraints in order to satisfy these objectives.

Typically, the accuracy of an achieved sample is not measured until the sample data have been collected. Accordingly, there are no such numerical accuracy constraints used in calculating the minimum sample size. Precision constraints, on the other hand, are set by the survey designer in advance, and refer to the preferred upper limits of sampling variance. These are used in the calculation of the sample size and are the main inequalities in the joint stratification and sample allocation minimisation problem.

At the early stages of survey design there was a debate over the validity of any form of sampling from finite populations versus censuses. Kiaër (1899) demonstrated that stratified samples could provide good estimates of finite population totals and means for univariate surveys. Bowley (1906) used Edgeworth's Bayesian version of the Central Limit Theorem (Edgeworth, 1883) to assess the accuracy of estimates made from large random samples drawn from large finite populations (Smith, 1976). Bowley (1906) recommended a stratification that would best minimise the variance of the estimate.

In stratified random sampling we allocate a specified sample size to each

stratum. The sample allocation can be in respect of one (a univariate survey) or more target variables (a multivariate survey).

2.1.1 Stratified sampling and sample allocation in the univariate context

(Neyman, 1934) derived the formula for optimal allocation in stratified sampling where measurements (typically mean or total) are sought for one target variable. This formula is known as "Neyman allocation". Neyman allocation minimizes variance $\mathbb{V}[\bar{X}_n]$ subject to $\sum_{h=1}^{H} n_h = n$, where each n_h is greater than 0.

For each stratum h, simple random sample of size n_h is randomly selected. The sample mean of stratum h is denoted as follows:

$$\bar{X}_h = \frac{1}{n_h} \sum_{i=1}^{n_h} X_{ih} \tag{2.1}$$

, where X_{ih} represents the ith sample value in the hth stratum. The stratified estimate $\bar{X}_n$ of the sampling frame mean, μ, is represented accordingly:

$$\bar{X}_n = \sum_{h=1}^{H} \frac{N_h \bar{X}_h}{N} \tag{2.2}$$

The sample sizes $n_1, \ldots, n_H$ that solve the optimization problem

$$\mathbb{V}[\bar{X}_n] = \sum_{h=1}^{H} N_h^2 \frac{\sigma_h^2}{n_h} \rightarrow min \quad s.t. \sum_{h=1}^{H} n_h = n \tag{2.3}$$

are given by:

$$n_h = n \left(\frac{N_h \times \sigma_h}{\sum_{h=1}^{H} (N_h \times \sigma_h)} \right) \quad h = 1, \ldots, H \tag{2.4}$$

where N_h is the population size in stratum h, n is the sample size, n_h is the sample allocation for stratum h, σ_h is the standard deviation of stratum h (Rice, 2006). The variances may not be known, so estimates are typically used (StatCan, 2003). We utilise an estimate of σ_h in chapter 3, S_h .

In cases where n_h is a rational number, the practice is to round it up to the nearest whole number value, for the purposes of selecting a sample. However, in the results for the experiments reported in subsequent chapters, we report the sample size as it was calculated, regardless of whether it was a whole or rational number, so

that the differences between the sample sizes can be observed.

Neyman allocation is structured so the sample size selected from each stratum is proportional to the size and variation. Larger sample sizes are allocated to larger strata. Sample allocations are either increased or decreased in size depending on the degree of variance within each stratum. The rationale is to survey more units in strata where the data are more dispersed. Likewise strata with more homogeneity require smaller sample sizes.

Neyman's method requires knowledge of the values of stratum variances for the target variables. However, even though we are using a current population register or most recent survey results, the survey population may have changed since the data were originally collected. As a result, these variances are generally unknown. In fact, we are using *a priori* distributions from the population to make predictions regarding the variance. It follows, therefore, that the sample size calculated from such strata variances may be more or less precise than anticipated. Although, we use the term coefficients of variation throughout the thesis, they may also be called the anticipated coefficients of variation (Isaki and Fuller, 1982).

Likewise, the population size and estimates for the key parameters for the target variables, e.g. totals, means and standard deviations, may have changed since the sampling frame was compiled. Indeed, it is realistic to assume (in dynamic populations) that they have changed, even if the change is minor. Accordingly, these parameters should also be viewed as anticipated values. However, given that variance is a measure of similarity, we assume that, even though the values and number of records in the target variables may have changed, they will retain the same degrees of homogeneity, as before. Therefore, when we speak of determining stratifications and sample allocations, we do on the assumption that the coefficients of variation for the achieved sample, will not vary greatly from the anticipated values.

Accordingly, the population size in each stratum N_h, the total population size N, the mean in each stratum $\bar{X}_h$, the population mean $\bar{X}$ and the standard deviation of stratum σ_h as well the resulting calculations, i.e. the sample size in each stratum n_h and total sample size n, should all be considered as estimates of the true values.

The Neyman allocation formula can be adjusted to include the average cost of surveying a unit in each stratum. For example, the formula can be adjusted to include the inverse of the average costs of sampling a unit in each stratum, $\frac{1}{\sqrt{(c_h)}}$. In this way less units are allocated to strata that are more costly to sample. This is known as

"optimal allocation".

$$n_h = n \left(\frac{\frac{(N_h \times \sigma_h)}{\sqrt{(c_h)}}}{\sum_{h=1}^{H} \frac{(N_h \times \sigma_h)}{\sqrt{(c_h)}}} \right) \tag{2.5}$$

where c_h is the cost of surveying one unit in stratum h. Given a fixed budget, or sample size, Neyman's method of optimal allocation divides the sample among the strata in a stratified sample survey. The purpose of this is to minimise the variance for estimating a population mean and total using the standard stratified estimator (Rao, 1977).

Similarly, we can remove the standard deviation in cases where the strata variances are equal. The Neyman allocation formula with variance removed is referred to as "proportional allocation".

$$n_h = n \left(\frac{N_h}{\sum_{h=1}^{H} N_h} \right) \tag{2.6}$$

Although the sample allocation formula is generally attributed to the proof provided by Neyman in 1934, it was subsequently discovered that an earlier proof had been published by (Tschuprow, 1923), and some of the literature refers to it as the Neyman-Tschuprow allocation.

2.1.2 Stratified Sampling and sample allocation in the multivariate context

The issues of stratified sampling and sample allocation to strata for surveys with multiple variable columns were also first discussed by (Neyman, 1934). For such multiobjective surveys, an optimal allocation can be selected for one variable. However, such an allocation may not yield estimates with the same precision levels for the remaining variables.

Since Neyman's method many methods have been proposed to address the problem of sample allocation in multivariate surveys. These methods are divided into two classes: The first class of methods search for an optimal allocation for the weighted average of the stratum variances. The second class of methods relates to allocation for an acceptable coefficient of variation for each of the target variables (Bethel, 1989). Our focus will be on a sample allocation method from the second class of methods in this thesis. We will discuss this further in chapter 3.

However, to continue with our discussion on the background and related work

for this problem, we now present three different approaches for sample allocation in stratified sampling which can be applied in univariate or multivariate surveys.

2.2 Approaches for stratified sampling

A number of methodologies for stratified sampling are prevalent in the literature. Ballin and Barcaroli (2013) classifies them into three approaches. The notation used to introduce each approach below may be not that used in each of the subsequent works which are discussed for that approach.

2.2.1 Determination of sample allocation for a fixed stratification

The first approach seeks to optimise the sample allocation in cases where the strata, H, have already been determined. We assume that a population of N units is divided into H strata, each containing N_h units, $h = 1, 2, \ldots, H$. A random sample of size n_h is to be taken from the h^{th} stratum and the population total $T_g, g = 1, 2, \ldots, G$, for each of G characteristics is to be estimated.

The stratum sample sizes n_h are to be determined so as to minimize the cost of obtaining the sample, subject to the restriction that the variances $V(\hat{T}_g)$ of the estimates do not exceed specified tolerances, V_g (Huddleston et al., 1970).

A linear objective function is assumed and the goal is to minimise:

$$C\left(n_1, \ldots, n_H\right) = C_0 + \sum_{h=1}^{H} C_h n_h \rightarrow min \tag{2.7}$$

subject to the following constraints:

$$\text{VAR}\left(\hat{T}_g\right) = \sum_{h=1}^{H} N_h^2 \left(1 - \frac{n_h}{N_h}\right) \frac{S_{h,g}^2}{n_h} \quad \leq \; V_g, \qquad 0 \leq n_h \leq N_h \tag{2.8}$$

The requirement to have $n_h \geq 2$ (in order to calculate variance) is often included as well.

Many studies have been conducted in this area including the following:

Cochran (1953) (this work was further developed in Cochran (1977)) discussed the determination of univariate and multivariate allocations after strata have already been constructed. The strata are constructed using various approaches, including mathematical (e.g. define stratum boundaries by cumulative root frequency rule

or distribution representing population), geographical (e.g counties and regions), economical (e.g. income level, family size, type of residence) and agricultural (e.g. stratification by farm districts). To determine the univariate sample allocation, Neyman optimal allocation is used. In the multivariate context, a compromise allocation is recommended that is based on the optimal allocation for each variable.

An alternative approach is to consider the problem of multivariate sample allocation as a non-linear one. A number of methods have been implemented to solve this consideration of the problem.

For example, Huddleston et al. (1970) used convex programming methods to determine the optimal allocation for a multivariate stratified sampling design - where there were multiple characteristics of interest and the strata were administrative subdivisions. (Kish, 1976) then discussed optimal and near optimal or "proximal" multipurpose allocations for linear sample designs.

Following this Bethel (1985, 1989) and Chromy (1987) proposed similar convex programming algorithms to solve the (univariate or multivariate) Neyman optimal allocation for a fixed stratification. In this thesis we use an algorithm, which combines the work of Bethel and Chromy, to evaluate the sample allocation for a given stratification. We discuss this in more detail in chapter 3.

Also, again in relation to non-linear stratified sample design problems, Stokes and Plummer (2004) implemented Excel solver tools, where either the sample size or variance is minimised. The solver works well generally but may require some adjustment of parameters such as starting points, model parameters, constraints, etc. in order to succeed.

Then Day (2006) used an evolutionary algorithm to solve the problem. In this case optimal stratum boundaries and optimal allocations given those boundaries are found separately. The optimal stratum boundaries are found using existing methods by Dalenius and Hodges Jr (1959), Singh (1971), Lavallée and Hidiroglou (1988), and Sweet and Sigman (1995).

Day (2006) also indicated that the method could be developed, to solve for optimal stratum boundaries and multivariate optimal allocations to those strata, simultaneously. Furthermore, the algorithm could be expanded to solve for allocations other than minimizing variance or cost, such as inclusion of a sufficient number of units with a particular condition.

Díaz-García and Cortez (2008) subsequently considered the problem as that of a nonlinear matrix optimisation of integers, for which they apply a number of

techniques to solve the optimal allocation. The choice of method requires experience with the problem and methods, however they provide some guidance to help in this selection.

Kozak et al. (2008) then introduced multivariate stratified sampling in domains; where a population has been subdivided into domains and each domain comprises a number of strata.

After this, Day (2010) followed their earlier work with an evolutionary algorithm, which extends the basics of the single-objective evolutionary algorithm, to search for the optimum combination of multiple objectives (e.g. cost and variance). The evolutionary algorithm used was the strength pareto evolutionary algorithm, with the evolutionary process entailing a balance of recombination and mutation.

2.2.2 Two-stage determination of stratification and sample allocation

Research has also been conducted on two-stage sampling which optimises strata initially, and subsequently addresses sample allocation, in the survey design. Optimum stratification in survey sampling was first discussed by Dalenius (1950), who used a scenario in which the stratification variable was identical to the target variable, in order to estimate an optimal stratification for Neyman optimal allocation. We use notation from (Dalenius, 1950) to introduce this research.

In this scenario, the population, π, is represented by the distribution function $f(y)$, and $a \leq y \leq b$ is to be divided into k mutually exclusive strata by $k-1$ points of stratification $(Y_0 = a), Y_1, Y_2 \ldots Y_i \ldots Y_{k-l}, (Y_k = b)$ in the range a, b. The mean of this population is estimated by

$$\bar{y} = \sum_i p_i \bar{y}_i \tag{2.9}$$

after a random sample of n observations n_i selected from stratum π_i, where $i = 1 \ldots k$. If $n_i = p_i n$ then $V(\bar{y})$, the variance of $\bar{y}$, is given by

$$V(\bar{y}) = \frac{1}{n}(\sum_i p_i \sigma_i)^2 \tag{2.10}$$

The problem seeks to find the set of points that makes $V(\bar{y})$ a minimum. This same set of points will also make $\sum_i p_i \sigma_i$ a minimum.

Subsequent work in this area includes Dalenius and Hodges Jr (1959), who

provided a method for choosing near optimum stratification points, which equalize the integrals over the various strata of the square root of the population density.

Sethi (1963) then provided an algorithm for stratifying a population into take-some strata (where a sample of less than the total number of records is taken), with a focus on estimating the population means of the normal and the set of chi-square distributions. Later, Singh (1971) suggested a new cumulative root frequency method for dividing a stratification variable into strata, requiring prior knowledge of the stratification and target variables.

After this Thomsen (1977) studied the effect of using two stratifying variables, instead of one stratifying variable, on the quality of estimates for one study variable. The evidence suggests that, there were greater gains in efficiency from using two stratifying variables over one.

Schneeberger and Pöllot (1985) then represented the problem of finding the optimum stratification points of a two-way stratification, in the case of proportional and optimum allocation, as a nonlinear programming one. The dependence of the objective function on the correlation coefficient, ρ, and the sampling fraction, $q = \frac{n}{N}$, was investigated. With proportional allocation, for greater values of ρ (called the ρ effect) and $q = 0$, they found an optimum stratification point. On the other hand, with optimum allocation, an optimum stratification point can be found with lower values of q and higher values of ρ. However, greater values of q resulted in a multitude of stratification points, this being dubbed the q effect.

This was followed by Hidiroglou (1986), who considered the stratification of a population into two strata: a take-all (where a sample of all records is taken) stratum and a take-some stratum so as to minimize the overall sample size, assuming simple random sampling without replacement in the take-some stratum. This type of stratification is particularly useful for populations, whose distribution exhibits a marked skew to the right, with a few large units and many small units (e.g. business surveys).

Lavallée and Hidiroglou (1988) subsequently combined a modified version of Sethi (1963)'s algorithm with Hidiroglou (1986)'s procedures to determine stratum boundaries, which split the population into a take-all stratum and a number of take-some strata. For the take-some strata, a simple random sample without replacement was selected, with power allocation (Bankier, 1988) being implemented instead of Neyman optimal allocation. Power allocation is implemented where there is a requirement for precision estimates for each stratum, rather than the whole sample. A disadvantage of Neyman optimal allocation being that the coefficients of variation

for each stratum may vary.

Then, Rizvi et al. (2000) proposed a cumulative root frequency rule to efficiently define stratum boundaries, after the number of strata and sample allocation method have been defined. After this, Briggs et al. (2000) introduced a combined heuristic and systematic local search approach to stratify with two variables, reporting gains in efficiency of 20% over stratification with one variable.

Following this, Lu and Sitter (2002) proposed a method for two-way stratification using linear programming, for situations where strata can be formed from the cross-classification of the categories in two stratifying variables, and the sample allocated to each stratum. This process can be extend to stratified multi-stage sampling designs.

Lednicki and Wieczorkowski (2003) then outlined an algorithm for selecting stratified samples in subpopulations (or domains), where in each subpopulation there are take-some and take-all strata, and coefficients of variation are obtained. The algorithm used a simplex local search method (Nelder and Mead, 1965), which was later shown to be not as efficient as a random search method by Kozak (2004).

Following this, Gunning and Horgan (2004) presented an algorithm for constructing strata, for the purposes of obtaining equal coefficients of variation in each stratum derived from positively skewed populations. The algorithm obtains stratum breaks using the geometric distribution. However, it has a limitation in that it is unable to determine a take-all stratum.

Then, Kozak and Singh (2007) applied an evolutionary algorithm on the bivariate stratification problem also considered by Lednicki and Wieczorkowski (2003). They found that the algorithm achieves an efficient stratification, and claim that it can be applied in a multivariate context also.

After this, Khan et al. (2008) formulated the problem of constructing stratum boundaries and stratum widths as a mathematical programming problem (MPP), which sought to minimise variance, so that the sum of the widths of all the strata is equal to the total range of the distribution. MPPs were formulated for triangular and standard normal distributions, and solved with Bühler and Deutler (1975)'s dynamic programming approach.

Subsequently, Reddy et al. (2016) used dynamic programming to construct strata where there are a number of stratification variables available. The proposed method is applicable to any skewed population, and led to improved efficiency when compared to the Geometric method of Gunning and Horgan (2004), the cumulative

root frequency method of Dalenius and Hodges Jr (1959) or the Lavallée and Hidiroglou (1988) method combined with with Kozak (2004)'s algorithm.

2.2.3 Joint determination of stratification and sample allocation

Finally, there is research into jointly stratifying and optimising sample size, i.e. in one step, where the attained sample size is a measure of the quality of the stratification. In this scenario, the population of N records is subdivided into H disjoint strata. The sample allocation, for which the coefficient of variation (CV) of the estimated population total, $\hat{T}$, of each target variable, g, is less than or equal to the precision constraints, U_g, is a measure of the quality of each stratification.

The goal is to find the best quality H, where quality is expressed by n, the optimal allocation, i.e. $\min \rightarrow n = \sum_{h=1}^{H} n_h$ subject to $CV\left(\hat{T}_g\right) \leq U_g$, where $g = 1, \ldots, G$, n_h is the sample allocation for each stratum, h and $h = 1, \ldots, H$.

Kozak et al. (2007) investigated scenarios of univariate and multivariate stratification, with an objective of finding the set of strata boundaries within domains that minimizes an overall sample size, with respect to different target levels of precision for different characteristics. This approach included the construction of take-all and take-some strata. They considered the simplex and random methods used by Kozak (2004), along with the method proposed by Lavallée and Hidiroglou (1988). They recommended their method for approximate optimisation in the case of univariate stratification, however multivariate stratification was more complex, and required further work.

Later that year, Keskintürk and Er (2007) used a genetic algorithm (GA) in stratum boundary determination and sample size allocation, where sample size and the number of strata were already decided. The performance of the GA on four methods of sample allocation (equal, proportional, Neyman, and GA allocation), was compared with Gunning and Horgan (2004)'s geometric and Dalenius and Hodges Jr (1959)'s cumulative square root of the frequency methods of stratification. The best results were obtained when both stratum boundaries and strata sample size allocations are determined by the GA.

After this, Benedetti et al. (2008) used a hierarchical divisive algorithm to jointly partition basic strata, or "atoms", created by combining all possible quantitative or qualitative classes in the stratification variables, into strata, and calculating the minimum sample size that would meet the precision constraints for that stratification.

Test results indicated gains in efficiency for a given sample size over the previous design of the Italian Farm Structure Survey. However, the stopping rules used in their method - as well as the algorithm - do not guarantee an optimum stratification of atoms. Indeed, the search for optimality is conditioned by the top-down path taken by the algorithm (i.e. where it splits the population at each step).

Subsequently, Baillargeon and Rivest (2009) compared Sethi (1963)'s, Gunning and Horgan (2004)'s and Kozak (2004)'s algorithms. In a single-step, these algorithms determine the optimal stratum boundaries for a univariate stratification - using proportional, Neyman, and power allocations. The optimisation can take into account varying non-response rates within strata, as well as the relationship between the stratification variable and the survey variable.

The optimisation also considers three types of strata: take-all, take-some and take-none (which puts units such as those with a $'0'$ value into a stratum from which no sample is drawn, and leads to biased estimates, but may reduce the sample size needed to meet a pre-specified relative root mean squared error (RRMSE)- used to measure precision as a proxy for the coefficient of variation). Kozak (2004)'s algorithm was found to be the best tool for determining strata boundaries that minimise the sample size.

Baillargeon and Rivest (2011) then followed with the R package *stratification*, which implements a number of methods. These include the cumulative root frequency method, the geometric method, Lavallée and Hidiroglou (1988)'s stratification algorithm, and Baillargeon and Rivest (2009)'s take-none stratum method. The package can be used for constructing strata, determining stratum sample sizes, and calculating the precision of the estimator of the population mean of some survey variable related to the stratification variable.

Following on from Benedetti et al. (2008)'s work, Ballin and Barcaroli (2013) considered the case of qualitative stratification variables which are correlated to the target survey variables, and the basic strata, referred to as "atomic strata", created from their cross-product. We will discuss basic strata and their atomic and continuous forms in more detail in section 2.2.4 below.

Each stratification of atomic strata is a solution which is evaluated by the Bethel-Chromy algorithm. They use a genetic algorithm combined with a set of heuristics to evolve solutions over a set number of iterations. Results of the algorithm show much better sample efficiency, when compared to benchmark samples for the same surveys. The performance of the algorithm is limited by the degree of correlation between the auxiliary stratification variables and the target variables - a

weaker correlation means less reliable sample and precision estimates. The same point could be made for the algorithms presented in this thesis as they also use the existing auxiliary and target variables to find solutions.

They also cited another limit of this approach, in the requirement to transform continuous auxiliary variables into discrete ones. Furthermore, after transformation some of the continuous variables may be characterized by non contiguous values requiring constraints on the generation of candidate solutions. Ballin and Barcaroli (2020) followed with an updated genetic algorithm implementing the grouping operators recommended in chapter 7, and suggesting the use of an initial solution created by a greedy algorithm to help shorten the search. They also proposed a classical genetic algorithm for dealing with the case of continuous auxiliary variables.

This thesis presents research aimed at further developing their work, by exploring alternative metaheuristics and machine learning techniques for joint stratification and sample allocation in survey design (on both discrete and continuous stratification variables). We support our claims by comparing computational results on publicly-available test data.

2.2.4 Basic strata: atomic and continuous

We now provide more background details on basic strata. In the context of multivariate stratification, it is not uncommon for survey survey designers to stratify by the variables they wish to report statistics on (whether in the design phase or in post-stratification). For example, in the design of household surveys Tepping et al. (1943) considers (amongst other stratifications) geographical groups broken into 16 cells (4 size classes $\times$ 4 rent classes). Alternatively, in the analysis phase, Engel et al. (1971), in relation to a study on child work and social class, considers distributions by grade and social class, or occupation by social class.

In business surveys, in addition to a take-all stratum, take-some stratifications often combine geographical location and industry with unit size, in order to ensure an adequate sample representation. In such surveys, stratification by size measures may be particularly applicable, where the aim is to estimate means or totals of several variables, each highly correlated with one of the size measures (Skinner et al., 1994).

Similarly, in agricultural surveys strata can be constructed by combining geographical information with farming-type (pastoral, arable, mixed) and farm size (e.g. acreage under crops, livestock, etc.) Ballin et al. (2016).

On a more general level, where several variables are both available and desirable for stratification, and furthermore there are a number of target variables in the survey, it is poor practice to try to tailor sample designs to the simple bivariate theory of one stratifier for one study variable (Kish and Anderson, 1978).

Indeed, the more detailed the stratification the more efficient the estimates will be but, on the other hand, there is a higher risk of a unit being re-allocated from one stratum to another between one survey and the next. This is due to changes in values for it in at least one of the stratification variables.

Additionally, if L_i is the number of classes (or distinct values in the case of a continuous variable) from the ith variable, the total number of usable strata is around $L = \Pi_i L_i$. Clearly, the number of stratifying variables as well as the number of classes within will have a bearing on the number of strata created. In such cases of detailed stratification, there may well be some strata (or cells) which are empty.

Furthermore, Benedetti et al. (2008) state that if the corresponding number of such basic strata, or atoms, exceeds a given threshold imposed by practical restrictions, it seems unavoidable to redesign the survey selecting a smaller number of stratifying variables or creating fewer classes from each of them. They reason that another way of dealing with this situation is in stratifying the atoms. Indeed, in cases of practical necessity it is not uncommon for survey designers to combine smaller strata into larger ones.

Ballin and Barcaroli (2013) refer to basic strata constructed from a subset of qualitative stratification variables, which are correlated to the target variables, as atomic strata, and note that when the number of atomic strata is high, this could lead to a costly and inefficient sample size. They advocate choosing the optimal arrangement of the atomic strata *in* strata. In Ballin and Barcaroli (2020), they also search for the optimal stratification of continuous strata (constructed from continuous auxiliary variable).

As we refer often to Ballin and Barcaroli (2013) and Ballin and Barcaroli (2020)'s work, we retain the descriptions to these two types of basic strata, as atomic strata (in the case of discrete variables) and continuous strata (in the case of continuous variables), throughout the remainder of the thesis.

We also advise that our algorithms (see chapters 7, 8, 9 and 10) were developed using atomic strata, and for this reason we refer to atomic strata more often than continuous strata. It was after their development that we tested the implementation of the algorithms, in chapters 8, 9 and 10, on the case of continuous strata.

2.3 Other relevant literature

Danish et al. (2017) reviews existing methods for obtaining optimum strata boundaries using mathematical programming, such as gradient methods, gradient projection methods, dynamic programming techniques, a genetic algorithm, the Lagrange multiplier technique, and include comparisons with mathematical approaches such as the cumulative root frequency method and geometric method. They conclude by indicating that there are increasing trends in using mathematical programming approaches rather than classical methods to obtain optimum strata boundaries.

Kashan et al. (2013) present a particle swarm optimization algorithm for grouping problems and compare it with a GGA on the various instances of the single batch–machine scheduling problem and bin packing problem. Their results indicate that their algorithm compares well with the GGA. Thus, they recommend it as an efficient solver the wide class of grouping problems.

Ramos-Figueroa et al. (2020) review recent metaheuristics developed for the solution of grouping problems. They compare the performance of three metaheuristic algorithms, namely a grouping genetic algorithm with controlled genes transmission, a genetic algorithm with extended permutation solution encoding and particle swarm optimization with machine-based encoding encoding on different instances for the parallel-machine scheduling problem with unrelated machines. Their experimental results suggest that the GA with the group-based encoding is the best option to address this problem.

The results of their experiments suggest that the performances of genetic algorithms using a representation scheme based on groups and suitable variation operators. Furthermore, their study suggests that a genetic algorithm with the group-based representation can attain better results than particle swarm optimization with the machine-based encoding.

They add that more study is needed to analyze the relationship between features of grouping problems and the performance of the algorithms used to solve them.

Chapter 3

Problem description

3.1 Cartesian Product

As we saw in Chapter 2, this problem requires a population data set in which there are N records. For each of these records, there are variables in the G target variable columns, i.e., those which are of interest to the survey, as well as the auxiliary/stratification variable columns (variable columns are referred to in machine learning as features or attributes, and also are often referred to as variables for short).

A subset containing the M most correlated auxiliary variables to the target variables is selected. The selection process can include expert knowledge of the data, combining geographical and key characteristics relevant to the survey (e.g. age group and property type in a household survey, or industry and enterprise size in a business survey), or some other selection procedure, such as using a correlation matrix.

The selection of correlated variables is particularly useful in data sets where there are a large number of auxiliary variables. It has a practical implication on the computational burden of calculating the Cartesian product and subsequent memory requirements, as well as the anticipated efficiency of the target variables. The auxiliary variables can be discrete or continuous. We discuss various approaches for selecting correlated variables in chapter 5.

In the case of atomic strata, continuous variables can be converted to discrete variables using the k-means clustering technique. This reduces the amount of atomic strata created by the Cartesian product of these variables. (Although the illustration here refers to atomic strata, the Cartesian product and subsequent aggregation can also be applied to continuous strata. However, the Cartesian product will mainly

Table 3.1.1: *Synthetic data set demonstrating target (Y) , auxiliary (X) and domain variable columns*

$Y1$	$Y2$	$X1$	$X2$	$DOMAIN$
57,324	1,314	18	3	2
32,429	6,007	17	1	1
28,161	5,034	17	1	1
31,223	2,344	19	1	2
24,321	1,632	19	1	2
45,621	6,756	19	1	2

create single sized continuous strata.) The auxiliary variables are described as follows:

$$X_m \ (m = 1, \ldots, M) \tag{3.1}$$

where X_m is an auxiliary variable (and M, mentioned above, is the total number of auxiliary variables).

The Cartesian product is used to combine the values of the auxiliary variables together for each domain (e.g. region, district, gender, social group, etc.). For example, let us consider a synthetic data set with two target variables, Y_1 and Y_2 , and two auxiliary variables, X_1 and X_2. There is one domain variable with two values "1" and "2". Domains usually represent geographical, categorical or administrative sub-units, and are independent of each other. In the minimisation problem, we find the solution for each domain and add the sample sizes together to get the total sample size. There are six records in total.

Each value in an auxiliary variable is crossed with all the values of another auxiliary variable. For example, the two auxiliary variables X_1 and X_2 have the values 17 and 1 respectively. These values could be used to make one atomic stratum called $(17, 1)$. All corresponding values of $(17, 1)$ in Y_1 and Y_2 are then aggregated to estimate means and standard deviations. In other words, we get the means and variances of the Y variable values in the atomic strata. The combinations of X_1 and X_2, i.e. those for which there are corresponding values in Y_1 and Y_2, are listed here:

$$(18, 3), (17, 1), (19, 1) \tag{3.2}$$

The Cartesian product is:

$$CP = X_1 \ \times \ X_2 \tag{3.3}$$

The Cartesian product results in the set of L atomic strata.

$$l = \{1, 2, \ldots, L\} \tag{3.4}$$

Combinations such as $(18, 1)$ or $(17, 3)$ are excluded because there are no values in Y_1 and Y_2. These create empty atomic strata, $\emptyset$, which cannot be used. Table 3.1.2 below presents the results of this procedure.

Table 3.1.2: *Atomic strata created from cross-product of auxiliary variables in synthetic data set*

Atomic Stratum, l	N_1	$\bar{Y}_1$	$\bar{Y}_2$	S_1	S_2	X_1	X_2	*Domain*
17, 1	2	30,295.0	5,520.5	2,134.0	486.5	17	1	1
18, 3	1	57,324.0	1,314.0	0	0	18	3	2
19, 1	3	33,721.7	3,577.3	8,873.4	2,266.4	19	1	2

where: there are G (g=1,...,G) different target variable columns, and $\bar{Y}_{l,g}$ is a variable mean calculated for each variable in each atomic stratum l. Likewise $S_{l,g}$ refers to the sampling frame standard deviations (population estimates) of each variable in each atomic stratum l.

Atomic stratum $(18, 3)$ has no standard deviation because it contains only one unit. Typically constraints are set that the minimum stratum size be 2 units in order that variance can be estimated.

This example uses a bivariate scenario, but the process is also applicable to larger multivariate scenarios. The Cartesian product of the auxiliary variables in a multivariate scenario is:we present a grouping genetic

$$CP = X_1 \times X_2 \times \ldots \times X_M \tag{3.5}$$

An example of atomic strata resulting from a Cartesian product in a multivariate scenario using the iris data set is included in chapter 7.

As described above, each constituent Y unit is aggregated to provide summary statistics for the atomic stratum. However, as also mentioned, there are some atomic strata which do not have any Y values and these are null sets, $\emptyset$, that cannot be used. The number of possible atomic strata is, formally, described as follows:

$$L = \prod_{m=1}^{M} l_m - I^* \tag{3.6}$$

where L is the number of atomic strata, and I^* is the number of impossible or absent combinations of values created by the Cartesian product of the M auxiliary variables.

After the creation of atomic strata the next step is to search for the partitioning/stratification that enables the global minimum sample size for the precision constraints. The set of all possible partitionings P of the L atomic strata is defined by $\{P_1, P_2, \ldots, P_B\}$.

Each partitioning is a candidate solution. The set of possible partitionings is known as the search space, and, if it was feasible in terms of time and cost, the full list of candidate solutions may be evaluated in order to identify the optimum solution. The Bell number representing the number of possible partitionings (stratifications) of a set of atomic strata grows very rapidly with the number of atomic strata and is calculated by the Bell formula Graham et al. (1989):

$$B_L = \sum_{i=0}^{L-1} \binom{L-1}{i} . B_i \quad (B_0 = 1) \tag{3.7}$$

As already mentioned, the optimum solution may be found by evaluating each stratification. However, given that the set of partitionings grows exponentially with the set of atomic strata, there comes a point where even for a moderate number of atomic strata and the most powerful computers, there are no known efficient algorithms to solve the problem.

Various heuristics and algorithms are used as they may find an optimum or near-optimal solution without evaluating all possible solutions. We apply them in more detail in the thesis. We base our outline of the problem on the descriptions provided by (Ballin and Barcaroli, 2013; Benedetti et al., 2008; Bethel, 1989).

3.2 Objective Function

The set of L atomic strata is partitioned into H subsets called strata. N_h is the population size for each stratum h. $S^2_{h,g}$ are the estimates of the variances of the G different target variables Y in each stratum h, where $h = 1, \ldots, H$ and $g = 1, \ldots, G$. We assume a simple random sampling of n_h units without replacement in each stratum.

The estimator of the population total $\left(\hat{T}\right)$ is:

$$\hat{T}_g = \sum_{h=1}^{H} N_h \bar{Y}_{h,g}; (g = 1, \ldots, G) \tag{3.8}$$

where:

$\bar{Y}_{h,g}$ is the mean of the G different target variables Y in each stratum h

The variance of the estimator of the total $\left(\hat{T}\right)$ is given by:

$$\text{VAR}\left(\hat{T}_g\right) = \sum_{h=1}^{H} N_h^2 \left(1 - \frac{n_h}{N_h}\right) \frac{S_{h,g}^2}{n_h} \quad (g = 1, \ldots, G) \tag{3.9}$$

The upper limit of expected variance or precision U_g is expressed as a coefficient of variation CV for each $\hat{T}_g$ (which normalises the measures of variability in the different target variables and means that the variance can be compared relatively) :

$$CV\left(\hat{T}_g\right) = \frac{\sqrt{\text{VAR}(\hat{T}_g)}}{\hat{T}_g} \leq U_g \tag{3.10}$$

where U_g is a constant and a measure for the expected sampling variance or precision for target variable g expressed as a coefficient of variation. These values are usually set by the survey designer in advance, using either existing data or survey requirements as a guide.

The objective function can be defined as:

$$C\left(n_1, \ldots, n_H\right) = C_0 + \sum_{h=1}^{H} C_h n_h \tag{3.11}$$

where C_0 is the fixed cost and C_h is the average cost of interviewing one unit in stratum h and n_h is the number of units, or sample, allocated to stratum h.

In our analysis we take the view that C_0 and C_h are constant in the linear cost model and both can be utilised after the minimum sample size is found. Therefore, C_0 is set to 0, and C_h is set to 1 for all h. However, the objective function (and consequently the algorithms we propose in this thesis - as sample size is calculated by the Bethel-Chromy algorithm) can also handle scenarios where C_h are non-equal and greater than 1. For such scenarios, we estimate the sample cost as opposed to the sample size.

The problem is consequently summarised as follow:

$$\begin{cases} \min \rightarrow n = \sum_{h=1}^{H} n_h \\ \\ CV\left(\hat{T}_g\right) \leq U_g \end{cases} \tag{3.12}$$

3.3 The allocation model

As mentioned above, each stratification is a solution, the quality (or sample size) of which is evaluated by the Bethel-Chromy algorithm. More details on the Bethel-Chromy algorithm are provided in section 3.5. The algorithm uses Lagrangian multipliers and gradient descent to search for a minimum to the above linear objective function with the convex constraints provided in equation 3.12. To solve this convex optimisation problem Bethel (1985, 1989) proposes scaling the sample allocation where x_h is expressed as a function of n_h:

$$\begin{cases} x_h = \frac{1}{n_h} & \text{if } n_h \geq 1 \\ \quad = \infty & \text{otherwise} \end{cases} \tag{3.13}$$

After the above scaling of the sample allocation (and for the purposes of the Bethel-Chromy algorithm, i.e. the objective function otherwise remains unchanged - we are still minimising n), the objective function can be updated accordingly

$$\left\{ f(x) = \sum_{h=1}^{H} \frac{1}{x_h}, \quad h = 1, 2, ..., H \right. \tag{3.14}$$

The coefficient of variation in equation 3.10 for variable g can be squared so that the variance inequality can be re-expressed as follows:

$$CV\left(\hat{T}_g\right)^2 = \frac{\text{VAR}\left(\hat{T}_g\right)}{\hat{T}_g^2} \leq U_g^2$$

We can then bring $\hat{T}_g^2$ to the right hand side by cross multiplication.

$$\text{VAR}\left(\hat{T}_g\right) \leq \hat{T}_g^2 U_g^2 \tag{3.15}$$

We can then substitute the left side of equation 3.15 with equation 3.9.

$$\sum_{h=1}^{H} N_h^2 \left(1 - \frac{n_h}{N_h}\right) \frac{S_{h,g}^2}{n_h} \leq \hat{T}_g^2 U_g^2 \tag{3.16}$$

Equation 3.16 in turn can be written as

$$\sum_{h=1}^{H} \frac{(N_h S^2{}_{h,g})}{n_h} - N_h S^2{}_{h,g} \leq \hat{T}_g^2 U_g^2$$

Through incorporating equation 3.13 this becomes

$$\sum_{h=1}^{H} (N_h S^2{}_{h,g}) x_h - N_h S^2{}_{h,g} \leq \hat{T}_g^2 U_g^2$$

or alternatively,

$$\sum_{h=1}^{H} \frac{(N_h S^2{}_{h,g}) x_h}{(\hat{T}_g^2 U_g^2 + \sum_{h=1}^{H} N_h S^2{}_{h,g})} \leq 1 \tag{3.17}$$

Then we substitute $\frac{N_h S^2{}_{h,g}}{(\hat{T}_g U_g^2 + \sum_{h=1}^{H} N_h S^2{}_{h,g})}$ with $a_{h,g}$ and replace the problem summary with the following:

$$\begin{aligned} &\min f(x) = \sum_{h=1}^{H} \tfrac{1}{x_h} \\ &a_g' x \ \leq 1 \quad (g = 1, \ldots, G) \end{aligned} \tag{3.18}$$

where x is the vector of all $x_h > 0$, and a_h is the hth row vector and a_g is the gth column vector of matrix A = $a_{h,g}$. The Bethel-Chromy algorithm uses Lagrangian multipliers to derive a solution for each n_h.

$$\frac{1}{n_h} = \begin{cases} \frac{1}{(\sqrt{\sum_{g=1}^{G} \alpha^* a_{h,g}} \sum_{h=1}^{H} \sqrt{\sum_{g=1}^{G} \alpha^* a_{h,g}})} \text{ if } \sum_{g=1}^{G} \alpha^* a_{h,g} > 0 \\ +\infty \text{ otherwise} \end{cases} \tag{3.19}$$

where $\alpha_g^* = \frac{\lambda_g}{\sum_{g=1}^{G} \lambda_g}$ and λ is the Lagrangian multiplier of the constraint on the maximum error allowed in estimating the g^{th} surveyed variable. The is explained in more detail in section 3.4 below.

3.4 The optimal allocation

We continue with notation from Bethel (1989) and with reference to Boyd et al. (2004), the optimum allocation (denoted by the vector x^*) in the case of a single variable is expressed as follows:

$$\begin{cases} x_h^* = \frac{1}{(\sqrt{\sum_{g=1}^{G} a_{h,g}} \sum_{h=1}^{H} \sqrt{\sum_{g=1}^{G} a_{h,g}})} \text{ if } \sum_{g=1}^{G} a_{h,g} > 0 \\ +\infty \text{ otherwise} \end{cases} \tag{3.20}$$

where $g = 1$ and x^* is the minimum of $f(x)$ subject to $\sum_{h=1}^{H} a_{h,g} x_h \leq 1$

The objective function is strictly convex for $x > 0$ and constraints are linear, so that this is a convex programming problem. Equation 3.20 can be extended to solve cases where $g > 1$ using Lagrangian multipliers, where let x^* and λ^* are primal and dual optimal points with zero duality gap, and satisfy the the Kuhn Tucker theorem conditions, namely:

- The primal constraints are ≤ 0, indicatively

$$a_g' x^* - 1 \leq 0 \tag{3.21}$$

- The dual constraints λ_g are ≥ 0.
- There is complementary slackness:

$$\lambda_g \left(a_g{}' x^* - 1 \right) = 0 \tag{3.22}$$

- The gradient of the Lagrangian with respect to x vanishes, i.e. is 0 when $x = x^*$

Accordingly:

$$\nabla f(x^*) + \sum_{g=1}^{G} \lambda_g a_g = 0 \tag{3.23}$$

where ∇ denotes the gradient and $g = 1, 2, ..., G$.

If $x > 0$, $\sum_{g=1}^{G} \lambda_g a_g' x \leq \sum_{g=1}^{G} \lambda_g$, then, by combining equations 3.22 and 3.23

$$-x' \nabla f(x^*) = \sum_{g=1}^{G} \lambda_g a_g' x \leq \sum_{g=1}^{G} \lambda_g = \sum_{g=1}^{G} \lambda_g a_g' x^* = -x^{*\prime} \nabla f(x^*) \tag{3.24}$$

where the symbol $'$ is used to denote the transpose.

As $f(x)$ is a differentiable convex function

$$f(x) - f(x^*) \geq (x - x^*)' \nabla f(x^*) \tag{3.25}$$

$$\textit{(for all } x > 0 \textit{ and } x' > 0\textit{)} \tag{3.26}$$

and by applying equation 3.24 we get

$$f(x) - f(x^*) \geq (x - x^*)' \nabla f(x^*) \geq 0 \tag{3.27}$$

From this we can deduce that x^* is the minimum of $f(x)$ subject to the following conditions:

$$\sum_{g=1}^{G} \lambda_g a_g{}' x \leq \sum_{g=1}^{G} \lambda_g \text{ for } \text{ all } x > 0 \tag{3.28}$$

We can add positive multiplicative constants to the constraints without affecting f, thus x^* also minimises $f(x)$ subject to the following $\sum_{g=1}^{G} \alpha_g^* a_g{}' x \leq 1$ where $x > 0$, and $\alpha_g^* = \frac{\lambda_g}{\sum_{g=1}^{G} \lambda_g}$.

We can now extend equation 3.20 into a multivariate optimum allocation formula by applying the weighted sum $\sum_{g=1}^{G} \alpha_g^* a_g$.

$$\begin{cases} x_h^* = \frac{1}{\left(\sqrt{\sum_{g=1}^{G} \alpha_g^* a_{h,g}} \sum_{h=1}^{H} \sqrt{\sum_{g=1}^{G} \alpha_g^* a_{h,g}}\right)} & \text{if } \sum_{g=1}^{G} \alpha_g^* a_{h,g} > 0, \ 1 \leq h \leq H \\ \\ = \infty & \text{otherwise} \end{cases} \tag{3.29}$$

This formula (equation 3.29) requires knowledge of the α_g^* values in order to calculate x^*, and it is used by Bethel-Chromy algorithm which uses gradient descent to converge on α_g^* values and thus obtain x^*.

3.5 The Bethel-Chromy algorithm

The Bethel algorithm uses the Kuhn Tucker theorem and Lagrangian multipliers and is proven to converge quickly on the optimum allocation, if it exists. The Chromy algorithm, which also uses the same conditions or Cauchy-Schwarz solutions, is simpler than the Bethel algorithm and has evidence of good convergence but this has not been proven. Nonetheless, there is good empirical evidence suggests that it

has good convergence properties (Bethel, 1989).

However, its notation can be adapted to fit-within the operation of the Bethel algorithm and the two have been combined into the *bethel.r* function (Barcaroli et al., 2014). It is also known as the Bethel-Chromy algorithm, and is discussed frequently in this thesis. It is used to estimate the minimum sample size for a fixed stratification with regard to a set of variance constraints for each variable. This Bethel-Chromy combination runs for a default maximum of 200 iterations, and it may not have converged on an optimum solution within that limit.

In order to describe the Bethel-Chromy algorithm mathematically we begin by saying that it is used to find α_g^* and x^* by searching over the weighted averages $\sum_{g=1}^{G} \alpha_g \xi_g$.

For each g variable there is an α_g coefficient or value. These are represented by the vector $\alpha = (\alpha_1, \alpha_2, ..., \alpha_G)'$. We use α to calculate the estimate of the optimum allocation $\tilde{x}(\alpha)$.

$$\begin{cases} \tilde{x}_h(\alpha) = \frac{1}{\left(\sqrt{\sum_{g=1}^{G} \alpha_g^* \xi_{h,g}} \sum_{h=1}^{H} \sqrt{\sum_{g=1}^{G} \alpha_g^* \xi_{h,g}}\right)} & if \quad \sum_{g=1}^{G} \alpha_g^* \xi_{h,g} > 0, \ \ 1 \leq h \leq H \\ \\ = \infty & otherwise \end{cases} \tag{3.30}$$

By solving α^* in $\tilde{x}(\alpha^*)$ we get the optimum allocation, i.e. x^*. The iterative algorithm for solving α^* and thus x^* is outlined below.

Algorithm 1 Bethel-Chromy algorithm as it is implemented in *bethel.r*

1: Set $\alpha_j^{(0)} = \frac{1}{G}$ and iter $= 0$
2: iter $\leftarrow$ iter $+1$;
3: Calculate $\tilde{x}(\alpha^*)$ as follows:

$$\tilde{x}(\alpha^*) = \frac{1}{\left(\sqrt{\sum_{g=1}^{G} \alpha_g^{(iter)} \xi_{h,g}} \sum_{h=1}^{H} \sqrt{\sum_{g=1}^{G} \alpha_g^{(iter)} \xi_{h,g}} + \epsilon\right)}$$

4: Calculate the α vector values for the next iteration $\alpha_g^{(iter+1)}$, as follows:

$$\alpha_g^{(iter+1)} = \frac{\alpha_g^{(iter)} (\xi_g' \tilde{x}(\alpha^{iter}))^2}{\sum_{g=1}^{G} \alpha_g^{(iter)} (\xi_g' \tilde{x}(\alpha^{iter}))^2}$$

5: while $max(abs(\alpha(iter) - \alpha(iter-1))) > \epsilon$ OR iter < maxiter repeat steps 2-4, otherwise STOP

Solutions (stratifications) are evaluated to estimate the minimum sample size necessary to meet the precision constraints using the *bethel.r* function.

3.6 Finite correction factor

In the Bethel-Chromy algorithm, as throughout this thesis, the finite population correction factor has been ignored (with the assumption that the sample sizes are less than 5% of each finite populations). Nonetheless, to adjust for the finite population correction factor the Bethel-Chromy algorithm may be updated by adding the FPC term, $\sum_{h=1}^{H} W_h^2 \frac{S_{h,g}^2}{N_h}$ (Bethel, 1989).

Chapter 4

Machine learning and metaheuristics

4.1 Machine learning

Even though, Turing (1950) referred to learning machines, the term machine learning was coined by Samuel (1959), who defined it as a "[f]ield of study that gives computers the capability to learn without being explicitly programmed". Samuel (1959) describes machine learning as the study of making machines more human-like in their behaviour and decisions by giving them the ability to learn and develop their own programs. This is done with minimum human intervention, i.e. no explicit programming. The learning process is automated and improved based on the experiences of the machines throughout the process. Tom (1997) puts it as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

(Theobald, 2017) indicates that although Samuel (1959) didn't explicitly indicate it in the above definition, a key characteristic of machine learning is the concept of self-learning. This implies the application of statistical modeling to detect patterns and improve performance based on data and empirical information; all without direct programming commands.

However, machines can not learn without following the code instructions set by the programmer. In doing so, machines apply algorithmic techniques on input data to find a solution to a given problem. Such solutions are predictions made by a model, using parameter values which are learned at various steps within the operation of the algorithm. Regular computer programs, on the other hand, follow a direct input command to produce a pre-defined or expected output.

To develop this in more detail, machine learning builds a decision model

from the input data, using probabilistic reasoning, trial and error, and other computationally-intensive techniques to interpret relationships and patterns in the data (and use these to compute the predictions). This means that the output of the decision model is determined by the contents of the input data (in this thesis, the input data are target and auxiliary variables subdivided into basic strata). Therefore, the output is not found by any pre-set rules defined by a human programmer, although the learning and decisions occur by following the pre-set code.

Nonetheless, the human programmer remains responsible for choosing and inputting the data into the model, selecting an appropriate algorithm and tweaking its settings (called hyperparameters) in a bid to reduce prediction error. In this thesis, reducing prediction error translates to either finding the optimal solution, or near-optimal solutions. Although, we don't measure the distance of the solution from the optimal solution, as we usually don't know the optimal solution in advance. Instead, we measure improvements in solution quality by comparing new solutions with existing solutions. this how we can tell if we are reducing prediction error.

For scenarios, where we know the solution in advance recall that with smaller problems it is possible to evaluate each possible solution and thus find the optimal solution. In those cases, algorithms are of interest (the same applies with larger problems also) if they can find the optimal solution in a faster time than complete enumeration.

Note, as these are computationally-intensive techniques, the tweaking of hyperparameters can significantly affect the efficiency of an algorithm, and this necessitates hyperparameter optimisation, which we will discuss in more detail below.

Durmus (2022) describes three main types of machine learning algorithms.

- Supervised learning algorithms. The purpose of supervised learning algorithms as to use known values of feature variables (or input variables) to predict (or estimate) the value of a predefined target variable (or output variable). The premise, then, is that the values for the target variable are known, or labelled. In the learning process, a model of the relationship between the feature and target variables using a subset of the input data (called training data) is developed. This model is tested on the unseen remainder of the labelled data, before being applied to unseen and unlabelled data to infer the most likely value of the target variable. Supervised learning is implemented on two types of problems:
    1. Regression problems, where the target variable of interest is continuous.

2. Classification problems, where the target of interest is a categorical variable.

- Unsupervised learning algorithms. These algorithms identify patterns and relationships in data, without using a predefined relationship of interest (i.e. the techniques they use do not rely on labelled training data as a means of developing a decision model). As they do not use training data, and thus must find useful patterns in the data, they are more exploratory in nature. However, as the evaluation of outputs guides this exploration, results are not necessarily less meaningful than those found in supervised learning, or reinforcement learning. This approach can be used to solve:

  1. clustering problems, which involves grouping units of observations based on similarities and dissimilarities between them.
  2. association analysis, where the goal is to identify salient relationships among variables within a data set.
  3. dimensionality reduction reduces the number of input variables to a manageable size while also preserving the data integrity.

  The problem described in this thesis is a clustering problem, although the objective function differs to the standard measurements of similarity.

- Reinforcement learning. In reinforcement learning, an "agent" explores its environment, and in doing so determines an optimal action or sequence of steps (the goal of interest) in response to that environment. As with unsupervised learning, it does not rely on labelled data. Instead, the learning process uses a reward function in which the agent strives to maximise its reward by iteratively improving through a trial and error approach. Agents maximise their cumulative return by learning from their experiences through feedback as they move through their environment. Reinforcement learning is relevant for problem types that sequential or dynamic in nature, such as robotics or games.

In addition to above main types of machine algorithms, there is semi-supervised learning, which is a hybrid form of unsupervised and supervised learning. It is used in cases where there are a number of datasets that contain a mix of labelled and unlabelled variables. Semi-supervised learning attempts to leverage unlabelled cases to improve the reliability of the prediction model. This approach can use supervised learning to build a model to use on unsupervised cases in the same dataset. The model can be further retrained using additional datasets, or by additionally newly labelled cases. There is no guarantee, however, that a semi-supervised model will

outperform a model trained exclusively on the original labelled cases.

4.2 Metaheuristics

The discussion in Hao and Solnon (2020) provides the basis for this descrption. Computer Scientists use metaheuristics to solve complex and challenging combinatorial search problems. The search space of combinations (candidate solutions) in these problems is usually exponential, and as we scale up, there is a realisation that it is, currently and may always be, impossible to evaluate every candidate solution. For this reason, algorithms are designed that can intelligently explore this combination space, and find good quality solutions for this hard problem within a reasonable amount of time. The metaheuristic algorithms outlined in this thesis use generic methods to solve this problem.

There are three main approaches to solving combinatorial problems:

- Exact approaches systematically explore the search space, until a solution is found or an inconsistency in the data is found. These approaches use pruning to reduce the search space and ordering heuristics to define the order in which the search space is explored. These techniques are not suitable for solving larger problem instances within a reasonable amount of time.
- Heuristic approaches use trial and error methods or various short-cuts to find solutions which are good enough in a time that is small enough. These approaches may not find the optimal solution, because the full space of solutions is not searched in problem instances of moderate sizes and upwards, and there is no other method to prove optimality. Nonetheless, near-optimal (or optimal) solutions can be found in polynomial time.
- Approximation approaches search for approximate solutions that can be guaranteed to be within a certain distance of the optimum solution. The factor α is a metric for this distance, and if the algorithm can find solutions with a factor of α, for every instance of the problem, it is referred to as an α-approximation algorithm.

4.2.1 Perturbative metaheuristics

These approaches explore the search space by iteratively perturbing either individual or groups of combinations, to generate new combinations. Perturbation amounts to minor or major stochastic modifications to previously generated solutions, in

anticipation of improved solutions. The degree to which these perturbations are conducted are said to be problem instance-based (Zlochin et al., 2004), hence the need for hyperparameter optimisation steps discussed in chapter 6. Genetic algorithms and local search are two perturbative approaches used in this thesis.

4.2.1.1 Genetic algorithms

Genetic algorithms (John, 1975) are nature-inspired evolutionary algorithms which combine natural selection, reproduction and mutation to iteratively evolve solutions towards optimality.

- Natural selection implies the retention of elite solutions from a population of candidate solutions that are best suited (fit) to the problem instance.
- Reproduction by cross-over implies the recombination of two parent solutions, using a crossover operator, to generate new solutions that retain genes from both parents, and hopefully be a more suitable fit (better quality solutions).
- Mutation implies randomly changes to the allelles of genes in a solution allowing changes and possible improvements in solution quality that combined with natural selection and reproduction, feed into future generations of solutions.

Algorithm 2 provides a basic outline of a genetic algorithm, and this is followed by an explanation of the main steps below.

Algorithm 2 Genetic algorithm

```
1: Initialise a population of solutions
2: while stopping criteria not met do
3:     Select solutions from the population
4:     Generate new solutions by recombination and mutation
5:     Update the population
6: end while
7: Return the best solution found
```

- Initialisation of the population: A population of candidate solutions is either randomly generated with respect to a uniform distribution of parameters, or the population from a previous run of the algorithm can be used as a starting point.
- Selection: The selection of two solutions (parents) for recombination is usually stochastic and favours the more elite parents, however with lower probabilities inferior quality solutions can also selected.
- Recombination (cross-over): Generate new solutions by recombining parents.

The goal is facilitate better exploration of the search space, while both retaining the qualities of the parents but also allowing diversification in the solution structure.

- Mutation: After cross-over, in this step solutions are modified by randomly selecting genes and selecting new values (alleles) from within the range of possible values for those genes.

4.2.1.2 Local search

In local search an initial solution is iteratively modified to a neighbourhood combination, by random perturbations. Improvements in the solution quality are always retained (so that the best found solution is not lost), however in some local search methods, eg. simulated annealing, dis-improvements in solution quality are also accepted in order to allow the algorithm explore new solution states. Local Search may be viewed as a very particular case of genetic algorithm, where the population has a size of one solution, and the solution is modified by mutation.

Algorithm 3 describes the local search approach below.

Algorithm 3 Local search

```
1: Generate an initial solution, either randomly or procedurally
2: while stopping criteria not met do
3:     Modify solution
4:     Retain modification if it leads to improvement in solution quality
5: end while
6: Return the best solution found
```

4.2.2 Constructive metaheuristics

These approaches iteratively add or update each solution combination component until there is a complete solution. They use a model (which are usually stochastic in nature) to choose the next component to be added or updated in that solution, for each component in an iteration. There may be more than one solution to be updated in each iteration, such as in population based methods. There are a number of greedy randomised strategies to choose what components are added or updated, at each iteration, such as estimation of distribution algorithms or ant colony optimisation. However, we will focus on estimation of distribution algorithms, as one is utilised later in the thesis.

4.2.2.1 Estimation of distribution algorithms

Estimation of distribution algorithms (Larrañaga and Lozano, 2001) are greedy randomised algorithms, which select promising solutions from a population (similar to elitism in evolutionary algorithms) and build probabilistic models of those solutions. They then sample from the corresponding probability distributions, for each component in a solution, to obtain new solutions for a given number of solutions in a population (Lima et al., 2011).

Algorithm 4 describes a generic estimation of distribution algorithm, with the key elements described in more detail below.

Algorithm 4 Estimation of distribution algorithm

```
1: Generate an initial population of solutions.
2: while stopping criteria not met do
3:     Use population to construct a probabilistic model
4:     Use model to generate new solutions
5:     Update population with these new solutions
6: end while
7: Return the best solution found
```

- Generation of the initial population: The population can either be generated randomly, or be a combination of randomly generated solutions and previous evaluated good quality solution(s), or previously evaluated good quality solutions.
- Construction of a probabilistic model: A number of different kinds of probabilistic models can be considered. In this thesis, we construct a probability model of the distribution of each component of the best solutions, and this defines the probability of each component being selected to update a component, for each component in a solution. We repeat this step to generate of a set of new solutions to add to the best solutions and re-complete the population.

4.2.3 Memetic algorithms/hybridisation

These hybrid algorithms combine population-based evolutionary algorithms with local search, to take benefits of the explorative properties (or diversification abilities) of population based algorithms, with the exploitative (intensification) properties of local search. Structurally memetic algorithms resemble a genetic algorithm which has been extended with a local search step.

Typically, selection and replacement mechanisms are used to determine the solutions that are recombined and those that are eliminated. However, the mutation operator is replaced by a local search process. In the hybrid estimation of distribution algorithm we propose in chapter 10, we combine the elitism and mutation aspects of a genetic algorithm, with local search. However, we remove the recombination step.

4.2.4 Inputs and outputs

The above algorithms require input data in order to generate outputs (solutions). Likewise, and unless otherwise specified, for the algorithms presented in chapters 7, 8, 9 and 10 the input data are the basic strata and the outputs are the solutions (stratifications).

Chapter 5

Sourcing data and preprocessing

5.1 Sourcing data

This problem requires the use of an accessible, real world, structured, relevant and pre-processed sampling frame. If a sampling frame is structured the data are numerical or categorical. Unstructured data refers to emails/text files, media files, websites, social media, mobile data, etc. and these data will need to be transformed into structured formats before they can be used by the algorithms in this thesis. Typically, sampling frames vary in size, and range from the very small to the very large.

If sampling frames are very small, e.g. (Anderson, 1935; Fisher, 1936), then machine learning algorithms can be expected to find the optimal solution in a small amount of time. However, the run times for the algorithms quickly escalate, as sampling frames get larger.

Nonetheless, while smaller sampling frames are useful in demonstrating the functionality of the algorithms, in order to be meaningful for real world problems sampling frames need to be of at least a moderate size and the target variables need to be of interest, while it is also important to have a subset of correlated auxiliary variables, in order for it to relevant to the survey designer.

Obtaining real world data sets can be challenging. They can be gathered in traditional data collection means e.g. surveys, interviews, focus groups, data measurements, etc., web scraping, e.g. house prices, jobs data, or purchased from a data provider. All can be time-consuming and costly tasks to complete. The data need to be of good quality, and this may be labour intensive and costly.

Also, with real world data there are issues of storage, access control, privacy

and security, retention policies, etc. to be considered. Those sourcing and using the data have the responsibility to ensure that they are used ethically, fairly and accurately. These safeguards also need to be addressed when sourcing data. In this section we address the issue of data sourcing. We discuss the pre-processing stage in section 5.2 below.

5.1.1 Open data

However, these logistic concerns can be mitigated by the sourcing of open data. Open data and content can be freely used, modified, and shared by anyone for any purpose (Foundation, 2015). These are made available by governmental bodies, publicly funded research and academic institutions and private enterprises. They are free, without formal control restrictions, such as copyrights or patents. They are rich, detailed data, however there can be issues regarding file formats, metadata, accessibility and periodicity and they may require pre-processing prior to use.

Open data providers include Government datasets, Kaggle, IPUMS, Amazon, UCI Machine Learning Repository, Google and Microsoft. This comprises a mix of publicly funded and privately funded entities. We have sourced data from (Anderson, 1935; Barcaroli et al., 2014; Fisher, 1936; Kiva, 2018; R Core Team, 2015; Ruggles et al., 2017; US Census Bureau, 2016). These are of good quality and there was very little, if any, preprocessing required.

5.1.2 Synthetic data

The basic idea behind the creation of synthetic data sets is the replacement of some or all of the observed values by sampling from appropriate probability distributions, in this way preserving the key statistical information (Nowok et al., 2016). This approach has developed along similar lines as imputation, which replaces missing values with model values or measures of central tendency but can distort the original distribution of values for a variable (StatCan, 2003). However, synthetic data differs in that they are generated from an assumption of knowledge of the full population parameters.

Synthetic data also enables protection of sensitive data. This is because data generated are new data which are intended to have the same characteristics of the original real data. They produce the same results and retain the same predictive properties. As the data are new, privacy and security concerns are eliminated. However, care should be taken to ensure that the original data cannot be recreated from the new data, or identifiable information has not been retained.

Broadly speaking, there are three types of synthetic data categories (Singh, 2021):

- Fully Synthetic Data. These data are completely synthetic, and do not have any of the original values from the real data. Instead, the data generator uses the parameters for variables in the real data to randomly generate data for the equivalent variables in the new data. Techniques for generating synthetic data include bootstrap methods and multiple imputations. There are no privacy concerns with fully synthetic data, although they retain key properties of the original data.
- Partially Synthetic Data. In this case, only values of some selected variables are replaced with synthetic values. Generally, speaking this process protects against disclosure of key information and preserve privacy in the new data. This technique is also useful for imputing missing values.
- Hybrid Synthetic Data. In this scenario, the newly generated data are combination of records from real data with records from synthetic data. As such it has the advantage of containing real data as well as the synthetic data. It also helps to preserve privacy. However, this method is more computation intensive than the above two.

Synthetic data creation techniques that preserve privacy include anonymization along with simple perturbation methods such as aggregation, recoding, record-swapping, suppression of sensitive values or adding random noise. We can also generate entirely new datasets, or randomly perturb the values of the existing target and auxiliary variables. These new variables may have different distributions to the original variables.

Machine learning techniques for generating synthetic data include classification and regression trees (CART), bagging and support vector machines (SVMs). SVMs are suitable for predicting the values of categorical variables. They can also be tuned for continuous variables, and in this case the process is called support vector regression. CARTs approximate the conditional distribution of a univariate outcome from multiple predictors. Random forests are collections of CARTs, and bagging is a predecessor of random forests. They can preserve relationships reasonably well while providing low disclosure risks. These methods require minimal tuning, can incorporate numerical, categorical, and mixed variables as predictors, is suitable for high dimensions, and can fit non-linear relationships (Caiola and Reiter, 2010; Drechsler and Reiter, 2011).

Although, the above synthetic data approaches focus on faithfulness to the

original data, the newly generated data do not need to have the same properties as the original data. Instead, in order to be useful in experiments the auxiliary variables should be correlated to the target variables, and the resulting distributions have a practical resemblance to real world scenarios.

5.2 Data preprocessing

Data quality is defined in terms of accuracy, completeness, consistency, timeliness, believability, and interpretabilty. These qualities are assessed based on the intended use of the data (Han et al., 2011). Preprocessing is the resolution of low quality data, i.e. those which contain errors, are incomplete, inconsistent, or lacking in certain characteristics or trends, through several techniques including data cleaning, data integration, data reduction and data transformation.

5.2.1 Data cleaning

In order to run effective machine learning algorithms, the data must first be clean. The reason for this is that the outputs depend on the input data, and if the data contain errors, then, clearly, this will affect calculations and inferences made from these data. From the perspective of the algorithms in the coming sections, good quality solutions which, hopefully, translate to collection of precise target variables (as the solution is made prior to the survey), depend on both good quality target and auxiliary variables.

Simply put, data cleaning refers to error detection and error repairing. It is generally cheaper to clean these errors in advance of data processing, than it is afterwards (Ahmed and Aziz, 2010). Examples of common errors include missing values, typos, mixed formats, replicated entries of or for the same real-world entity, and violations of business rules (Chu et al., 2016). It has become easier to acquire and store large amounts of data, more regularly than before. However, in the context of this data volume, velocity and variety, it follows that data veracity is critical to making meaningful decisions and analysis from that data. Therefore, data cleaning is a central topic not merely for datasets, but data warehouses as well.

This is especially true with respects to sampling frames which nowadays are as much likely to be compiled from a combination of administrative or other cleaned datasets, as they are population lists. The achieved sample arising from samples selected from sampling frames could lead to biased and imprecise estimates, or a binary combination of these, whereas we would be aiming for unbiased and precise estimates.

In the sections that follow we outline a number of techniques for error detection and error repairing at the pre-decision making and/or pre-analysis stages of the data. In order to detect data that requires cleaning a good understanding of the data is necessary. This is usually achieved with experience of designing the survey, gathering and/or using the data, reviewing the existing metadata and documentation describing the data (or creating these). The detection process involves identifying incomplete, inaccurate, irrelevant, corrupt or incorrectly formatted data. Errors can occur in either qualitative (categorical) or quantitative variables. We first deal with error detection in qualitative variables, and then quantitative variables. We have used Han et al. (2011) as a base to guide the following discussion.

5.2.2 Error detection

Errors occur in the process of data collection. Such errors occur when the data is being collected, such as misunderstandings of the questions asked, or too many possible responses, inputting error, recording and system errors, or coding/classification errors. Errors arise as a result of a decayed sampling frame, e.g. where address details have changed since data for the frame were originally collected. Errors may also occur when data sets with conflicting information for variables in the same observation are matched.

A good starting point for error detection is to use the metadata for the data, if they are available. Such metadata provide information on what values to expect in qualitative and quantitative variables, e.g. what categories, codes or frequencies to expect for a qualitative variable or whether the data might be skewed or normally distributed in a quantitative variable. An expected mean, median or mode for a quantitative variable will help to identify potential bias or outliers. Metadata is also useful for knowing the format and names for each variable. However, in the absence of metadata it may be useful to use data scrubbing tools which implement domain knowledge to detect and correct errors, data auditing tools which analyse the data to detect errors, and/or a manual review, again to detect errors.

5.2.3 Missing values

Missing Values occur due to item or unit non-response in a survey, or more generally speaking, where values for a variable or variables for a given observation are missing in data collected through other means. Methods for dealing with missing values include:

1. Ignore the observation, or delete the observation. This is practical in cases

where there are a number of values missing in a given observation, or in large data sets where there is an insignificant number of observations with missing values and the assumption is that their removal will not materially affect the distributions within that data set. It is not effective in scenarios where there is a sufficient number observations deleted. This could materially impact the accuracy of a machine learning algorithm (e.g. in classification or clustering).

2. Fill in the missing values manually. This is a useful approach where the manually inputted values are exactly the same or in close proximity to the missing values. However, the approach is time consuming and is impractical in cases where the approach is applied to a large number of observations.
3. Use a global constant to substitute for missing values. In qualitative variables this entails replacing missing values by a description such as "Missing" or "Unknown". In numeric variables missing values are indicated by the numerical programmatic equivalent of 'Missing', e.g. '.' or 'NA', or substituted with a '0'. In qualitative variables the algorithm could take the substitution to be a valid classification and thus predict that certain values will be such. This is further misleading as such a classification also groups otherwise dissimilar objects together. For numerical variables, the substitution will affect predictions arising from the use of such edited variables.
4. Use the mean or median to replace a missing value. The use of a central tendancy to replace a missing value with the middle value of a distribution (of a population or stratum) is an approach that can be taken. In normally distributed data the mean can be used, whereas in skewed distributions the median can be applied. In distributions where a large number of means or medians are used to replace missing values, this can bias the updated means or medians and totals away from their true values.
5. Use most probable value to fill missing values. Such values are found by techniques utilising regression, Bayesian inference, or decision trees. Such techniques rely on the accuracy of the existing data to estimate the missing values, and may bias the data. However, this approach is more popular as it does not rely on the distribution of the variable with the missing values, but related variables which are fully populated.
6. Imputation. Algorithms like support vector machine (SVM) or K-nearest neighbor (KNN) use information for other fully populated variables to predict missing values in a variable. Multivariate Imputation by Chained Equations involves multiple cycles of imputing missing values in one variable at a time

using regression - for each variable with missing values in a cycle, using the updated information for the other variables to impute missing values again for one variable at a time in the next cycle. The number of cycles is a hyperparameter, that needs to be set in advance.

In some cases, missing values may not be an error in the data, it may be because for legitimate reasons a value is not available for a given observation. For example, a person cannot supply a student number if they have not been registered as a student. An algorithm will require instructions on how to proceed in these cases, in particular with respect to how the missing values in such cases should be substituted, and how such substitutions could impact the key outputs.

In the ideal circumstance, there would not be any missing values. However, is less than ideal circumstances, the number of missing values would be kept to a minimum. In the data experiments below, observations with missing values were deleted from data sets (in cases what that arose), under the assumption the small scale of deletions did not materially impact the quality of the sampling frame.

5.2.4 Noisy data

Noisy data describes the presence of random errors or variance in data measurements. Basic statistical techniques such as data summaries, box plots, or scatter plots can be used to identify outliers, which may be noise. There are some techniques which can be implemented to smooth out such noise.

1. Binning. This relates to sorting the data and partitioning the data into bins of equal size, and replacing each value of the bin by the mean or median. This is also smoothing by bin boundaries which entails replacing each value by either the maximum or minimum value, depending on which it is closest to. Instead of being equal size bins may also be of equal width, where the range of values in each bin is constant. Binning may also be used to discretize data.
2. Regression. The linear regression between two variables, or multiple regression in the multivariate context, describes the relationship between variables, where an independent variable or independent variables are used to predict a dependent variable. Using regression as a smoothing technique does not account for the errors in the data, instead predicting the expected value of the dependent variable.
3. Outlier analysis. By clustering data, e.g. k-means, db-scan, etc., data values which fall outside of clusters of similar values may be considered outliers.

These groupings of similar values are summarised by their means (or medians). Calculating the z-score of a data value (in a Gaussian distribution), gives a measure of how far it is from the mean, and thus provides an indication of whether or not it is an outlier.

5.2.5 Data integration

The merging of data from multiple data sources can help to reduce the redundancies and inconsistencies in the resulting data set. These data sources can include multiple databases (each containing multiple data sets), data cubes (multi-dimensional databases) or flat files (relational databases). We explore a number of issues related to data integration, below. However, as our focus is on sampling frames utilising an existing population list, or which can be compiled from the merging of administrative data records in a relational database using common identifiers, we will confine our discussion to these areas.

5.2.6 Entity integration

Matching entities from multiple data sources can be tricky, when it is not initially clear that they are equivalent. For example in administrative data sets, which tend, generally, to have been compiled and maintained by multiple domain experts for record purposes rather than for machine learning, the same identifier variable might have a different name in different data sets. For example, in tax records, what is denoted as the *person_id* variable in one data set might be called *p_id* in another data set. Furthermore, in the same data set or another data set again, *CustomerNumber*, although conceptually equivalent, *might or might not* be a different identifier to the *person_id*.

In other examples, a qualitative variable might have categorical values in one data set, but these have been converted to numeric values (which may have a numeric or character format) in a quantitative variable on another data set. Likewise, time and date variables can be recorded in different formats. Similarly, addresses can come in one variable, or in separate variables, and the details and syntax of the address differ. A variable might be in character format in one data set and numeric format on another, or it may be in character format on both data sets, but with variables of different format lengths (causing possible truncation in the matching step).

In these scenarios, metadata will help clarify the name, meaning, data type, and range of values permitted for a variable, and also clarify the rules for handling blank, zero, or null values. In the absence of metadata, a manual investigation of the

variables concerned might clarify their similarity.

However, care should be taken in such cases, as there might be a good reason for the difference in variable names, in that there are important nuances between variables. For example, *CustomerNumber* might refer to active customers in a given data set, and while it might be the same as the *person_id* variable which also relates to inactive customers, using *CustomerNumber* variable to match variables from this data set with another data set, should only be applied if the intention is to compile a data set with active customer information only.

Finally, care should be taken when matching data from different time periods, that data from a more out of date time period do not over-write those from a more current one. In particular this can occur with numeric variables, such as *amount_paid*, which may be specific to the time period in question, more so than categorical variables such as *address*, which tend not to be as susceptible to change (although they too may change over time).

5.2.7 Redundancy and correlation analysis

When integrating data sets, and more generally speaking, variables may be redundant if they can be derived from another variable, or the same variable exists under different names. Furthermore, variables may be closely correlated. Correlation analysis is implemented to measure how strongly one variable implies another, using the available information. For categorical variables a chi-squared (χ^2) test is used, and for quantitative variables a correlation coefficient or covariance is used to measure this similarity.

5.2.8 Chi-squared (χ^2) test for correlation

This test measures the correlation between two qualitative attributes X and Y, where each variable contains distinct values used to separate the group the characteristics of that variable into classes. We base this summary on the description provided by Han et al. (2011). Suppose X has c distinct values, namely $x_1, x_2, ...x_c$. Y has r distinct values, namely $y_1, y_2, ...y_r$. . Let (X_i, Y_j) denote the joint event that attribute X takes on value x_i and attribute Y takes on value y_j, that is, where $(X = x_i, Y = y_j)$. We count every possible (X_i, Y_j) joint event. The χ^2 value, or *Pearson* χ^2 *statistic (Pearson, 1900)* is calculated as follows:

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{5.1}$$

where o_{ij} is the observed frequency (i.e., actual count) of the joint event (X_i, Y_j) and e_{ij} is the expected frequency of (X_i, Y_j), which can be computed as:

$$e_{ij} = \frac{count(X = x_i) \times count(Y = y_j)}{n}, \tag{5.2}$$

where n is the number of rows, $count(X = x_i)$ is the number of rows having value x_i for X, and $count(Y = y_j)$ is the number of tuples having value y_j for Y. The sum in Eq. 5.3 is computed over all of the $r \times c$ cells. Note that the cells that contribute the most to the χ^2 value are those for which the actual count is very different from that expected.

The χ^2 statistic tests the hypothesis that X and Y are independent, that is, there is no correlation between them. The test is based on a significance level, with $(r \times c)$ degrees of freedom. If the hypothesis can be rejected, then we say that X and Y are statistically correlated. The χ^2 co-efficient for a representative sample in an asymptotically unbiased estimator for the population χ^2 co-efficient. The χ^2 statistic can have a value of greater than 1, is affected by sample size and its p-value varies as a function of sample size (Allen, 2017).

5.2.9 Cramér's V and Tschuprow's T

Cramér's V is based on the *Pearson* χ^2 *statistic* and can be used for categorical data which have been converted into discrete numerical values (Cramér, 1999).

As above the chi-squared statistic then is:

$$\chi^2 = \sum_{i=1}^{c} \sum_{j=1}^{r} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{5.3}$$

Cramér's V is computed by taking the square root of the chi-squared statistic divided by the sample size and the minimum dimension minus 1:

$$V = \sqrt{\frac{\chi^2/n}{\min(k-1, r-1)}} = \sqrt{\frac{\varphi^2}{\min(k-1, r-1)}} \tag{5.4}$$

where:

φ is the phi coefficient, k is the number of columns and r is the number of rows. This measure of association ranges between 0 and 1, with 0 indicating no association and 1 indicating perfect association.

Tschuprow's T (Tschuprow, 1939; Tschuprow and Tschuprow, 1925) closely resembles Cramér's V, and both range from 0 to 1, although Tschuprow's T can only equal 1 in cases where there is a perfect association in a square cross-classification table (of the variables):

$$T = \sqrt{\frac{\varphi^2}{(k-1, r-1)}} \tag{5.5}$$

However, Cramér's V and Tschuprow's T can be heavily biased estimators of the population statistic, in finite (and smaller) sample sizes. They may also overestimate the strength of association. Bergsma (2013) provides the following bias correction:

$$\tilde{V} = \sqrt{\frac{\tilde{\varphi}^2}{\min(\tilde{k}-1, \tilde{r}-1)}} \tag{5.6}$$

where

$$\tilde{\varphi}^2 = \max\left(0, \varphi^2 - \frac{(k-1)(r-1)}{n-1}\right) \tag{5.7}$$

and

$$\tilde{k} = k - \frac{(k-1)^2}{n-1}, \tilde{r} = r - \frac{(r-1)^2}{n-1} \tag{5.8}$$

Then $\tilde{V}$ estimates the same population statistic as Cramér's V, however the mean squared error tends to be lower.

Correlation Coefficient for numeric data

If X and Y are numbers the correlation coefficient (also known as Pearson's product moment coefficient (Pearson, 1907)) can be used to evaluate the correlation between the two variables. The coefficient is calculated as follows:

$$r_{XY} = \frac{\sum_i x_i y_i - n\bar{X}\bar{Y}}{\sqrt{\sum_i x_i^2 - n\bar{X}^2}\sqrt{\sum_i y_i^2 - n\bar{Y}^2}} \tag{5.9}$$

where n is the number of rows, x_i and y_i are the respective values of X and Y in row i, $\bar{X}$ and $\bar{Y}$ are the respective mean values of X and Y. Note that $-1 \leq r_{X,Y} \leq 1$,

where a negative value between 0 and -1 indicates a negative correlation, and a positive value between 0 and 1 indicates a positive correlation. A value of 0 indicates no correlation, i.e. the variables are independent. Note: correlation between X and Y does not imply that X causes Y or vice versa, and the correlation may only exist in the data set under study.

5.2.10 Covariance of Numeric Data

The correlation coefficient and covariance are similar measures for determining if two attributes are related (whether positively or negatively). As before we have two numeric variables X and Y, in a data set with n rows. The means, or expected values, of X and Y are calculated as follows:

$$E(X) = \bar{X} = \frac{\sum_{i=1}^{n} x_i}{n} \text{ and } E(Y) = \bar{Y} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{5.10}$$

The covariance between X and Y is defined as follows;

$$Cov(X,Y) = E((X - \bar{X})(Y - \bar{Y})) = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{n} \tag{5.11}$$

Note the similarities with equation 5.9. If X and Y tend to change together the covariance is positive. If they tend to change in opposite directions, the covariance is negative. If X and Y are independent, the covariance is 0.

5.2.11 Scatter plots

Scatter plots are useful to visualise the relationship or lack of relationship between numeric variables, in cases where visual observation of the relationship suffices for the selection of redundant variables.

5.2.12 Correlation coefficient for numerical and categorical data

The Spearman correlation coefficient (Spearman, 1904), r_s, is an adaptation of the Pearson product moment coefficient and assesses the strength and direction of a relationship between two ranked variables.

For a sample of size n, the n raw scores X_i, Y_i are ranked (numerically, i.e. after the categorical variable is converted to numerically) $\mathrm{R}(X_i), \mathrm{R}(Y_i)$, and r_s is calculated as follows:

$$r_s = \rho_{\mathrm{R}(X),\mathrm{R}(Y)} = \frac{\mathrm{cov}(\mathrm{R}(X),\mathrm{R}(Y))}{\sigma_{\mathrm{R}(X)}\sigma_{\mathrm{R}(Y)}}, \tag{5.12}$$

where ρ is the Pearson correlation coefficient of the ranked variables, and this is calculated with $\mathrm{cov}(\mathrm{R}(X),\mathrm{R}(Y))$ being the covariance of the ranked variables and $\sigma_{\mathrm{R}(X)}$ and $\sigma_{\mathrm{R}(Y)}$ being the standard deviations of the ranked variables.

5.2.13 Row duplication

Where duplicate rows occur, one row should be deleted to avoid situations where information for the same observation is include in the computations more than once. In some cases, duplication can occur, but the rows are not exactly equivalent (due to some variations at the variable level). In these scenarios, the most accurate row should be retained.

5.2.14 Data value conflicts

In the same way, data values for the same variables from different data sets, may differ at the observation level in representation, scaling, or encoding. Such values should be transformed so that they are consistent with those of the variable(s) under study, and the most up-to-date values retained.

5.2.15 Data reduction

Data reduction techniques essentially a larger dataset into one smaller in volume, by summarising and retaining the key information so that the integrity of the original data is retained. Examples of data reduction strategies are dimensionality reduction, and data compression.

5.2.16 Dimensionality reduction

In dimensionality reduction, we reduce the number of variables under consideration. Examples of dimensionality reduction methods comprise principal component analysis and variable subset selection.

5.2.17 Principal component analysis

Principal component analysis is similar to variable subset selection, in that it reduces the variable set size into a subset, however it does so by combining the key

information of these variables into a smaller subset of variables. The smaller set can then be mapped back to the initial data set. Also, this subset can be more easily visualised than the initial variables, especially in multivariate scenarios. Briefly put, the process runs as follows:

- Compute k orthonormal vectors (principal components) from the input data (which has been normalised beforehand). The input data can then be computed as a linear combination of these principal components.
- These principal components are then sorted in order by their decreasing variance, so that the top sorted components have the highest variance and this enables elimination of the lower variance components as they contribute little to approximating the input data.

Principal component analysis can be used on ordered and unordered variables. It can also be applied to sparse data and skewed data. It can also be used in multiple regression analysis and in clustering, e.g. to generate starting solutions for the algorithms considered later in this thesis.

5.2.18 Variable subset selection in general

The aim of variable subset selection (which in machine learning is also called feature subset selection) is to remove unnecessary, irrelevant, or redundant variables, being left with a set of variables for which the probability distribution of their data classes best resembles the original distribution obtained using all the variables. In large data sets, it makes sense to select a representative subset of variables. For machine learning algorithms, this reduces the complexity of the input data, search space and the resulting solutions. A number of greedy heuristic methods are used to find a locally optimum variable subset selection. These are:

- Stepwise forward selection: In this procedure we start with an empty set of variables and iteratively add the best of the remaining variable to the set using tests of statistical significance.
- Stepwise backward elimination: This procedure is the reverse of the above method. We start with the full set and iteratively remove the worst of the remaining variables.
- Combination of forward selection and backward elimination: Again an iterative procedure, in which the best variable is selected and the worst variable from the remaining variables is removed, before beginning the iteration by selecting the next best variable.

- Decision tree induction: This procedure uses decision tests, to construct a tree of the best variables.

5.2.19 Data compression and variable subset selection for this problem

Data compression refers to a reduced representation of the original data. It can either be *lossless*, which means that the data can be reconstructed from the compressed data, or *lossy* which an approximation of the original data can be reconstructed.

5.2.20 Selection of auxiliary variables

It makes sense that we should find the auxiliary variables that are most correlated or are the best predictors of the target variables. This selection may be completed using expert knowledge of the data and survey requirements, univariate and multivariate regression, the correlation and covariance coefficients outlined above or we could use the F-test (to test the linear relationship and compare standard deviations of variables) or Mutual Information Criterion, to compare the information one variable provides about another in linear and non linear relationships.

5.2.21 Data transformation

As the name suggests, in data transformation the data are transformed into data more suitable to machine learning, than the original form. Data transformation techniques include the following:

- Smoothing, using techniques such as binning, regression, and clustering to remove noise from the data.
- Variable construction (or feature construction), where new variables are added from the given set of variables to help the machine learning process. For, a number of values could be added together to give a total, or address variables could be concatenated into one address string.
- Aggregation, where the data is summarised by aggregation. The creation of basic strata, as discussed in chapter 3, is an example of data aggregations, where the means and standard deviations of variable summarise the data points in each basic stratum and can be used to calculate population totals.
- Normalization, where the variables are scaled so as to fall within a smaller range, e.g. -1.0 to 1.0 or 0.0 to 1.0

- Discretisation, where numerical are grouped into discrete interval labels or categorical labels.

5.2.22 Qualitative variables

Data variables which contain strings need to be converted to numerical variables, in order to make sense to a machine learning algorithm. Here are two methods utilised for that purpose.

- Label encoding. This is where we convert categories in a string to unique numbers, each number representing a category.
- One-Hot Encoding. One variable of categories is split into multiple columns, where the number of columns equates the number of categories in that variable. In this new arrangement, each column is now filled with 1's and 0's, 1 indicates that the observation in that column is in that category, whereas 0 indicates that it is not.

Both approaches have the advantage of converting categorical strings into a format accessible by an algorithm. Label encoding also reduces the amount of memory required to store information for a given variable, however the use of numbers to encode categories may imply an ordering , ranking or arithmetic (e.g. sum or average) of those categories, that does not make sense.

One-Hot Encoding does not have this issue, as each variable column only contains a binary representation of a category. Nonetheless, as the number of categorical variables increases, the dimensionality of the one-hot encoding also increases. In this cases, principal component analysis could be applied to help in dimensionality reduction.

Chapter 6

Hyperparameters

In machine learning algorithms, there are parameters and hyperparameters. Parameters are those which are tuned in the learning process, and these are often denoted by symbols such as α or λ. We do not have any direct control over these parameters, although hyperparameters affect the rate at which they are tuned, in the running of an algorithm.

Hyperparameters are set manually or automatedly in advance of the algorithm, and they refer to controllable parameters such as the number of iterations the algorithm runs for, the number of solutions evaluated in a given iteration (e.g. grouping genetic algorithm), or the rate at which the probability of accepting inferior solutions decays (e.g. simulated annealing algorithm).

These hyperparameters need to be sufficient for the algorithm to be effective in finding an optimal or near optimal solution. However, they are generally specific to the characteristics of the input data (i.e. dimensions of basic strata, the degree of their homogeneity, and number of domains), and whether algorithm is exploratory, exploitative, or a combination of both, in nature.

The hyperparameters utilised in efficiently finding a good quality solution for one set of input variables, for a given algorithm, may be utilised on another problem instance. However, there is no guarantee that they will perform as well, and they may inhibit and delay the search. Again, for a given algorithm, where hyperparameters have been chosen from within certain ranges (input domain), those ranges may also be used to guide the fine-tuning process in another problem instance. However, it is likely that they will need to be adjusted further according to the characteristics of the input data.

In practice, selecting the most appropriate hyperparameters, to minimise (or

maximise) the objective function for a given data input in a machine learning algorithm, is in itself an optimisation problem. Furthermore, as is the case with the main thesis problem, often evaluating the objective function is computationally expensive (and/or financially expensive) in non linear problems where the derivatives and convexity properties are unknown (Brochu et al., 2010). Accordingly, the number of evaluations is limited. Indeed, as the problem instances scale up, the number of evaluations that can be completed within a defined time and cost budget will come down.

In these optimisation problems (of the hyperparameters) there is not an expression of the objective function that we can analyze, and we do not know its derivatives. Such functions are said to be black box functions (Brochu et al., 2010). In these circumstances, the aim is to find the optimal set of hyperparameters in a minimal number of runs.

It is possible to manually fine tune the hyperparameters by taking a randomly generated set of hyperparameters, or alternatively, using knowledge of the input data and the machine learning algorithm - to select a set of hyperparameters. These hyperparameters could be tweaked following each run of the algorithm, using the output from the previous run (and visualisations of the search), as a guide to tune the hyperparameters for the next run. This process could require many runs, before finding a satisfactory set of hyperparameters, but for which we may not be sure of their optimality.

The sequential decision-making nature of this fine-tuning process, also can also be extended to updating hyperparameters using Monte Carlo simulation or various adaptive schemes. However, these approaches, too, can be computationally expensive and there is a certain amount of hope involved in finding the optimal set.

Sequential model based optimization uses models to iteratively determine which configurations of hyperparameters to investigate. It can handle both numerical and categorical parameters and be applicable to multiple problem instances (Hutter et al., 2010). It has become the go-to optimization strategy in recent years. In the process, Bayesian optimisation is implemented to incorporate prior belief about the problem to help direct the sampling, and to trade off exploration and exploitation of the search space, with the assumption of a deterministic output. The advantage of using model based optimisation over trial and error fine-tuning, is that the optimisation has been shown, empirically, to progress sequentially towards the optimal set of hyperparameters.

6.1 Sequential model-based optimization (SMBO)

We draw from (Bischla et al., 2017) in the discussion below. The generic SMBO procedure starts with an initial design of evaluation points (e.g. hyperparameters) for which there are deterministic outcomes, and then iterates the following steps:

- Fit a regression model to the outcomes (objective function) and design points (set of hyperparameters) obtained so far,
- query the model to propose a new, promising point, often by optimizing an infill criterion (acquisition function),
- evaluate the new point with the black-box function and add it to the design.

We start with the black-box function, $(f(x)\colon\ X \rightarrow \mathbb{R})$, discussed above, which takes a set of hyperparameters drawn from a d-dimensional input domain $X = X_1 \times X_2 \times \cdots \times X_d$ and through each run of the machine learning algorithm maps them to a deterministic output y.

Our hyperparameters, X_i $(i = 1, \ldots, d)$, are selected from within a bounded and finite range of either numeric (i.e. $X_i = [l_i, u_i] \subset \mathbb{R}$) or a set of s categorical values ($X_i = \{v_{i1}, \ldots, v_{is}\}$).

Our goal is to find the optimum set of hyperparameters x^*, from X, where f(x) attains the minimum:

$$x^* = \arg\min_{x \in X} f(x). \tag{6.1}$$

In sequential model based optimisation, the expensive evaluation of f is estimated using surrogate models $\hat{f}$ which cheaply estimate f, and which are sequentially or iteratively updated and refined. The general approach is described in more detail below.

1. Sample an initial design of n_{init} points $x^{(j)}$ $(j = 1, \ldots, n_{\text{init}})$ from X. Evaluate f by running the algorithm, for each of these n_{init} points, to provide a set of outcomes $y^{(j)} = f(x^{(j)})$. The rows $\left(y^{(j)}, x^{(j)}\right)$ in the table of design points constitute the data to build the initial surrogate model $\hat{f}$ in the next step.
2. Then fit a surrogate model to all evaluated points $x^{(j)} \in X$ and the corresponding objective functions $y^{(j)}$.
3. Next use an *infill criterion* to propose m points $x^{(j+i)}$ $(i = 1, \ldots, m)$. This criterion uses X as a guide to select m points and these are evaluated using the

surrogate function $\hat{f}$ to determine suitable points for f.

4. We then evaluate these points using f and the new row comprising the hyperparameters and resulting objective function, $\left(y^{(j+i)}, x^{(j+i)}\right)$, is added to the design.
5. If the stopping rules have not been met, go to step 2. Otherwise, finish and return the proposed solution for the optimization problem.

6.1.1 Initial design

The initial design is basically a table of hyperparameters from the input domain for each of which the black-box function is evaluated in order to build the initial surrogate model $\hat{f}$. This design requires a balance between selecting too few suitable points from X and too many. If too few are selected, the optimisation is likely to be sub-optimal, whereas if there are too many, the optimisation may not be completed within a time that is small enough. In the optimisation of hyperparameters discussed later in the thesis, we fill the initial design using the Latin Hypercube Designs technique (McKay et al., 2000).

6.1.2 Surrogate models

The choice of an appropriate surrogate model $\hat{f}$ depends on the ranges from which the X are selected. If the hyperparameters are real numbers, $X \subset R^d$, *Kriging* regression Jones et al. (1998) is recommended. However, as in the experiments below, if X also includes categorical values, *random forests* regression are recommended Hutter et al. (2010).

6.1.3 Infill criteria

The infill criterion, or acquisition function, trades-off exploitation and exploration to guide the optimisation. This is achieved by combining the posterior mean $\hat{\mu}(x)$ and posterior standard deviation $\hat{s}(x)$ (or posterior variance $\hat{s}^2(x)$) (which are estimated by the surrogate model) in a single formula, to search for points with low $\hat{\mu}(x)$ and high $\hat{s}(x)$. We use the *lower confidence bound* to balance $\hat{\mu}(x)$ and $\hat{s}(x)$ for a point x

$$\mathrm{LCB}(x, \lambda) = \hat{\mu}(x) - \lambda \hat{s}(x), \tag{6.2}$$

where $\lambda > 0$, in this case, is a constant that controls the "mean vs. uncertainty" trade-off.

6.1.4 Infill optimization

The infill optimizer uses *focus search* (which shrinks the search space around the best point and samples new random points) to search for the point x which yields the best infill value. The optimisation on the infill criterion is relatively inexpensive which means that more points can be evaluated, to find x.

6.1.5 Termination

In the tests below, we set an upper limit for the total number of evaluations of f.

6.1.6 Final point

We report the hyperparameters leading to the best solution quality.

6.1.7 Implementations of SMBO

We use sequential model-based optimization to fine-tune the hyperparameters in experiments comparing the algorithms in subsequent sections. In these experiments SMBO is used in the manner (i.e. all steps) in which it is described above (section 6.1). In particular, SMBO was implemented in fine-tuning the hyperparameters in the following comparisons:

- the grouping genetic algorithm and simulated annealing algorithm - see sections 8.5.5 and 8.5.6.
- the classical genetic algorithm and simulated annealing algorithm - see section 8.6.2.
- the self-organising map, fuzzy K-means clustering and Neural gas algorithms (and also k-means, expectation maximisation and hill climbing which we didn't train hyperparameters for) - see chapter 9
- the hybrid estimation of distribution algorithm - see sections 10.7.2 and 10.8.1.

Chapter 7

A grouping genetic algorithm

7.1 Introduction

In principle the optimal stratification (i.e. that which yields the smallest sample size) can be found by testing all possible partitionings of *atomic strata*, but the number of possible partitionings grows exponentially with the number of atomic strata.

An efficient search algorithm is necessary to avoid evaluating each possible partitioning. Genetic algorithms (GAs) often converge quickly to optimal or near optimal solutions, and are particularly good at navigating rugged search spaces containing many local minima. Ballin and Barcaroli (2013) combine a GA with the Bethel-Chromy algorithm to search for the minimum sample size. It is used to evaluate each partitioning created by the GA. A full description of the methodology and problem statement is found in (Ballin and Barcaroli, 2013). However, they use a classical GA which is known to be unsuitable for partitioning problems.

We apply genetic operators to the GA that are better suited to this application. It is an example of the class of evolutionary algorithms called *Grouping Genetic Algorithms* (GGAs). The GA, which is a central algorithm in the *SamplingStrata* package, has been updated following this work (Ballin and Barcaroli, 2020). In other words the classical crossover operators have been replaced with grouping genetic operators- making it a grouping genetic algorithm. Since this update, the genetic algorithm has been used by Eurostat in the design of its Land Use/Cover Area frame Survey (LUCAS) survey (Ballin et al., 2018) and Banca d'Italia in the design of the 2020 Survey on Household Income and Wealth (SHIW).

In addition, the algorithm has been embedded as a means of stratified sampling (with the *SamplingStrata* package) by the World Bank in its Survey

Solutions Sampling Tools application. Likewise, the algorithm is currently in use by the Italian National Statistical Institute in the design of various surveys. Furthermore, the *SamplingStrata* package is at differing stages of implementation by the NOAA Alaska Fisheries Science Center, New Zealand Statistical Institute, Statistics Denmark, and Statistics Canada. Inspite of its highly specific nature, *SamplingStrata* is in the 82% percentile of downloads from CRAN (Ballin and Barcaroli, 2020; Barcaroli et al., 2021).

Section 7.2 motivates the work and introduces GGAs. Section 7.2.3 describes our GGA for the problem. Section 7.4 compares the original GA with our GGA on publicly-available test data. Section 7.6 describes a version of our GGA with enhanced performance, using a fast *C++* implementation of the *bethel.r* function which we integrated into *R* using the *Rcpp* package. Section 7.7 concludes the chapter.

7.2 Classical vs grouping genetic algorithms

In this section we discuss "classical" and "grouping" GAs, and explain why the latter are more appropriate for our problem.

7.2.1 Classical genetic algorithms

GAs are a nature-inspired class of optimisation algorithms, modelled on the ability of organisms to solve the complex problem of adaptation to life on Earth. The variables of an optimisation problem are called *genes* and their values *alleles*. A candidate solution is a list of alleles called a *chromosome*. A set of chromosomes is usually called a *population*, so to avoid confusion with the target population we shall use *chromosome population* when referring to GAs. The objective function (which is maximised by convention) is called the chromosome's *fitness*. The search for fit chromosomes (solutions with high objective) uses two *genetic operators*: small random changes called *mutation*, equivalent to small local moves in a hill-climbing algorithm; and large changes called *crossover* in which the genes of two *parent chromosomes* are *recombined*. One well-known recombination operator is *single-point crossover*: choose two *parent* chromosomes with alleles

$$a_1, \ldots, a_N \qquad b_1, \ldots, b_N$$

select a random integer i (the *crossover point*) such that $1 \leq i < N$, and generate two new *offspring* chromosomes

$$a_1, \ldots, a_i, b_{i+1}, \ldots, b_N \qquad b_1, \ldots, b_i, a_{i+1}, \ldots, a_N$$

These might be further subjected to random *mutation*, in which a few alleles are changed, before placing them back into the chromosome population. There are a variety of methods for selecting parents and replacing existing chromosomes. In *generational* GAs the entire chromosome population is replaced by offspring, and parents are often selected randomly but with a bias toward fitter chromosomes; while in *steady-state* GAs only one offspring is generated in each GA iteration, and usually replaces the least-fit chromosome in the chromosome population. GAs often give more robust results than search algorithms based on hill-climbing, because of their use of recombination. They have found many applications since their introduction in 1975 by John Holland.

The original GA which is represented in the *R* (R Core Team, 2015) package *SamplingStrata* (Barcaroli et al., 2014), is an elitist generational GA in which the atomic strata L are considered to be elements of a set (or genes) for a standard crossover strategy. In each iteration the best solutions (the *elite*) are carried over to the next generation. Each gene represents a variable in the problem. We refer to this as a classical GA because a classical problem representation and genetic operators are used, as described below.

Dividing atomic strata into disjoint groups is an example of a *grouping* problem, related to *cutting*, *packing* and *partitioning* problems. The motivation for our work is that classical GAs are known to perform poorly on grouping problems. The reason is that the chromosomal representation of a grouping contains a great deal of *symmetry* (or *redundancy*): permuting the group names yields an equivalent grouping, so each grouping has multiple representations. Symmetry has a damaging effect on GAs because recombining similar parent groupings might yield a very different offspring grouping, violating the basic GA principle that parents should tend to produce offspring with similar fitness. In extreme cases, a classical GA might perform even worse than a completely random search. We provide two examples to illustrate the problem.

To illustrate the problem with symmetry in our first example the parents represent the same grouping in different ways. Note that to increase readability, letters A - F are used as alleles instead of integers in the presentation here. Consider the following two chromosomes:

chromosome	groups represented					
	A	B	C	D	E	F
ABCDEF	{1}	{2}	{3}	{4}	{5}	{6}
FEDCBA	{6}	{5}	{4}	{3}	{2}	{1}

which both represent the grouping $\{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\}\}$. Now suppose we apply single-point crossover to obtain two new offspring chromosomes from these parents. Arbitrarily choosing the center of the chromosomes as the crossover point, we obtain offspring:

chromosome	groups represented					
	A	B	C	D	E	F
ABCCBA	$\{1,6\}$	$\{2,5\}$	$\{3,4\}$	$\emptyset$	$\emptyset$	$\emptyset$
FEDDEF	$\emptyset$	$\emptyset$	$\emptyset$	$\{3,4\}$	$\{2,5\}$	$\{1,6\}$

which both represent the completely unrelated grouping $\{\{1,6\},\{2,5\},\{3,4\}\}$: no groups at all are passed from the parents to the offspring. Hence the offspring and parent fitnesses can be completely unrelated to each other, which reduces the GA to near-random search. As another example, consider the following two classical chromosomes:

chromosome	groups represented					
	A	B	C	D	E	F
AECFEC	{1}	$\emptyset$	$\{3,6\}$	$\emptyset$	$\{2,5\}$	{4}
DFFDAA	$\{5,6\}$	$\emptyset$	$\emptyset$	$\{1,4\}$	$\emptyset$	$\{2,3\}$

which in turn represent the different groupings $\{\{1\},\{3,6\},\{2,5\},\{4\}\}$ and $\{\{5,6\},\{1,4\},\{2,3\}\}$. Using the same crossover strategy we obtain offspring:

chromosome	groups represented					
	A	B	C	D	E	F
AECDAA	$\{1,5,6\}$	$\emptyset$	{3}	{4}	{2}	$\emptyset$
DFFFEC	$\emptyset$	$\emptyset$	{6}	{1}	{5}	$\{2,3,4\}$

representing the groupings:

$\{\{1,5,6\},\{3\},\{4\},\{2\}\}$ and $\{\{6\},\{1\},\{5\},\{2,3,4\}\}$.

Note that these offspring have very little in common with their parents, as the only preserved groups are $\{1\}$ and $\{4\}$.

7.2.2 Grouping genetic algorithms

The symmetry problem can be tackled by designing more complex genetic representations and operators (Galinier and Hao, 1999) or by clustering techniques (Pelikan

and Goldberg, 2000). The risk of clustering is that genetic diversity may be lost if the clusters are too tight, leading to search stagnation (Prügel-Bennett, 2004). Instead we follow the former approach by designing a GGA (Falkenauer, 1998), which have been shown to perform far better than classical GAs on grouping problems.

GGAs are designed specifically to solve grouping problems and have found many applications, including WiFi network deployment (Agustín-Blas et al., 2011), wireless network design (Brown and Vroblefski, 2004), steel plate cutting (Hung, Sumichrast, and Brown, 2003), production plant layout (De Lit, Falkenauer, and Delchambre, 2000) and social network analysis (James et al., 2010). They may use the same heuristics as other GAs (parent selection, offspring replacement, etc) but they use different genetic encoding and operators: that is, how they map a problem to chromosomes and how they perform recombination and mutation.

Genetic operators

As mentioned above, a new chromosome population is generated by the application of various genetic operators to the solutions in the current one. The classical GA uses two genetic operators: mutation and single-point crossover in which the genes of two parent chromosomes (solutions) are recombined. The most important operator is the crossover, which is also called the recombination operator. In practice, the role of this operator is to combine together (in a cut-and-concatenate manner) pieces of information from different chromosomes in the chromosome population. However, the crossover happens at the allele or basic stratum level.

The classical genetic operators are not suitable for grouping problems, as recombining similar parent groupings in this way might yield a very different offspring grouping. However, the objective function depends on the grouping of atomic strata. Thus, this method of recombination is unlikely to consistently generate fitter offspring. More generally speaking, Falkenauer (1998) puts it this way: the crossover casts context-sensitive information out of context. This can be quite inefficient and violates the *schema theorem* that the GA should only explore solution structures which have the best chances of containing the optimum.

The reason for this is that the structures of classical GA chromosomes are object oriented, and are not suitable for the optimisation of a grouping cost function. To put it another way, the cost function depends on groups, whereas the groups are not directly part of a classical GA chromosome structure. To remedy this, the genetic operators work with a representation of the grouping, where each gene is in one group, rather than singular genes in the chromosome. For example, consider the

following chromosomes from section 7.2.1:

	groups represented					
chromosome	A	B	C	D	E	F
AECFEC	$\{1\}$	$\emptyset$	$\{3,6\}$	$\emptyset$	$\{2,5\}$	$\{4\}$
DFFDAA	$\{5,6\}$	$\emptyset$	$\emptyset$	$\{1,4\}$	$\emptyset$	$\{2,3\}$

These chromosomes represent different groupings of basic strata

$$\{\{1\},\{3,6\},\{2,5\},\{4\}\} \text{ and } \{\{5,6\},\{1,4\},\{2,3\}\}.$$

In the two classical chromosomes each atomic stratum is assigned a letter, i.e. $AECFEC$ and $DFFDAA$. On the right hand side of the table, we see the group representation of each chromosome, where an integer represents an atomic stratum.

In the classical GA, the genetic operators work on the chromosome. However, in the grouping genetic algorithm the genetic operators work with the groups $ACEF$ and ADF, rather than the objects within the groups. Another representation of this is:

$$AECFEC{:}ACEF$$
$$DFFDAA{:}ADF$$

where the groups $ACEF$ and ADF represent an augmentation of the chromosome. However, in the grouping genetic algorithm these are the chromosome.

To put it another way, each gene is the smallest part, or building block, of a solution, and represents a group rather than an object. The idea being that information from the groups are transmitted from parents to offspring in the recombination process, thus exploiting the fitness of the parents, while simultaneously allowing exploration of the search space. If a genetic algorithm does not succeed in this, it is little more than a random search or naive evolution.

Crossover

As we have seen, in the classical GA, genes are combined in a cut and concatenate manner, independently of each other. As such, in some grouping problems a cut and concatenate approach could produce invalid offspring, in other cases, the offspring might be valid, but of poor quality. However, since the GGA has one gene per group, it is hardly ever the case that genes from one parent can be combined with genes from another parent to generate an offspring, without some overlap of the same objects in the two genes.

This is achieved through the adaptation of some groups from one parent to those in the other. In this way, the offspring will contain objects that are not present

in exactly the same way as they are in the parents, but the main groupings remain intact. As constraints and cost functions differ between grouping problems, the crossover will not manifest in the same way for all of them. In other words, the implementation details for crossover depend on the particular grouping problem at hand. However Falkenauer (1998) provides the following five stage pattern:

1. Randomly select two crossing points, which represents the crossing section, in each of two parents. As the crossover work with the groups part of the chromosome, this means selecting some of the groups in each parent.
2. Inject the contents of the crossing section of the second parent at the first crossing site of the first parent.
3. Eliminate all objects now occurring twice from the groups they were members of in the first parent. The 'old' membership of these groups gives way to the membership pattern of the 'new' injected groups. As a result, some of the old groups no longer contain the same objects, as they have been eliminated.
4. If necessary, adapt the resulting groups, according to the hard constraints and the cost function to optimise. At this stage, local problem-dependent heuristics can be applied. This may involve emptying of the groups modified by the injection step, followed by the reinsertion of objects 'unassigned' in that step.
5. Apply points 2 through 4 to the two parents, with their roles reversed, to generate the second child.

The remaining grouping operators in the GGA, namely mutation and inversion, have secondary roles of maintaining diversity in the chromosome population (mutation) and assisting the evolution of the fitness of solutions (inversion).

Mutation

A mutation operator for grouping problems should work on the groups part of the chromosome, rather than the objects. As with crossover, mutation depends on the problem at hand. Falkenauer (1998) provides two strategies: create a new group or eliminate an existing group. Another approach can be to move a small number of randomly selected objects among their respective groups (this is the approach we apply below).

Inversion

The inversion operator changes the position of some of the chromosome genes, in order to assist the transmitting of good groups from one parent to another, by

virtue of the fact that these groups could now be within the crossing section of one parent. Unlike crossover and mutation, this only affects positions of genes on the group representation side of the chromosome, rather the positions of objects in the chromosome. To illustrate consider the following representation of the chromosomes from section 7.2.1:

$$AECFEC{:}ACEF$$
$$DFFDAA{:}ADF$$

The groups could be inverted to:

$$AECFEC{:}CEAF$$
$$DFFDAA{:}DFA$$

The groups still contain the same objects, as before, only the position of genes within the group representation has changed. In the first chromosome, for example, groups E and A are closer together than before, and are now more likely to be selected in a crossing section for that chromosome. A similar example could be made for the second chromosome, as F and A are now closer together.

Use of the genetic operators for this problem

We shall illustrate our application of these genetic operators in a grouping genetic algorithm for this problem. Again, we use the chromosomes from section 7.2.1 to illustrate this process.

	groups represented					
chromosome	A	B	C	D	E	F
AECFEC	$\{1\}$	$\emptyset$	$\{3,6\}$	$\emptyset$	$\{2,5\}$	$\{4\}$
DFFDAA	$\{5,6\}$	$\emptyset$	$\emptyset$	$\{1,4\}$	$\emptyset$	$\{2,3\}$

The GGA represents a grouping as an ordered list of subsets, omitting empty sets. We have two groupings: $ACEF$ and ADF. Each group contains basic strata. Through the groupings, the two parent chromosomes might, thus, be represented in this way:

$$\langle\{1\},\{3,6\},\{2,5\},\{4\}\rangle \qquad \langle\{5,6\},\{1,4\},\{2,3\}\rangle$$

GGA mutation is simple: an item is moved from one group to another. However, the GGA recombination operator is more complicated. Choose a crossing section in each parent, for example $\langle\{1\},\{3,6\}\rangle$ from the 1st parent and $\langle\{1,4\}\rangle$ from the 2nd parent. Then inject the 1st crossing section into the 2nd parent at a

random point, and vice-versa:

$$\langle\{1\},\{3,6\},\underline{\{1,4\}},\{2,5\},\{4\}\rangle \qquad \langle\{5,6\},\{1,4\},\underline{\{1\},\{3,6\}},\{2,3\}\rangle$$

Next, remove any repeated objects that were already in the receiving parent:

$$\langle\emptyset,\{3,6\},\{1,4\},\{2,5\},\emptyset\rangle \qquad \langle\{5\},\{4\},\{1\},\{3,6\},\{2\}\rangle$$

Finally remove any empty sets:

$$\langle\{3,6\},\{1,4\},\{2,5\}\rangle \qquad \langle\{5\},\{4\},\{1\},\{3,6\},\{2\}\rangle$$

These are the offspring. Clearly, both offspring have much in common with both parents, as 5 of the 7 parent groups survive in the offspring: $\{1\}$, $\{4\}$, $\{1,4\}$, $\{2,5\}$ and $\{3,6\}$. This property of the GGA injection-based recombination makes it much more likely that offspring have similar fitness to parents, which in turn helps the GGA to iteratively improve the chromosome population.

It might be noticed that the GGA problem representation still contains symmetry: any grouping still has multiple representations, obtained by permuting the subsets in the ordered list. But the genetic operators are almost independent of this ordering so it is almost irrelevant. The only effect of the ordering is to limit the set of possible injections: in the second example of Section 7.2.1 we cannot inject a non-existent crossing section for example such as $\langle\{1\},\{4\}\rangle$ from parent 1 because those two groups are not adjacent. This limit is removed by the inversion genetic operator which selects a section of the chromosome and reverses it. For example:

$$\langle\{1\},\{2\},\underline{\{3,6\},\{4\},\{5\}}\rangle \longrightarrow \langle\{1\},\{2\},\underline{\{5\},\{4\},\{3,6\}}\rangle$$

This does not change the grouping represented by the chromosome, but reordering the groups in the chromosome makes all injections possible.

The GGA uses grouping operators which have little in common with the standard operators in the classical GA. Falkenauer (1998) indicates that the GGA differs a lot from the GA by so much that the author was recommended to call it an evolutionary algorithm. However, as evolutionary algorithms are a much wider class of search algorithms, with a broader range of evolutionary strategies, and given that the GGA uses crossover and operators, it became known as the grouping genetic algorithm.

Although, crossover, mutation and inversion are the common operators used in

GGAs, there is no canonical algorithm. Instead GGAs tend to be tailored for specific applications, and in principle any GA can be adapted to grouping problems by using grouping operators (although this results in a different algorithm (Falkenauer, 1998)). In Section 7.2.3 we outline the GGA for our problem.

7.2.3 Application to the joint stratification and sample allocation problem

As mentioned above our GGA is based on the GA described in (Ballin and Barcaroli, 2013) and represented in *R* in the *SamplingStrata* package (Barcaroli et al., 2014), but with grouping operators and chromosomes instead of the classical versions. This change is the only novelty of our algorithm (except for the optimisation described in Section 7.6) but its effect on performance is large. We inserted the GGA into a modified version of the function called *rbga.r* from the *genalg* R package (Willighagen, 2005). It is designed to work with the other functions in *SamplingStrata*, and is applied to the joint stratification and optimum sample size problem as follows:

Firstly we generate a random initial chromosome population. We obtain the full set of viable basic strata of size L from the Cartesian product of the auxiliary variables (see above). A chromosome population, or set of candidate solutions, of a predetermined size N_p, is randomly generated or utilised from a previous run of the algorithm. Each candidate solution (or individual) is a vector representing a stratification of the atomic strata.

The stratification in the vector is generated by K integers randomly generated from the interval $[1, K]$, so the genome of each individual contains integers ranging in value v_k from $1 < v_k < K$. The frequency with which the integers occur is used to define the groups for the atomic strata. The GGA now evolves the chromosome population as follows.

1. Evaluate the fitness of each individual in the new chromosome population, by calculating their quality, and rank them.
2. Calculate the number of individuals E to be kept from one generation to the next. This is obtained by multiplying the elitism rate e by the chromosome population size, N_p and rounding down to the nearest integer: $E = \lfloor e \times p \rfloor$.
3. Then take the first E individuals from the sorted chromosome population, which equates to the E best individuals carried forward to the next generation.
4. At a preset probability invert each individual in the chromosome population.

Inversion is described in more detail above.

5. Select two parents to generate an offspring. This needs to be done for each remaining individual in the chromosome population after the elite individuals have been selected, i.e. $N_p - E$ times. The selection of parents is done using the the Probability Density Function, with a mean of 0 and a standard deviation of the population size divided by three.
6. Create $N_p - E$ child chromosomes, by randomly selecting two crossover points of the groups list in parent 1 and inserting the corresponding values from 'parent 1' into a copy of 'parent 2', to create the child. Remove any empty groupings from each chromosome. Empty groups can occur when the new values are inserted into the child. Rename (renumber) the groups so that there is a consecutive order to the group names.
7. Apply mutation to each value of the child, at a rate which is determined by a preset mutation probability.
8. Repeat steps 1 to 7 until the last iteration, the output from the last iteration is the chromosome population. This contains the best quality solution (chromosome) that has been found, from the search space of possible solutions, up to this stage.

The number of iterations which we wish to run the algorithm for should be enough to give the GGA a chance to converge on a near optimal or optimal solution. If, however, the optimum solution is known beforehand (for testing purposes, e.g. the iris data set example in chapter 7 - where it is possible to do an evaluation of all possible solutions, and the optimal solution is known) the algorithm can be set to stop at this point.

The fitness for each chromosome in the chromosome population is calculated by estimating the minimum sample size required to meet constraints. The resulting sample size or cost is then ranked from lowest to highest and the chromosome population is sorted accordingly.

The evaluation function comprises two functions called *aggrStrata.r* and the *bethel.r* function to calculate the total cost or minimum sample size for each chromosome in the chromosome population.

The *aggrStrata.r* function groups the atomic strata according to their integer grouping while calculating the mean, standard deviation and total cost for each of the target features. This information is then taken by the *bethel.r* function to calculate the minimum fitness value for each individual of the chromosome population.

For each individual, there is a partition $P(v)$ of the L number of atomic strata. Recall that, the partition is represented by the vector $v = [1, ..., K]$. In other words, this vector represents the grouping of the atomic strata (into larger strata).

The notation $D_i(i = 1, 2, ..., Q_P(v))$ describes one stratum that results from this partition. As mentioned above, it is possible that the grouping will be such that each atomic stratum is in a single constituent group, i.e. each D_i is the same as each l_k. More likely however is the grouping of atomic strata into disjoint groups such that each $D_i = l_j^i, ..., l_l^i$ is an aggregation of a partition of the atomic strata, $l_j^i, ..., l_l^i$.

aggrStrata.r does not need to recalculate the means and variances if D_i is the same as each l_k as they have already been calculated. On the other hand if the atomic strata have been grouped then *aggrStrata.r* will calculate the means and variances as follows:

$$\bar{Y}_{g,D_i} = \frac{\sum_{l_k \in D_i} \bar{Y}_{g,l_k} N_{l_k}}{\sum_{l_k \in D_i} N_{l_k}} \tag{7.1}$$

$$S^2_{g,D_i} = \left\{ \frac{\sum_{l_k \in D_i} (N_{l_k} - 1) S^2_{g,l_k} + \sum_{l_k \in D_i} N_{l_k} (\bar{Y}_{g,l_k} - \bar{Y}_{g,D_i})^2}{\sum_{l_k \in D_i} (N_{l_k} - 1)} \right\} \tag{7.2}$$

where:

$\bar{Y}_{g,D_i}$ and $\bar{Y}_{g,l_k}$ are the mean values of the aggregated stratum D_i and atomic strata l_k N_{l_k} is the number of units in atomic stratum l_k S^2_{g,D_i} and S^2_{g,l_k} are the variances in the aggregated stratum D_i and atomic stratum l_k

For each aggregated stratum *aggrStrata.r* will calculate the average cost per sampling unit by multiplying the per unit cost for each atomic stratum by the population total for each atomic stratum. These are then summed together and divided by the aggregated population total:

$$c_{D_i} = \frac{\sum_{l_k \in D_i} C_{l_k} N_{l_k}}{\sum_{l_k \in D_i} N_{l_k}} \tag{7.3}$$

The total population in each aggregated stratum is calculated by summing together the population totals for each atomic stratum:

$$N_{D_i} = \sum_{l_k \in D_i} N_{l_k} \tag{7.4}$$

, where $n_1, ..., n_{H_P(v)}$ are the sample sizes for each stratum in the partitioning of individual $P(v)$.

There is now enough information to use the *bethel.r* function to calculate the

fitness for each individual in the population, which is the total cost that incorporates the sampling cost and minimum sample size for each aggregated stratum, along with fixed cost, if included.

$$C(n_1, ..., n_{H_P(v)}) = C_0 + \sum_{h=1}^{H_P(v)} C \tag{7.5}$$

The number of individuals to be kept from one generation to the next, E is obtained by multiplying the elitism rate, e by the population size, N_p and rounding down to the nearest integer.

$$E = floor(eXp) \tag{7.6}$$

The mutation function is as described by Cortez (2014). For each atomic stratum of the child chromosomes a random number is generated between 0 and 1. If this number is less than the mutation probability, the value or group number of the atomic stratum is changed (by generating a new value from between 1 and the maximum grouping number), otherwise it is not changed.

Recall that strata should be mutually disjoint but internally homogeneous subsets of the population. There are two levels of strata in this problem. The basic level is the atomic strata created from the cartesian product of auxiliary variables (see section 2). We intend to create higher level strata/subsets or groupings of atomic strata.

If each atomic stratum contains values that are the same or close in value then smaller sample sizes are needed for a precise estimate of the mean or total for the target variables. It follows also that grouping homogeneous atomic strata into strata will also require a smaller sample size to meet precision constraints. This is what leads to a good candidate solution, i.e. the strata are internally homogeneous but mutually disjoint and smaller samples sizes result. The GGA is summarised in Figure 5.

Algorithm 5 Grouping Genetic Algorithm (GGA)

1: Randomly generate a chromosome population of size (N_P)
2: Selection part 1
   (a) Rank chromosomes based on sample size
   (b) Save best E chromosomes for the next generation
3: Inversion
   With probability 0.01 invert groups in the N_{P} chromosomes
4: Selection part 2
   For each of the remaining $(N_{\mathrm{P}} - E)$ chromosomes in the new generation:
   (a) Draw parents 1 and 2 from the aforementioned N_{P} chromosomes (higher ranked chromosomes have a higher probability of being selected)
   (b) Perform crossover as explained in Section 7.2.2
   (c) Remove empty groups
   (d) Renumber groups
5: Mutation
   Mutate integers in $(N_{\mathrm{P}} - E)$ chromosomes at a selected probability
6: If number of iterations < maximum (optional: and sample size > desired value) go to step 2

7.2.4 Hyperparameters

Typically we decide in advance the number of iterations which we wish to run the algorithm for. This should be enough to give the GGA a chance to converge on the optimum solution after the mutation and inversion probabilities have been applied. If, however, (in cases where complete enumeration is possible and has already been completed) the quality of the optimum solution is known beforehand the algorithm can be set to stop at this point.

The number of iterations is usually decided with experience of using the GGA on similar target and auxiliary variables for similar data sets, or with the existing data set and target and auxiliary variables. It may require a number of experiments using the GGA (or GA) before the number of iterations needed to reach convergence can be estimated. In fact there is a possibility that either the GGA or GA would appear to have reached convergence after a set number of iterations, but instead have become trapped in a local minimum. It may be useful to increase the number of iterations and try alternative mutation probabilities in order to be more sure that it has converged on a global minimum.

This implies a number of trial runs before finally deciding the parameters under which to run the algorithms. Therefore the fact the GGA has been shown to attain

convergence quicker than the GA is likely to compound the improvement in total training time. In the experiments described below we keep the number of iterations small as we want to demonstrate the ability of the GGA to converge on a solution within that number of iterations.

We use either the mutation settings specified in the examples provided by (Ballin and Barcaroli, 2013) or the default mutation settings in (Barcaroli et al., 2014). Principally, we have applied grouping genetic operators and inversion to the GA designed by (Ballin and Barcaroli, 2013): it is the grouping genetic operators that make it a GGA.

In the following experiments, we therefore, wanted to demonstrate the performance between the different GA and GGA genetic operators rather than experiment with parameters such as varying the number of iterations, chromosome population size, mutation probability, or elitism rate.

However, these are not the optimal settings for either algorithm and an alternative approach would have been empirically determine suitable hyperparameters for a particular problem instance. We demonstrate the output resulting from fine tuning the hyperparameters for the GGA in chapter 8.

The size of the chromosome population can be decided by trial and error or using finetuning. It is advisable to consider the evaluation time of each chromosome when setting the size: if there are too many chromosomes in the set, it might take an extra long time to move from one iteration to the next, and we found that the *bethel.r* algorithm (i.e. the Bethel-Chromy evaluation algorithm in (Barcaroli et al., 2014)) takes several seconds to evaluate even one chromosome for the larger data sets we used in this chapter (we discuss this further in Section 7.6).

The mutation probability can be selected in advance by the user. Typically, the probability of mutation should be such that it increases the chance of the GGA leaving a local minimum, but not disrupt the natural evolution of chromosomes from one generation to the next. On the other hand we have fixed the inversion probability at 0.01, to maintain diversity (although this setting is fixed, in principle, it could be fine-tuned to suit the problem instance) .

We used an elitism rate of 0.2 in each of the experiments. Primarily, this was because it was the default setting in the GA, but also, again, because our focus was on demonstrating the effect of the grouping operators. However, this value can also be tuned, in order to strike a balance between the number of solutions to be carried over to the next generation of the population, and the number of new solutions generated. Again, the value of this parameter, can be fine tuned for the particular

problem instance.

7.3 Memoisation

Memoisation (Michie, 1968) is a run-time optimisation technique commonly used in computer science that caches the results of referentially transparent function calls which evaluate the same inputs, in order to avoid repetitive computation of results and thus speed up computer programs. It is particularly useful for expensive function calls, eliminating the cost of calling that function again after the first call.

A memoisation function works by storing previously evaluated results in a look up table and searching that look up table for that result when the function is called for the same input. Memoisation lowers a functions evaluation time cost in exchange for memory cost, i.e. the run-time is optimised in a trade-off with memory space. As such, it is useful in cases where inputs re-occur for previously computed results, such as in dynamic programming (where an input is broken down into simpler sub-inputs), but also has uses in cases where the same input can appear more than once, such as in genetic algorithms.

Memoisation entails recording an entry in the look-up table as each new solution is evaluated. This process usually begins when the algorithm commences running. However, look-up tables may also be pre-populated. The look-up table grows in memory allocation as the results for more inouts are added, and in some cases there is an upper limit on table size, before previously recorded results start to be replaced by newer recorded results.

This has advantages in problem instances where the look-up table is lilely to grow such a memory allocation that slows the run-time of the algorithm, thus reducing the gains attainable from the run-time optimisation. This also has advantages in cases where the same solution structures are no longer likely to appear, i.e. the algorithm has evolved the solution structure to a point where inferior solution qualities are no longer likely to appear. Both the GA and GGA use the same memoisation function (Wickham et al., 2021) in R.

7.4 Comparing the genetic algorithms

We now run a number of comparisons between the original GA and our GGA using publicly available data sets. Unless otherwise stated, for all the cases presented below, we adopt the following parameter setting for both genetic algorithms, where

$N_P = 20$, $U_g \equiv 0.05$, the elitism rate is 0.2, and the mutation probability is 0.05.

7.4.1 A comparison for the iris data set

(Ballin and Barcaroli, 2013) use the iris data set (Anderson, 1935; Fisher, 1936; R Core Team, 2015) to demonstrate that the GA they propose can find the optimum stratification i.e. the stratification or grouping of atomic strata which supplies the minimum sample size. The iris data set is small and is widely available. It has 150 observations for 5 variables Sepal Length, Sepal Width, Petal Length, Petal Width and Species.

Species is a categorical variable which has three levels, setosa, versicolor and virginica, each of which have 50 observations. The remaining four variables are continuous measurements for length and width in centimetres. (Ballin and Barcaroli, 2013) select Petal Length and Petal Width as variables of interest, i.e. target variables. They select Sepal Length and Species as two auxiliary variables.

They convert Sepal Length to a categorical variable using a k-means algorithm (Hartigan and Wong, 1979) to define three clusters (i.e. 4.3 to less than 5.5, 5.5 to less than 6.5, 6.5 to 7.9). The Cartesian product of the categorical version of Sepal Length with Species creates 9 atomic strata. However, one atomic stratum is empty because there are no corresponding values in Petal Length and Petal Width. Therefore there are 8 usable atomic strata for this example.

Table 7.4.1: *Reproduction of table of atomic strata for iris data set as found in (Ballin and Barcaroli, 2013), P.379*

Stratum	N	M1	M2	S1	S2	X1	X2	DOMAIN
[4.3; 5.5] (1)*setosa	45	1.466667	0.244444	0.17127	0.106574	[4.3; 5.5] (1)	setosa	1
[4.3; 5.5] (1)*versicolor	6	3.583333	1.166667	0.491313	0.205481	[4.3; 5.5] (1)	versicolor	1
[4.3; 5.5] (1)*virginica	1	4.5	1.7	0	0	[4.3; 5.5] (1)	virginica	1
[5.5; 6.5] (2)*setosa	5	1.42	0.26	0.172047	0.08	[5.5; 6.5] (2)	setosa	1
[5.5; 6.5] (2)*versicolor	35	4.268571	1.32	0.367051	0.189435	[5.5; 6.5] (2)	versicolor	1
[5.5; 6.5] (2)*virginica	23	5.230435	1.947826	0.318194	0.28873	[5.5; 6.5] (2)	virginica	1
[6.5; 7.9] (3)*versicolor	9	4.677778	1.455556	0.193091	0.106574	[6.5; 7.9] (3)	versicolor	1
[6.5; 7.9] (3)*virginica	26	5.876923	2.107692	0.494825	0.228579	[6.5; 7.9] (3)	virginica	1

The initial atomic strata are reproduced in Table 7.4.1 where M_g refers to the means for the corresponding Y_g values in each atomic stratum l_k; S_g refers to the corresponding stratum population standard deviation estimates. There are 4,140 possible partitionings of the 8 atomic strata. Consequently, it is possible to test within a reasonable amount of time the sample size for the entire search space using the

bethel.r function. This has already been done (Ballin and Barcaroli, 2013) and the minimum sample size is known to be 11.

This test can be used to determine whether the new GA correctly finds the minimum sample size without exploring the entire search space. We use $N_p = 20$ in this case. For this test the *bethel.r* function will search for the minimum sample size, in integers rather than real numbers, i.e. real numbers are rounded up to the nearest integer. The chromosomes will then be ranked by sample size in ascending order. Accordingly the elite chromosomes are taken into the next iteration and the remaining chromosomes are generated using the recombination method for each algorithm.

We will compare the number of chromosomes generated to find the optimal stratification in the two algorithms as well as the number of iterations. This may also be thought of as the number of calls made to the Bethel-Chromy algorithm, because each generated chromosome is evaluated (although the GA uses memoisation, so that it doesn't evaluate the same solution twice). Our anticipation is that the GGA should be more efficient, and thus typically find the optimal solution in fewer iterations than the GA.

The maximum number of iterations is set to 200, because using Ballin and Barcaroli (2013) as a guide we anticipate that both algorithms will find the correct solution in fewer iterations than this. Thus we have added a piece of code to both algorithms such that they stop when the optimal sample size, n=11, has been reached and supply the number of iterations taken to reach that point. This approach is different to that of Ballin and Barcaroli (2013) who report the number of times in 10 experiments the GA finds the correct solution for a given number of iterations ranging incrementally from 25 to 200. However, we feel this approach would better demonstrate that the GGA can find the correct solution in less iterations even on the small iris data set experiment.

Table 7.4.2: *Iris data set experiment results for GA and GGA*

(a) GA			(b) GGA		
	Number of			Number of	
Experiment	Iterations	Chromosomes	Experiment	Iterations	Chromosomes
1	22	356	1	6	100
2	14	228	2	12	196
3	18	292	3	5	84
4	31	500	4	6	100
5	2	36	5	10	164
6	2	36	6	9	148
7	17	276	7	10	164
8	23	372	8	5	84
9	11	180	9	6	100
10	36	580	10	8	132
11	13	212	11	9	148
12	117	1876	12	4	68
13	14	228	13	9	148
14	17	276	14	7	116
15	14	228	15	9	148
16	2	36	16	6	100
17	8	132	17	13	212
18	32	516	18	6	100
19	13	212	19	6	100
20	8	132	20	5	84
21	26	420	21	9	148
22	0	4	22	5	84
23	18	292	23	5	84
24	16	260	24	4	68
25	26	420	25	9	148
26	7	116	26	7	116
27	24	388	27	9	148
28	8	132	28	19	308
29	19	308	29	3	52
30	11	180	30	8	132

Table 7.4.2 provides the number of iterations (and chromosomes generated) taken to find n=11 over 30 experiments, with the same hyperparameters and a seed of 1509 incrementing by 1 for each new experiment, for both GAs.

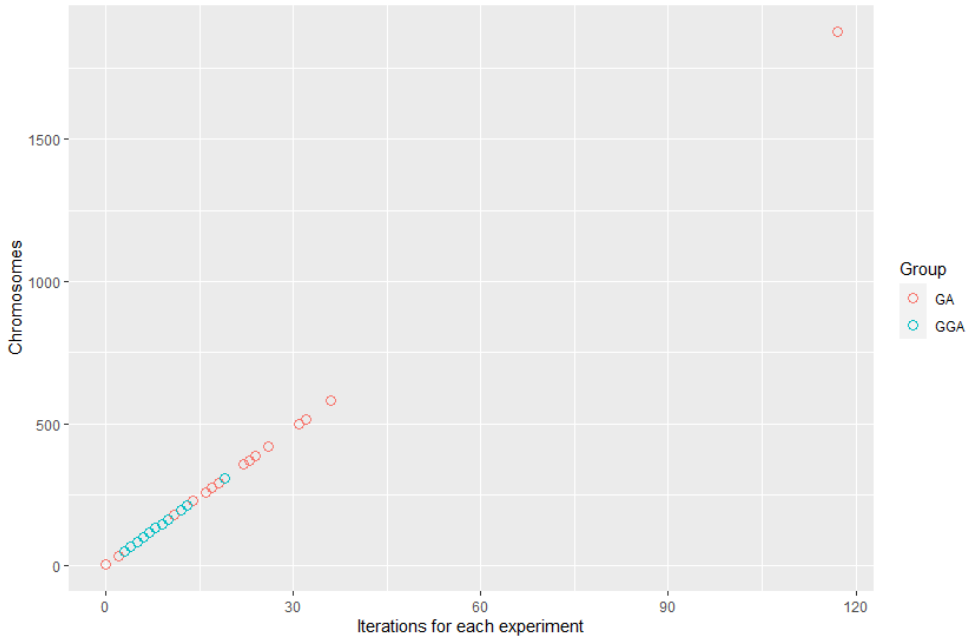


Figure 7.1: *Scatter plot of Results for Chromomomes generated v Iterations to find n=11 for GA and GGA after 30 experiments*

In Figure 7.1 the linear relationship between the number of iterations and the

number of chromosomes generated is clear (each iteration $N_p - E$ new chromosomes are generated). However, we can also see that, generally speaking, for the GGA less iterations are needed, and thus less chromosomes generated and the Bethel-Chromy algorithm is called less often, to find $n = 11$. Note, for the GA experiment # 22 the randomly generated population had the optimum solution prior to implementing the GA (hence the result shows 0 iterations).

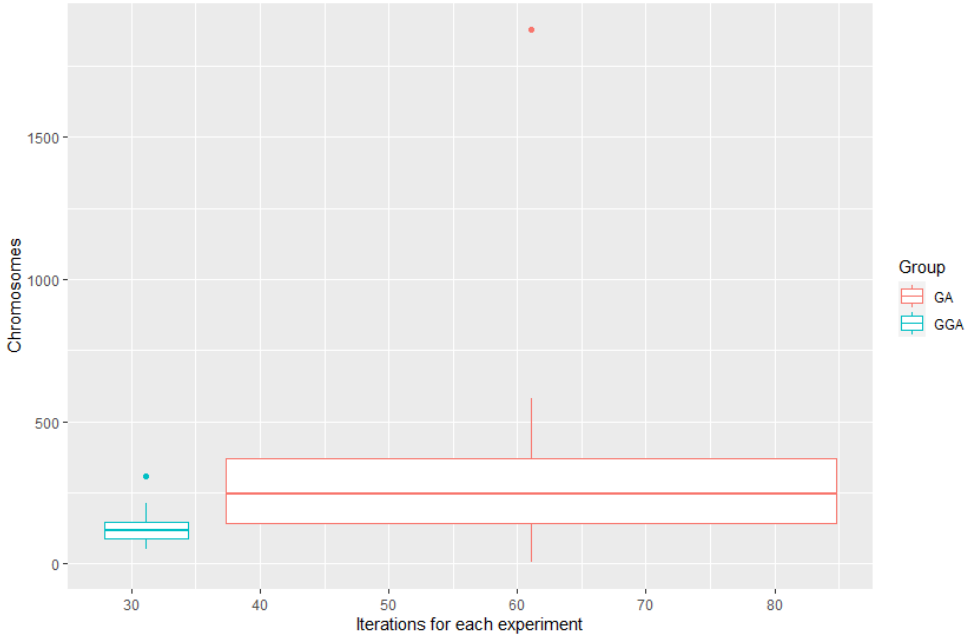


Figure 7.2: *Boxplot distribution of number of Chromomomes generated to find n=11 for GA and GGA after 30 experiments*

Figure 7.2 provides the distribution of the number of chromosomes generated to find the optimal solution for the GA and the GGA. As with Figure 7.1, the boxplots indicate that the GGA typically needs to generate fewer chromosomes to find the optimum solution.

Table 7.4.3: *Example stratifications for the GA and GGA on the iris data set for n=11*

		Y1			Y2		
	Stratum	**N**	**Mean**	**SD**	**Mean**	**SD**	**Sample Size**
GA	**1**	50	1.462	0.1685	0.246	0.1026	2
	2	50	4.26	0.4562	1.326	0.1911	3
	3	1	4.5	0	1.7	0	1
	4	23	5.2304	0.3112	1.9478	0.2824	3
	5	26	5.8769	0.4852	2.1077	0.2241	2
Total		**150**					**11**
	Stratum	**N**	**Mean**	**SD**	**Mean**	**SD**	**Sample Size**
GGA	**1**	23	5.2304	0.3112	1.9478	0.2824	3
	2	50	1.462	0.1685	0.246	0.1026	2
	3	26	5.8769	0.4852	2.1077	0.2241	2
	4	51	4.2647	0.4529	1.3333	0.1962	4
Total		**150**					**11**

Table 7.4.3 provides example stratifications for the GA and GGA that both provide the optimal sample size necessary to meet precision constraints. (Ballin and Barcaroli, 2013) indicate that a number of partitionings from the total of 4,140 possible partitionings provide the minimum sample size. These range in size from 3 to 5 strata. The same tendency can be observed in the latter cases.

7.4.2 Swiss municipality data set

The swissminucipalities data set provided by (Barcaroli et al., 2014) refers to the Swiss municipalities in 2003. Each municipality belongs to one of seven regions which are at the NUTS-2 level, i.e. equivalent to provinces. Each region contains a number of cantons, which are administrative subdivisions. There are 26 cantons in Switzerland. The data, which was sourced from the Swiss Federal Statistical Office and is included in the *sampling* and *SamplingStrata* packages, contains 2,896 observations (each observation refers to a Swiss municipality in 2003). They comprise 22 variables, details of which can be examined in (Barcaroli et al., 2014).

The target estimates are the totals of the population by age class in each Swiss region. In this case, the G target variables will be:

Y1: number of people aged between 0 and 19,
Y2: number of people aged between 20 and 39,
Y3: number of people aged between 40 and 64,
Y4: number of people aged 65 and over.

We consider 6 auxiliary variables, chosen because they relate in some way to the use of land area, and converted to categorical variables using the same k- means clustering method as the iris data set example:

X1: Classes of total population in the municipality. 18 categories.
X2: Classes of wood area in the municipality. 3 categories.
X3: Classes of area under cultivation in the municipality. 3 categories.
X4: Classes of mountain pasture area in the municipality. 3 categories.
X5: Classes of area with buildings in the municipality. 3 categories.
X6: Classes of industrial area in the municipality. 3 categories.

There are 7 regions, which we treat as population domains of design to distinguish them from the design strata, replicating the experiment outlined in (Barcaroli et al., 2014). The number of non-empty atomic strata is 641 in the population. We set the

minimum population size of stratum to be 2, and the maximum number of iterations to be 400. The results for Sample Size and Strata after 30 experiments each with 400 iterations are summarised in figure 7.3 below.

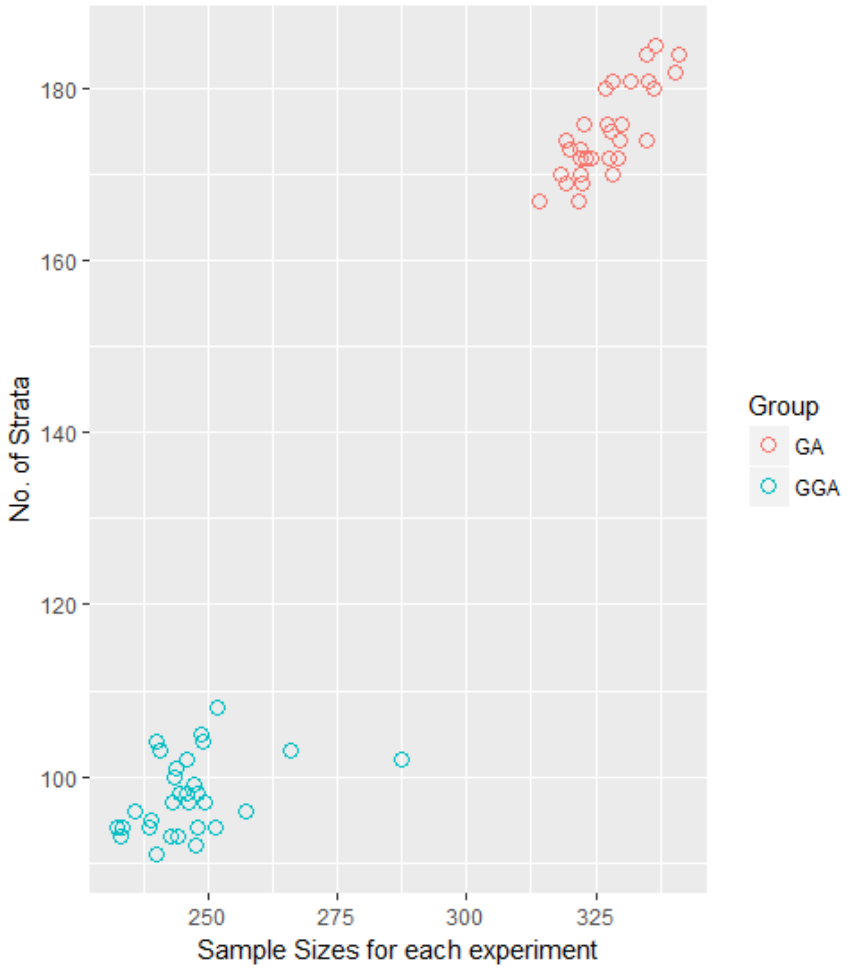


Figure 7.3: *Scatterplot of Results for Strata v Sample size for GA and GGA after 30 experiments*

Figure 7.3 clearly shows that the GGA returns a smaller sample size in comparison with the GA for these settings. The median for the GGA, 246, is 25% lower than that for the GA, 328.

7.4.3 2015 American Community Survey Public Use Microdata

The United States has been conducting a decennial census since 1790. In the 20th century censuses were split into long and short form versions. A subset of the population was required to answer the longer version of the census, with the remainder answering the shorter version. After the 2000 census the longer

questionnaire became the annual American Community Survey (ACS) (US Census Bureau, 2013).

The sampling frame for the ACS is a sample from the Census Bureau's Master Address File (MAF), which is a directory of residential and selected non-residential addresses in the United States and Puerto Rico. The MAF is not publicly available.

Independent samples of housing units (HU) or group quarters (GQ) are selected from each of the *"counties and county equivalents in the U.S., including the District of Columbia, as well as for each of the 78 municipalities in Puerto Rico. . . Each county sub-frame is a representative sample of addresses in the county. We assign these sub-frames to specific years and rotate them annually."*(Bureau, 2017b).

These samples combine into a representative sample of 3.5 million from the population. The sampling per county, county equivalent or municipality has two phases. The first phases assign blocks to each of 16 sampling strata from which samples are selected at pre-determined sampling rates, which vary according to strata. The second phase selects a sample of non-response for Computer Assisted Personal Interviewing (CAPI). The 16 sampling strata are determined by Smallest Entity Measure of Size (SEMOS) and Tract Measure of Size (TMOS) (Bureau, 2017b).

The actual 2015 ACS Public Use Microdata Sample (PUMS) file is a sample of actual responses to the American Community Survey (ACS) representing 1% of the US population. The dataset contains features which represent nearly every question in the survey. There are also new features which are derived from multiple survey responses after the survey. The PUMS sample contains 1,496,678 records each of which represents a unique HU or GQ. There are 235 features. The full data dictionary is available in US Census Bureau (2016).

The PUMS data contain cases from nearly every town and county in the country, but these are not identified in the dataset. As such, we cannot sample by county. Nevertheless, the PUMS data are useful in that we can construct strata to sample by state using target features and correlated features. This allows us to test the performance of the two algorithms on a larger dataset to the examples provided above.

The data contain records for which there are missing values for some variables. We would like to start with a complete sampling frame and our approach to deal with missing values is as follows.

We selected the following to be target variables:

1. Household income (past 12 months)
2. Property value
3. Selected monthly owner costs
4. Fire/hazard/flood insurance (yearly amount).

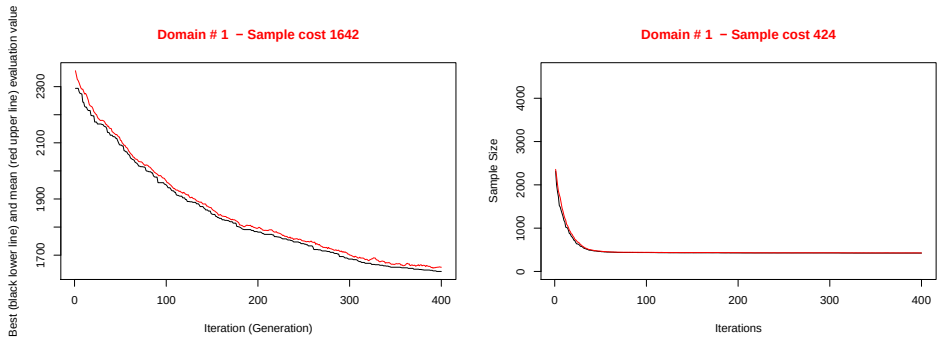
and the following auxiliary variables:

1. Units in structure
2. Tenure
3. Work experience of householder and spouse
4. Work status of householder or spouse in family households
5. House heating fuel
6. When structure first built.

Once the target and auxiliary features were selected, a subset of the PUMS data for which all values were present for all selected features was then read into R. This subset had 619,747 records. We use the 51 states (based on census definitions) as domains. Finally SERIALNO (Housing unit/GQ person serial number) was set as the optional unique identifier feature - although this does not form part of the atomic strata.

In keeping with the above analysis on the *swissmunicipalities* data 30 experiments were conducted for each algorithm. Similarly, the number of iterations was set to 400. Also, and as indicated in section 7.4 the population size was set to 20, the upper limits for the Coefficients of Variation for the target features were set to 0.05, elitism was set to be 20%, and the default mutation probability was used.

In the convergence plots of Figure 7.4, the black line represents the best or lowest sample size for the chromosome population in each iteration, whereas the red line represents the mean sample size for the chromosome population in each iteration.



(a) GA **(b)** GGA

Figure 7.4: *Convergence plots for Sample Size after the 1st experiment for GA and GGA. Note the different scales on the vertical axes*

The GA appears to be reducing the sample size steadily but does not appear to have reached a local minimum after 400 iterations. The GGA appears to have reached a local or global minimum very quickly.

7.4.4 Kaggle Data Science for Good challenge Kiva Loans data

The online crowdfunding platform kiva.org provided a data set of loans issued to people living in poor and financially excluded circumstances around the world over a two year period for a Kaggle Data Science for Good challenge. The data set has 671,205 unique records. We selected these target variables:

1. term in months
2. lender count
3. loan amount

and the following auxiliary variables:

1. sector
2. currency
3. activity
4. region
5. partner id

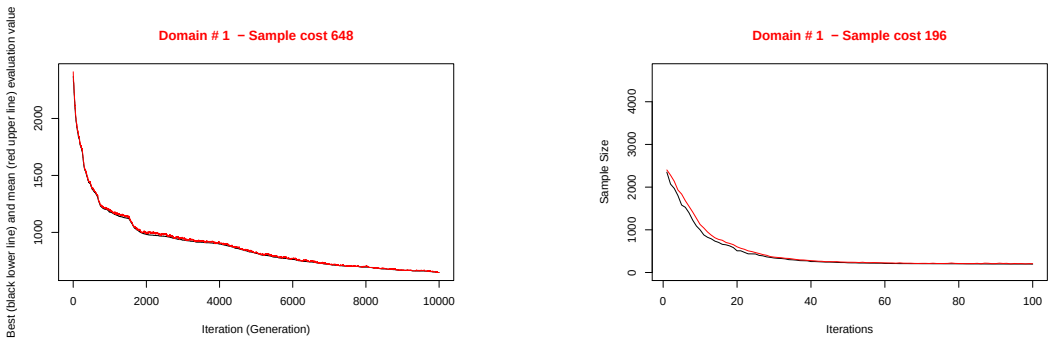
to create atomic strata. For these variables we removed any records with missing values. We then proceeded to remove any countries with less than 10 records from the sampling frame. This resulted in a sampling frame with 614,361 records. We

again used 100 iterations, along with the above mentioned starting hyperparameters.

Table 7.4.4: *Sample size and strata for the Kiva Loans data from the GA and the GGA after 100 iterations*

GA		GGA		Reduction	
Sample size	**Strata**	**Sample size**	**Strata**	**Sample size**	**strata**
78,018	43,030	11,963	1,793	84.67%	95.83%

Table 7.4.4 shows an 84.67% reduction in sample size and a 95.83% reduction in the number of strata after 100 iterations. Figure 7.5 shows that for the same starting chromosome population size for Domain 1 of the Kiva Loans data set, the GGA attained a good sample size in less than 100 iterations, but after 10,000 iterations the GA had not converged and the sample size was still much higher than the GGA.



(a) GA **(b)** GGA

Figure 7.5: *Convergence plots for Sample Size for the 1st Domain for GA (10,000 iterations) and GGA (100 iterations) in the Kiva Loans data set experiment. Note the different scales on the vertical and horizontal axes*

7.4.5 UN Commodity Trade Statistics data

Kaggle also hosts a copy of the UN Statistical Division Commodity Trade Statistics data. Trade records are available from 1962. We took a subset of data for the year 2011 and removed records with missing observations. This resulted in a data set with 351,057 records. We selected the following target variable:

1. trade_usd

which refers to the value of trade in USD (US dollars), and the following auxiliary variables:

1. commodity
2. flow

3. category

The variable commodity is a categorical description of the type of commodity, e.g. Horses, live except pure-bred breeding. The variable flow describes whether the commodity was an import, export, re-import or re-export. The variable category describes the category of commodity, e.g. silk or fertilisers. The 171 categories of country or area were selected as domains.

Table 7.4.5: *Sample size and strata for the UN Commodity Trade Statistics data from the GA and the GGA after 100 iterations*

GA		GGA		Reduction	
Sample size	**Strata**	**Sample size**	**Strata**	**Sample size**	**strata**
288,638	191,000	84,181	16,555	70.84%	91.33%

7.4.6 2000 US census data

The Integrated Public Use Microdata Series extract is a 5% sample of the 2000 US census data (Ruggles et al., 2017). The file contains 6,184,483 records. The US Census Data will be very similar to the ACS data as the latter is an annual version of the former. But for this experiment we selected different target and auxiliary variables. The single target variable in this test is usually a key focus of household surveys:

1. Total household income

We used the following information as auxiliary variables (note these are variables which are likely available in administrative data):

1. Annual property insurance cost
2. Annual home heating fuel cost
3. Annual electricity cost
4. House value

The house value variable (VALUEH) reports the midpoint of house value intervals (e.g. 5,000 is the midpoint of the interval of less than 10,000), so we have treated it as a categorical variable. As with the 2015 ACS Public Use Microdata Sample (PUMS) dataset we have taken a subset for which all values are present. For example we have removed all records where values for total household income are not applicable. This has resulted in a subset with 627,611 records. The domain for this experiment was Census region and division. As before we use the same starting hyperparameters.

Table 7.4.6: *Sample size and strata for the 2000 US census data by region and division from the GA and the GGA after 100 iterations*

	Sampling frame		GA solution		GGA solution	
Division	**Sampling Units**	**Atomic Strata**	**Sample sizes**	**Strata**	**Sample sizes**	**Strata**
New England	116,045	87,084	81,012	52,628	376	58
Middle Atlantic	183,543	138,470	130,862	86,002	416	75
East North Central	65,480	58,055	53,075	35,794	327	42
West North Central	31,408	29,413	26,525	18,248	324	38
South Atlantic	97,189	83,357	76,716	51,457	440	49
East South Central	21,631	20,429	18,256	12,500	451	62
West South Central	22,582	20,919	18,750	12,730	407	39
Mountain	26,765	25,041	22,161	14,791	351	30
Pacific	62,968	54,864	50,136	33,653	358	49
Total	**627,611**	**517,632**	**477,493**	**317,803**	**3,446**	**442**

The results show a sample size of 3,446 (which is an aggregation of the rounded samples for each Census region and division) for the GGA and a sample size of 477,493 for the GA after 100 iterations.

7.5 Selection of the target and auxiliary variables

Target variables are those of interest to a survey, or more broadly speaking, an analysis. Auxiliary variables should be correlated to the target variables, so that a stratified sampling approach can be more efficient. There are different approaches for determining a subset of variables correlated to the target variable variables, e.g. correlation and covariance coefficients, regression, plots and visual inspection, etc.

There are also approaches which utilise knowledge of the data and survey requirements, for example information for the target variables may be required to be reported by certain categories, e.g. business turnover is Nace code and enterprise size class, and household income could similarly be reported by region, tenure and household composition (age group of members). These categories are often used to construct strata. We discuss approaches for variable selection in chapter 5.

We outline the reasons for selection of the auxiliary variables for the six experiments below.

1. iris data set. For comparison purposes, we select Sepal Length and Species auxiliary variables as they were utilised the iris dataset experiment in Ballin and Barcaroli (2013).
2. Swissmunicipalities data set. Again for comparison, we use the Swissmunicipalities experiment in Ballin and Barcaroli (2013) as a reference point. The

target estimates are the totals of the population by age class in each Swiss region. The auxiliary features were chosen because they relate in some way to the use of land area.

The reason for selecting the auxiliary variables in the following experiments is that they would be of interest, as possible stratification variables, to survey designers.

4. 2015 American Community Survey Public Use Microdata. We selected BLD (Units in structure), TEN (Tenure), WKEXREL (Work experience of householder and spouse), WORKSTAT (Work status of householder or spouse in family households), HFL (House heating fuel), YBL (When structure first built).
5. Kaggle Data Science for Good challenge Kiva Loans data. We selected sector, currency, activity, region and partner id.
6. UN Commodity Trade Statistics data. Likewise, we selected commodity, flow and category.
7. 2000 US census data. We chose property insurance cost, Annual home heating fuel cost, Annual electricity cost and House value.

Although, they were selected using non computational methods, the efficient sample sizes attained with the above auxiliary variables, provide in an indication that they are well correlated to the target variables. However, in the continuous data experiments which follow in chapters 8, 9, and 10, the auxiliary variables are each exactly positively correlated to at least one target variable. These achieve more efficient sample sizes, again.

7.6 An improved Bethel implementation

Our GGA was proposed and developed so that it would work with the rest of the functions in *SamplingStrata*. Therefore the rest of the functions in the package remained unchanged. This includes the *bethel.r* function which evaluates the fitness of chromosomes in every iteration and is computationally expensive. For instance, for the PUMS data set the experiment took approximately 30 days for either GA or GGA with 100 iterations. The exact times are not reported because we used three different computers, and we addressed processing times in subsequent work, namely chapters 8, 9 and 10.

As mentioned, these experiments were run on three different computers, one with Windows 10 and the other two with Ubuntu 14.04. Two computers were laptops

and one was a desktop PC. All computers had 8GB RAM. In each of the experiments only one core was used.

We searched for performance bottlenecks in *bethel.r* using the R *lineprof* package. Our analysis of results suggested that the function within *bethel.r* called *chromy* appears to take the bulk of computational time. A further examination reveals that *chromy* contains a while loop with a default setting of 200 iterations. Furthermore *bethel.r* itself can be run on each chromosome in any chromosome population on a data set of any functional size (which we have the computation power to process) for any number of iterations. Bigger data sets will take longer to process. We expected that performance would be improved by converting the *bethel.r* algorithm into *C++* then integrating that into *R* using the *Rcpp* package (Eddelbuettel, 2013).

Table 7.6.7: *Performance comparison for the above data sets using the R and Rcpp versions of the Bethel-Chromy algorithm after 100 iterations*

dataset	Records	Domains	Atomic Strata	Bethel/ seconds	BethelRcpp/ seconds	Speed-up Factor
iris	150	1	8	0.0027	0.0001	18.76
swissmunicipalities	2,896	7	641	0.0999	0.0107	9.29
American Community Survey 2015	619,747	51	123,007	565.2785	47.8582	11.81
US Census Data 2000	627,611	9	517,632	2.6868	1.3037	2.06
Kiva Loans Data	614,361	73	84,897	826.2977	82.8945	9.97
UN Commodity Trade Data 2011	351,057	171	350,895	139.7498	87.5559	1.60

Table 7.6.7 shows the median time taken to run the Bethel algorithm per solution (computed as a function of total CPU time), one hundred times for the data sets we used to conduct our analysis. Our results confirm that the *C++* version of Bethel is faster than the R version. The speed up could make a practical difference in the number of iterations that can be run in *SamplingStrata* due to the processing times required for *bethel.r*. Performance will vary according to the size and complexity of the problem.

The speed up is achieved because *C++* enables communication at a lower level with the computer than R. However, it is also due to the complexity of the analysis conducted in each for loop as well as the fact that larger data will restrict the available memory.

The speed-up factors between the *R* and *C++* version versions differ between experiments, mainly for the following data characteristics:

- The number of target and auxiliary variables.
- The number of domains and the number of atomic strata in each domain.

- The hyperparameter settings and their influence on the number of solutions evaluated
- The number of cores available versus the number of domains.

Recall that in these experiments, only one core was utilised and, where there were more than one domains, they were processed sequentially

It should also be noted that the *C++* version of Bethel was compared with the R version as two stand alone functions. The performance of the *C++* version of Bethel within the GGA is not compared with that of the *R* version in the GA. This would be part of a larger project to create a *C++* version of the *SamplingStrata* package and integrating it into R.

7.7 Conclusion

We created a GGA as an alternative to the existing *SamplingStrata* GA in R. We then compared the two algorithms using a number of data sets. Due to the use of genetic operators, the GGA compares favourably with the GA at finding better solutions and meeting constraints on smaller data sets, but significantly outperforms the GA on larger data sets where the number of iterations was restricted. This is useful for data sets where the number of iterations has to be constrained owing to computational burden. We have also reported faster processing times by integrating the *bethel.r* function with *C++* using the *Rcpp* package.

Chapter 8

A simulated annealing algorithm

8.1 Introduction

We have seen in chapters 2 and 7 that a number of heuristic algorithms have been developed to search for optimal or near optimal solutions, for both univariate and multivariate scenarios of the joint stratification and sample allocation problem. This includes the hierarchichal algorithm proposed by Benedetti et al. (2008), the genetic algorithm proposed by Ballin and Barcaroli (2013) and the grouping genetic algorithm proposed in chapter 7. Although effective, the evaluation function in these algorithms can be costly in terms of running time.

We now add to this work with a simulated annealing algorithm (SAA) (Černỳ, 1985; Kirkpatrick et al., 1983). SAAs have been found to work well, where there are many local minima and finding an approximate global solution in a fixed amount of computation time is more desirable than finding a precise local minimum (Takeang and Aurasopon, 2019). We present a SAA to which we have added delta evaluation (see section 8.4) to take advantage of the similarity between consecutive solutions and help speed up computation times.

We compared the performance of the SAA on atomic strata with that of the grouping genetic algorithm (GGA) in the *SamplingStrata* package (Ballin and Barcaroli, 2020). This algorithm implements the grouping operators described in chapter 7. To do this, we used sampling frames of varying sizes containing what we assume to be completely representative details for target and auxiliary variable columns.

Further to the suggestion of a *Survey Methodology* reviewer, we subsequently compared the SAA with a classical genetic algorithm (CGA) used by Ballin and

Barcaroli (2020) on continuous strata. In both sets of experiments, we used an initial solution created by the k-means algorithm (Hartigan and Wong, 1979) in a two-stage process (see section 8.2.2 for more details).

Section 8.2 introduces the SAA and motivates the addition of delta evaluation as a means to improve computation time. Two-stage simulated annealing is also discussed. Chapter 3 of the thesis describes the objective function and evaluation algorithm. Section 8.3 provides an outline of the SAA. Section 8.4 presents the improved SAA with delta evaluation. Section 8.5 provides a comparison of the performance of the SAA with the GGA using an initial solution and fine-tuned hyperparameters. Section 8.6 then provides details of the comparison of the SAA with the genetic algorithm in Ballin and Barcaroli (2020) on continuous strata. Section 8.10 presents the conclusions. Sections 8.5.6 and 8.6.2 outline the process of fine-tuning the hyperparameters for both comparisons and section 8.9.17 contains the computer specifications. The appendix - sections A.1 contains background details on precision constraints and tables describing the hyperparameters, for each of the 20 tests, for each algorithm in both comparisons.

8.2 Background information

In section 8.2 we provide some background information on simulated annealing algorithms, and discuss the process of two-stage approach simulated annealing.

8.2.1 Simulated annealing algorithms

The basic principle of the SAA (Černỳ, 1985; Kirkpatrick et al., 1983) is that it can accept solutions that are inferior to the current best solution in order to find the global minima (or maxima). It is one of several stochastic local search algorithms, which focus their attention within a local neighbourhood of a given initial solution (Cortez, 2014), and use different stochastic techniques to escape from attractive local minima (Hoos and Stützle, 2004).

Based on physical annealing in metallurgy, the SAA is designed to simulate the controlled cooling process from liquid metal to a solid state (Luke, 2013). This controlled cooling uses the temperature parameter to compute the probability of accepting inferior solutions (Cortez, 2014). This acceptance probability is not only a function of the temperature, but also the difference in cost between the new solution and the current best solution. For the same difference in cost, a higher temperature means a higher probability of accepting inferior solutions.

For a given temperature, solutions are iteratively generated by applying a small, randomly generated, perturbation to the current best solution. Generally, in SAAs, a perturbation is the small displacement of a randomly chosen particle (Van Laarhoven and Aarts, 1987). In the context of our problem, we take perturbation to mean the displacement (or re-positioning) of q (generally $q = 1$) randomly chosen atomic strata from one randomly chosen stratum to another (see section 8.3.1 for more details).

With a perturbation, the current best solution transitions to a new solution. If a perturbation results in a lower cost for the new solution, or if there is no change in cost, then that solution is always selected as the current best solution. If the new solution results in a higher cost, then it is accepted at the above mentioned acceptance probability. This acceptance condition is called the Metropolis criterion (Metropolis et al., 1953). This process continues until the end of the sequence, at which point the temperature is decremented and a new sequence begins.

If the perturbations are minor, then the current solution and the new solution will be very similar. Indeed, in our SAA we are assuming only a slight difference between consecutive solutions owing to such perturbations (see section 8.3.1 for more details). For this reason we have added delta evaluation, which will be discussed further in section 8.4, to take advantage of this similarity and help improve computation times.

Accordingly, and as mentioned in the introduction, we present a SAA with delta evaluation and compare it with the GGA when both are combined with an initial solution. We also compare it with the classical genetic algorithm used by Ballin and Barcaroli (2020) on continuous strata. We provide more background details on initial solutions in section 8.2.2 below.

8.2.2 Two-stage simulated annealing

A two stage simulated annealing process, where an initial solution is generated by a heuristic algorithm in the first stage, has been proposed for problems such as the *cell placement problem* (Grover, 1987; Rose et al., 1988) or the *graph partitioning problem* (Johnson et al., 1989).

Lisic et al. (2018) combined an initial solution, generated by the k-means algorithm, with a simulated annealing algorithm, for a problem similar in nature to this problem, but where the sample allocation as well as strata number are fixed, and the algorithm searches for the optimal arrangement of sampling units between strata.

The simulated annealing algorithm used by Lisic et al. (2018) starts with an initial solution (stratification and sample allocation to each stratum) and, for each iteration, generates a new candidate solution by moving one atomic stratum from one stratum to another and adjusting the sample allocation for that stratification. Each candidate solution is then evaluated to measure the coefficient of variation (CV) of the target variables and is accepted, as the new current best solution, if its objective function is less than the preceding solution. Inferior quality solutions are also accepted at a probability, ρ, which is a function of a tunable temperature parameter and the change in solution quality between iterations. The temperature cools, at a rate which is also tunable, as the number of iterations increases.

Following this work, Ballin and Barcaroli (2020) recommended combining an initial solution, generated by k-means, with the grouping and classical genetic algorithms. They demonstrate that the k-means algorithm provides better starting solutions when compared with the starting solution generated by a stochastic approach. We also combine a k-means initial solution with the SAA in the experiments described in sections 8.5 and 8.6.

8.3 Outline of the simulated annealing algorithm

The SAA with delta evaluation is described (as an R function) in algorithm 6 below. We then describe the heuristics we have used in the SAA. Delta evaluation is explained in more detail in section 8.4.

Algorithm 6 Simulated annealing algorithm

function SIMULATEDANNEALING(S is the starting solution, f is the evaluation function (Bethel-Chromy algorithm), *best* is the current best solution, *BSFSF* is the best solution found so far, *maxit* is the maximum number of sequences, J is the length of sequence, T_{max} is the starting temperature, T_{min} is the minimum temperature, DC is the Decrement Constant, $L_{max\%}$ is a % of L (number of atomic strata), $P(H+1)$ is the probability of a new stratum, $H+1$, being added)
  $T \leftarrow T_{max}$
  $best \leftarrow$ S
  $Cost(best) \leftarrow f(best)$ ▷ using Bethel-Chromy algorithm
  while $i < maxit$ && $T > T_{min}$ **do**
    if RANDOM$(0,1) \leq 1/J$ **then**
      for $l = 1$ to L **do**
        if RANDOM$(0,1) \leq P(H+1)$ **then**
          move atomic stratum l to new stratum $H+1$ ▷ see section 8.3.3
        end if
      end for
    end if
    for $j = 1$ to J **do**
      if $i = 1$ & $j = 1$ **then** $q = L \times L_{max\%}$
      else if $i = 1$ & $j > 1$ **then** $q = ceiling(q \times 0.99)$ ▷ 0.99 is not tunable
      else if $i > 1$ **then** $q = 1$
      end if
      Randomly select h and h'
      $next \leftarrow$ **PERTURBATION**(*best*)
      ▷ Assign q atomic strata from h to h'
      $Cost(next) \leftarrow f(next)$ ▷ using delta evaluation
      $\Delta E \leftarrow$**COST**$(next)$ − **COST**$(best)$
      if $\Delta E \leq 0$ **then**
        $best \leftarrow next$
      else if RANDOM$(0,1) < e^{(-\frac{\Delta E}{T})}$ **then** ▷ Metropolis Criterion
        $best \leftarrow next$
      end if
      if best $<=$ *BSFSF* **then**
        $BSFSF \leftarrow best$
      end if
    end for
    $T \leftarrow T * DC$

  end while
  return *BSFSF*
end function

8.3.1 Perturbation

Consider the following solution represented by the stratification:

$$\{1,3\}, \{2\}, \{4,5,6\}$$

The integers within each stratum represent atomic strata. In perturbation, the new solution below is created by arbitrarily moving atomic strata, in this example $q = 1$, from one randomly chosen stratum to another.

$$\{1,3,2\}\ , \{\emptyset\}, \{4,5,6\}$$

The first stratum gains an additional atomic stratum $\{2\}$ to become $\{1,3,2\}$, whereas the middle or second stratum has been "emptied" (and is deleted), and there remains only two strata. Strata are only emptied when the last remaining atomic stratum has been moved to another stratum.

To clarify how this works in the algorithm: each solution is represented by

a vector of integers - atomic strata which have the same integer are in the same stratum. A separate vector of the unique integers in the solution represents the strata. For example, the first solution $\{1,3\}$, $\{2\}$, $\{4,5,6\}$ would be represented by the vector $[1 \quad 2 \quad 1 \quad 3 \quad 3 \quad 3]$ and the strata would be represented by the vector $[1 \quad 2 \quad 3]$. When the new solution is created, the second stratum has been removed and is no longer part of the solution. That is to say, the vector for the new solution is: $[1 \quad 1 \quad 1 \quad 3 \quad 3 \quad 3]$ and the strata vector is $[1 \quad 3]$. With stratum 2 removed, and for clarity, we rename stratum 3 to 2 so that this solution becomes: $[1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2]$, and the strata are now represented by the vector $[1 \quad 2]$. Strata $[1 \quad 2]$ will remain in any further solutions unless another stratum is "emptied" or a new stratum is added (see section 8.3.3 for more details).

8.3.2 Evaluation and acceptance

Each new solution is evaluated using the Bethel-Chromy algorithm and the Metropolis acceptance criterion is applied. If accepted, the new solution differs from the previous solution only by the above mentioned perturbation. If it is not accepted, we continue with the previous solution, and again try moving q randomly chosen atomic strata between two randomly selected strata.

8.3.3 Sequences and new strata

This continues for the tunable length of the sequence, where a sequence of J solutions are perturbed. This should be long enough to allow the sequence to reach equilibrium. However, there is no rule to determine J. At the commencement of each new sequence, we have H strata in the current best solution. Also at the commencement of each new sequence, with a fixed probability of $1/J$, an additional stratum is added. However, there is scope to fine tune $1/J$. The low probability is intended to minimise the number of new strata added, as new strata should only be created in circumstances where:

- Due to the explorative properties of the algorithm, i.e. the q parameter described in section 8.3.1 and the acceptance of inferior solutions - see section 8.3.4, if as a result of the movement of atomic strata between strata (and the emptying of strata) the number of strata has become too few (i.e. the optimal stratification has a certain number of strata).
- Adding a new stratum would assist the search (i.e. in doing it enables escape from a local minimum, or alternatively the resulting arrangement of atomic strata within strata leads to a better quality solution).

If a new stratum is to be added, the SAA loops through each atomic stratum and moves it to a new stratum, which is called $H+1$, because each stratum is labelled sequentially from 1 to H (see section 8.3.1), at a tunable probability, $P(H+1)$. The algorithm runs for a tunable number of sequences, $maxit$.

8.3.4 Temperature

The temperature is decremented from a starting temperature, T_{max}, to a minimum temperature, T_{min}, or until $maxit$ has been reached. As we are starting with what we assume to be a near optimal solution (generated by the k-means clustering algorithm), we select T_{max} as no greater than 0.01 and we set T_{min} to be 1.0×10^{-11}.

This is to allow for the advanced nature of the search, and allows the algorithm to focus more on the search for superior solutions, with an ever-reducing probability of accepting inferior solutions. However, a low temperature, T, does not always equate to a low probability of acceptance.

Small positive differences in solution quality (where the new solution has a marginally inferior quality to the current best solution), ΔE, occur often because we are starting with a good quality initial solution. Figure 8.1 demonstrates the probability of such solutions being accepted, $e^{(-\frac{\Delta E}{T})}$, increases the smaller this difference becomes for the same T. Nonetheless, figure 8.1 also demonstrates that for the same changes in solution quality as the T decreases, the probability also decreases (and it behaves increasingly like a hill climbing algorithm).

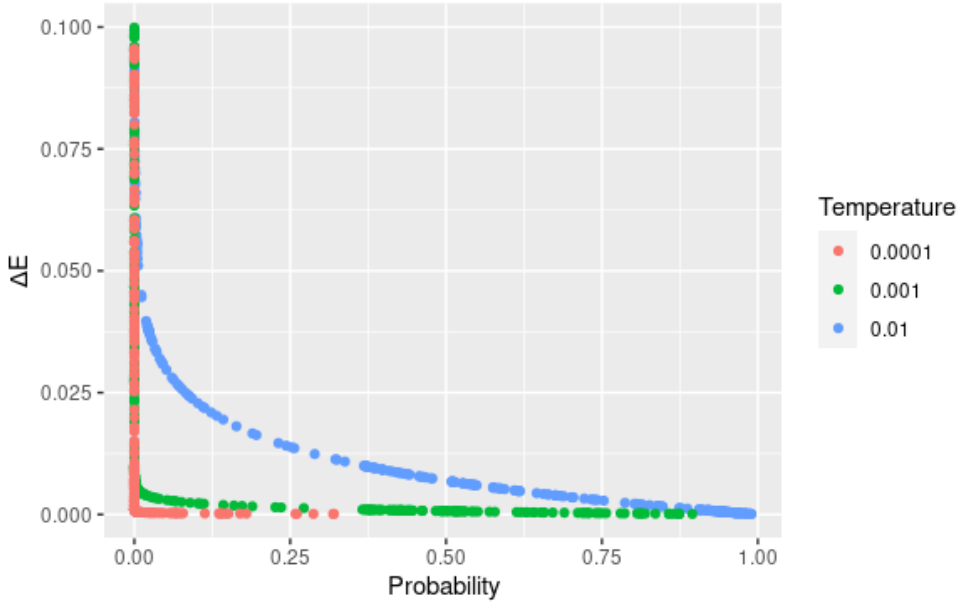


Figure 8.1: *Probability of accepting an inferior solution as a function of ΔE and T*

8.4 Improving the performance of the simulated annealing algorithm using delta evaluation

As outlined earlier, the only difference between consecutive solutions is that q atomic strata have been moved from one group into another. The starting value for q can be varied for the first sequence, where we have added the option of setting $q > 1$ and reducing q for each new solution in that sequence until $q = 1$. The reason for this is that, where $q > 1$, the increased size of the perturbation can help reduce the number of strata. For this purpose, there is an option of setting the starting value for q as a tunable percentage of the solution size / the number of atomic strata, L, to be partitioned. After the first sequence $q = 1$.

Furthermore, as the strata are mutually exclusive, this movement of q atomic strata from one stratum to another does not affect the remaining strata in any way. Ross et al. (1994) introduce a technique called delta evaluation, where the evaluation of a new solution makes use of previously evaluated similar solutions, to significantly speed up evolutionary algorithms/timetabling experiments. We use the similar properties of two consecutive solutions to apply delta evaluation to the SAA.

Conceptually, even though we have added a facility to set $q > 1$ for a portion of the first sequence, delta evaluation implies a minimal change between solutions. We define this minimal change as $q = 1$ with the anticipation that this facilitates that greatest speed-up in evaluation times (however, there is also scope to trade off between $q > 1$ and the number of solutions evaluated). It follows, therefore, that in the first sequence q should be kept low and the reduction to $q = 1$ should be swift.

The algorithm requires the means and variances for each stratum in order to calculate the sample allocation. However, we use the information already calculated for the remaining $H - 2$ strata, and simply calculate for the two strata affected by the perturbation. Thus, the computation for the means and variances of the H strata is reduced to a mere subset of that otherwise required.

Now recall that the Bethel-Chromy algorithm starts with a default value for each α_g, and uses gradient descent to find a final value for each α_g. This search continues up to when the algorithm reaches a minimum step-size threshold, or alternatively exceeds a maximum number of iterations. This minimum threshold is characterised by ϵ, which is set as 1.0×10^{-11} in Ballin and Barcaroli (2020), and the maximum number of iterations is 200. We make the assumption that this search will be substantially reduced if we use the α_g values from the evaluation of the current solution as a starting point for the next solution.

The above two implementations of delta evaluation result in a noticeable reduction in computation times as demonstrated in the experiments described below.

8.5 Comparing the performance of the two algorithms

8.5.1 Evaluation plan

In this section, we outline the comparison of the performance of the grouping genetic algorithm with the simulated annealing algorithm. We used a number of data sets of varying sizes in these experiments. There are a number of regions in each data set (labelled here as domains). An optimal stratification and minimum sample allocation was selected for each domain.

The sum of the samples for all domains provides the total sample size. The sample size, or cost of the solution, defines the solution quality. For more details on domains refer to chapter 3. The aim of these experiments was to consider whether the SAA can attain comparable solution quality with the GGA in less computation time per solution thus resulting in savings in execution times.

However, we also compared the total execution times as this is a consequence of the need to train the hyperparameters for both algorithms. More details are available in the appendix - see section A.1.2.

We tabulate the results of these experiments in section 8.5.7 where for comparison purposes we express the SAA results as a ratio of those for the GGA.

8.5.2 Number of strata

The sample size, or cost of the solution, defines the solution quality. However, the number of strata, H, is an indicator of whether the algorithm has found the optimal solution. However, there is no known method of knowing H in advance, unless each candidate solution has been evaluated. As mentioned earlier, more often than not this is not practical. This is further complicated by the fact that there could be a number of different H.

Both algorithms can reduce H when transitioning to new solutions. The reduced size of H reduces the search space (as H is the upper limit of strata). This usually continues until further reductions do not reduce the sample size any further. This means that in order to improve on the best solution found so far, after a certain

point, the problem becomes more about finding the arrangement of similar atomic strata within a fixed number of strata. We report the number of strata as in indication of the reduction of H. However, a solution with less strata than another solution may not be the better solution.

This is illustrated referencing the *iris* dataset example in (Ballin and Barcaroli, 2013), in which for set target and auxiliary variables each valid solution was evaluated and it was known that there are 3 possible 'optimal solutions', each containing either 3, 4 or 5 strata. Simply put, if the algorithm reduces the number of strata in a solution to 2, it will not find the minimum sample allocation without creating a new stratum.

As such, it is possible that the algorithms decrease the number of strata to a point where it is no longer possible to attain an optimal arrangement of atomic strata so that a minimum sample can be selected.

8.5.3 Comparing the number of solutions generated

After the first iteration the GAA retains the elite solutions, E, from the previous iteration. E is calculated by the product of the elitism rate (the proportion of the chromosome population which are elite solutions), E_R, and the chromosome population size (the number of candidate solutions in each iteration), N_P. As the E solutions have already been evaluated they are not evaluated again.

For this reason, we compared the evaluation times for the evaluated solutions in the GGA with all those of the SAA. For the GGA, the total number of evaluated solutions, N_{GGAsol}, is a function of the number of domains, D, the chromosome population size, the non-elite solutions (calculated by the product of $1 - E_R$ and N_P), and the number of iterations, I. For more details on the implementation of GGAs (e.g. elite solutions, elitism rate, chromosome population) we refer the reader to (Falkenauer, 1998).

$$N_{GGAsol} = (D \times (N_P + (N_P \times (1 - E_R) \times (I - 1)))) \tag{8.1}$$

For the simulated annealing algorithm, the maximum number of solutions, N_{SAAsol}, is the number of domains, D, by the number of sequences, $maxit$, by the length of sequence, J. Recall that the SAA also stops if the minimum temperature has been reached - hence we refer to the *maximum* number of solutions rather than the *total*. For comparability purposes however, because the temperature is decremented only at the end of each sequence and we have a small number of sequences in the

experiments below we assume the full number of solutions has been generated.

$$N_{SAAsol} = D \cdot maxit \cdot J \tag{8.2}$$

equation

8.5.4 Data sets, target and auxiliary variables

Table 8.5.1 provides a summary by data set of the target and auxiliary variables.

Table 8.5.1: *Summary by data set of the target and auxiliary variables*

Data set	Target variables	Description	Auxiliary variables	Description
SwissMunicipalities	Surfacebois	wood area	POPTOT	total population
	Airbat	area with buildings	Hapoly	municipality area
American Community Survey, 2015	HINCP	Household income past 12 months	BLD	Units in structure
	VALP	Property value	TEN	Tenure
	SMOCP	Selected monthly owner costs	WKEXREL	Work experience of householder and spouse
	INSP	Fire/hazard/flood insurance yearly amount	WORKSTAT	Work status of householder or spouse in family households
			HFL	House heating fuel
			YBL	When structure first built
US Census, 2000	HHINCOME	total household income	PROPINSR	annual property insurance cost
			COSTFUEL	annual home heating fuel cost
			COSTELEC	annual electricity cost
			VALUEH	House value
Kiva Loans	term_in_months	duration for which the loan was disbursed	sector	high level categories, e.g. food
	lender_count	the total number of lenders	currency	currency of the loan
	loan	the amount in USD	activity	more granular category, e.g. fruits & vegetables
			region	region name within the country
			partner_id	ID of the partner organization
UN Commodity Trade Statistics data	trade_usd	value of the trade in USD	commodity	type of commodity e.g. "Horses, live except pure-bred breeding"
			flow	whether the commodity was an import, export, re-import or re-export
			category	category of commodity, e.g. silk or fertilisers

The target and auxiliary variables for the Swiss Municipalities data set were selected based on the experiment described in Ballin and Barcaroli (2020). Accordingly, *POPTOT* and *HApoly* were converted into categorical variables using the k-means clustering algorithm. However, we used more domains and iterations in our experiment. More information on this data set is provided by Barcaroli et al. (2014).

For the remaining experiments we selected target and auxiliary variables which we deemed likely to be of interest to survey designers. Further details on the American Community Survey, 2015 (US Census Bureau, 2016), the US Census, 2000 (Ruggles et al., 2017), Kiva Loans (Kiva, 2018), and the UN commodity trade

statistics data (UN, 2017) metadata are available in chapter 7.

A further summary by data set of the number of records and atomic strata, along with a description of the domain variable, is provided in table 8.5.2 below.

Table 8.5.2: *Summary by data set of the number of records and atomic strata and a description of the domain variable*

Data set	**Number of records**	**Number of atomic strata, L**	**Domain variable**
Swiss Municipalities	2,896	579	REG
American Community Survey, 2015	619,747	123,007	ST (the 51 states)
US Census, 2000	627,611	517,632	REGION
Kiva Loans	614,361	84,897	country code
UN Commodity Trade Statistics data	352,078	351,916	country or area

8.5.5 Hyperparameters for the grouping genetic algorithm and simulated annealing algorithm

Tables 8.5.3 and 8.5.4 below outline the number of domains in each experiment, along with number of iterations and chromosome population size for the grouping genetic algorithm and along with the number of sequences, length of sequence, and starting temperature for the simulated annealing algorithm. The ranges for the hyperparameter tuning were set on a trial and error basis. While they demonstrate the explorative nature of the GGA and the exploitative nature of the SAA, the finetuning of the hyperparameters should not be taken an exhaustive optimisation. Instead, this process was intended to find comparable near optimal solution qualities, while demonstrating the faster processing times of the SAA.

Section 8.5.6 provides details on the fine-tuning of the hyperparameters. For more details on the hyperparameters of the GGA refer to chapter 7 and of the SAA refer to sections 8.2.1 and 8.3.

Table 8.5.3: *Summary by data set of the hyperparameters for the grouping genetic algorithm for each domain*

Data set	**Domains**	**Number of iterations, I**	**Chromosome population size, N_p**	**Mutation chance**	**Elitism rate, E_R**	**Add strata factor**
Swiss Municipalities	7	4,000	50	0.0053360	0.4	0.0037620
American Community Survey, 2015	51	5,000	20	0.0008134	0.5	0.0610529
US Census, 2000	9	100	20	0.0000007	0.4	0.0000472
Kiva Loans	73	3,000	20	0.0007221	0.5	0.0685005
UN Commodity Trade Statistics data	171	1,000	20	0.0004493	0.3	0.0866266

Table 8.5.4: *Summary by data set of the hyperparameters for the simulated annealing algorithm for each domain*

Data set	Domains	Number of sequences, $maxit$	Length of sequence, J	Temperature, T	Decrement constant, DC	% of L for maximum q value, $L_{max\%}$	Probability of new stratum, $P(H+1)$
Swiss Municipalities	7	10	3,000	0.0000720	0.5083686	0.0183356	0.0997907
American Community Survey, 2015	51	3	3,000	0.0002347	0.6873029	0.0076477	0.0291729
US Census, 2000	9	2	2,000	0.0006706	0.5457192	0.0189395	0.0806919
Kiva Loans	73	5	2,000	0.0009935	0.7806557	0.0143925	0.0317491
UN Commodity Trade Statistics data	171	3	3,000	0.0007902	0.5072737	0.0234728	0.0013775

8.5.6 Fine-tuning the hyperparameters for the grouping genetic algorithm and simulated annealing algorithm

In order to fine-tune the initial parameters or *hyperparameters* we used sequential model-based optimization (Hutter et al., 2010). We first generated an initial design of hyperparameters from the value ranges described for the GGA in table 8.5.5 and in table 8.5.6 for the SAA below using the latin hypercube design method (McKay et al., 2000).

Table 8.5.5: *Ranges for fine-tuning the hyperparameters for the grouping genetic algorithm*

	Iterations			**Population size**		
Value type	**Discrete**			**Discrete**		
Value range	**Lower value**	**Upper value**	**Increments**	**Lower value**	**Upper value**	**Increments**
Swiss Municipalities	500	5,000	500	10	50	10
American Community Survey, 2015	1,000	5,000	1,000	10	20	10
Kiva Loans	1,000	3,000	1,000	10	20	10
UN Commodity Trade Statistics data	500	1,000	500	10	20	10
US Census, 2000	50	100	50	10	20	10

	Mutation chance		**Elitism rate, E_R**			**Add strata factor**	
Value type	**Numeric**		**Discrete**			**Numeric**	
Value range	**Lower value**	**Upper value**	**Lower value**	**Upper value**	**Increments**	**Lower value**	**Upper value**
Swiss Municipalities	0	0.10	0.1	0.5	0.1	0	0.1
American Community Survey, 2015	0	0.001	0.1	0.5	0.1	0	0.1
Kiva Loans	0	0.001	0.1	0.5	0.1	0	0.1
UN Commodity Trade Statistics data	0	0.001	0.1	0.5	0.1	0	0.1
US Census, 2000	0	0.000001	0.1	0.5	0.1	0	0.0001

Table 8.5.6: *Ranges for fine-tuning the hyperparameters for the simulated annealing algorithm*

	Number of sequences, maxit			**Length of sequence, J**		
Value type	**Discrete**			**Discrete**		
Value range	**Lower value**	**Upper value**	**Increments**	**Lower value**	**Upper value**	**Increments**
Swiss Municipalities	10	50	10	1,000	3,000	1,000
American Community Survey, 2015	1	3	1	1,000	3,000	1,000
Kiva Loans	1	5	1	1,000	2,000	1,000
UN Commodity Trade Statistics data	1	3	1	1,000	3,000	1,000
US Census, 2000	1	2	1	1,000	2,000	1,000

	Temperature, T		**Decrement constant, DC**		**% L for maximum q value, $L_{max\%}$**		**Probability of new stratum, P(H+1)**	
Value type	**Numeric**		**Numeric**		**Numeric**		**Numeric**	
Value Range	**Lower value**	**Upper value**	**Lower value**	**Upper value**	**Lower value**	**Upper value**	**Lower value**	**Upper value**
Swiss Municipalities	0	0.001	0.5	1	0.0001	0.025	0	0.1
American Community Survey, 2015	0	0.001	0.5	1	0.0001	0.025	0	0.1
Kiva Loans	0	0.001	0.5	1	0.0001	0.025	0	0.1
UN Commodity Trade Statistics data	0	0.001	0.5	1	0.0001	0.025	0	0.1
US Census, 2000	0	0.001	0.5	1	0.0001	0.025	0	0.1

As some of the hyperparameter value ranges were discrete, we used a random forest with regression trees to develop a surrogate learner model. After this, a confidence bound using a lambda value, λ, to control the trade-off between exploitation and exploration was used as the acquisition function. The focus search approach (see the methodology described in section 6.1) was used to optimise the acquisition function which, in turn, was used to propose the hyperparameters which were evaluated using the surrogate function (which is a cheaper alternative to using the GGA or SAA algorithms). From these, the most promising hyperparameters were then evaluated by the GGA or SAA and the hyperparameters and solution costs added to the initial design. The process was then repeated for a set number of iterations and the best performing hyperparameters and solution outcomes were selected. We implemented this using the *MBO* function with the parameters outlined in table 8.5.7. These are distinct from the parameters being fine-tuned, which are outlined in tables 8.5.5 and 8.5.6 above.

Table 8.5.7: *Parameters used in the* MBO *Function*

MBO parameters	**Value**
Initial Design size (Latin Hypercube Design method)	10
Iterations, number of	10
Number of Trees	500
Lambda, λ	5
Focus Search Points	1,000

As can be seen from the limited scope of the *MBO* function parameters this was

not an exhaustive fine-tuning of the hyperparameters for the GGA and SAA. The aim of these experiments was to consider whether the SAA can attain comparable solution quality with the GGA in less computation time per solution thus resulting in savings in execution times. However, we also compared the total execution times as this is a consequence of the need to train the hyperparameters for both algorithms.

Tables A.1.2 and A.1.3 outline the hyperparameters in each of the fine-tuning iterations. The first 10 sets (*Fine-tuning iteration No.* 1-10) of hyperparameters were randomly generated from the ranges laid out in tables 8.5.5 and 8.5.6. The ranges selected were identified using practical knowledge of the algorithms and data. The second 10 sets reflects the *MBO* function's attempts to learn the hyperparameters that best lead each algorithm towards the optimal solution using the previous solutions as a guide.

8.5.7 Results

As mentioned previously, we used an initial solution in each experiment that is created by the *KmeansSolution* algorithm (Ballin and Barcaroli, 2020). We then compared the performance of the algorithms in terms of average computation time (in seconds) per solution and solution quality. Table 8.5.8 provides the sample size, execution times and total execution times for the SAA and GGA.

Table 8.5.8: *Summary by data set of the sample size and evaluation time for the grouping genetic algorithm and simulated annealing algorithm*

	GGA			**SAA**		
Data set	**Sample size**	**Execution time (seconds)**	**Total Execution time (seconds)**	**Sample size**	**Execution time (seconds)**	**Total Execution time (seconds)**
Swiss Municipalities	128.69	753.82	10,434.30	125.17	248.91	8,808.63
American Community Survey, 2015	10,136.50	13,146.25	182,152.46	10,279.44	517.76	6,822.42
US Census, 2000	228.81	2,367.36	36,298.35	224.75	741.75	8,996.85
Kiva Loans	6,756.19	15,669.11	288,946.79	6,646.67	664.30	7,549.87
UN Commodity Trade Statistics data	3,216.68	6,535.97	88,459.22	3,120.07	1,169.26	12,161.80

The total execution time is the sum of the execution times for 20 evaluations of the GGA and SAA algorithms (by SMBO - see the methodology described in section 6.1) function in the R package *mlrMBO* (Bischla et al., 2017)) using 20 sets of selected hyperparameters (i.e. one set for each evaluation). Details on the precision constraints and hyperparameters for each experiment can be found in the appendix - section A.1. Table 8.5.9 expresses the SAA results as a ratio of those for the GGA.

Table 8.5.9: *Ratio comparison of the sample sizes, execution times, and total execution times for the simulated annealing algorithm and grouping genetic algorithm*

Data set	Sample size	Execution time	Total execution time
Swiss Municipalities	.97	.33	.84
American Community Survey, 2015	1.01	.04	.04
US Census, 2000	.98	.31	.25
Kiva Loans	.98	.04	.03
UN Commodity Trade Statistics data	.97	.18	.14

As can be seen, the sample sizes are similar. However, the SAA shows significantly lower execution and total execution times (which both represent total CPU times). When these experiments are run in parallel, for cases where there is a large number of domains, there may not be enough cores to cover all domains in one run. Indeed, it may take several parallel runs to complete the task, and this will affect mean evaluation time. The computer specifications are provided in table 9.13.1.

Table 8.5.10 shows the number of solutions evaluated by each algorithm to obtain the results shown in table 8.5.8. It also provides a ratio comparison of the average execution time (in seconds) per solution.

Table 8.5.10: *Number of solutions and ratio comparison of execution time between the simulated annealing algorithm and grouping genetic algorithm*

	Number of solutions evaluated	
Data set	**GGA**	**SAA**
Swiss Municipalities	840,140	210,000
American Community Survey, 2015	2,550,510	459,000
US Census, 2000	10,872	36,000
Kiva Loans	2,190,730	730,000
UN Commodity Trade Statistics data	2,395,026	1,539,000

	Average execution time per solution		
Data set	**GGA**	**SAA**	**Ratio**
Swiss Municipalities	0.0009	0.0012	1.3210
American Community Survey, 2015	0.0052	0.0011	0.2188
US Census, 2000	0.2177	0.0206	0.0946
Kiva Loans	0.0072	0.0009	0.1272
UN Commodity Trade Statistics data	0.0027	0.0008	0.2784

The above results indicate that the GGA has evaluated more solutions to find a solution of similar quality to the SAA in all cases, except for the *US Census, 2000* experiment. However, we also can see that the SAA takes less time to evaluate each solution in all cases except for the *Swiss Municipalities* experiment. The average execution time for each experiment can be considered in the context of the size of the data set, parallelisation, and the particular sets of hyperparameters used for the GGA and SAA. In addition to this, there is also memoisation in the evaluation

algorithm for the GGA, and the gains obtained by delta evaluation by the SAA. Ballin and Barcaroli (2020) use a memoisation function, which keeps a list of previously evaluated solutions- so that the are not evaluated again. There is a certain amount of overhead involved in checking the memoisation list.

Gains are more noticeable for larger data sets, because of the size of the solution and number of atomic strata in each stratum. As the strata get larger in size, the movement of q atomic strata from one stratum to another (where q is small) will have a smaller impact on solution quality and, therefore, the delta evaluation will be quicker.

We now visualise the results for both algorithms across each of the 20 fine tuning tests.

For the SwissMunicipalities data set finetuning tests, Figure 8.2, shows that the SAA typically finds a lower sample size than the GGA. It also does so, generally, in less time.

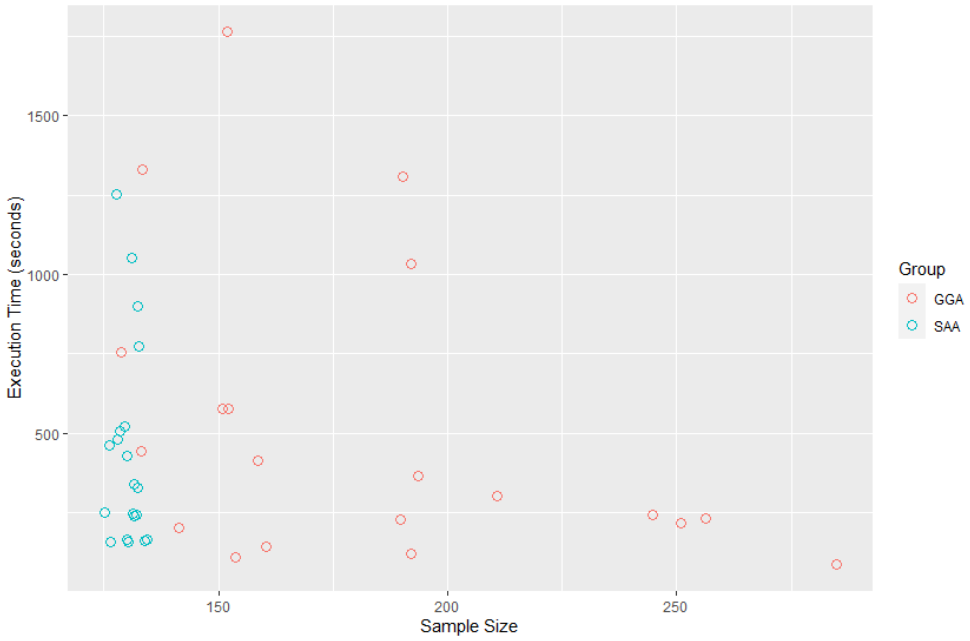


Figure 8.2: *Scatter plot of Execution Time v Sample Time for the SwissMunicipalities data set for GGA and SAA after 20 finetuning tests*

For the American Community Survey, 2015 data set finetuning tests, Figure 8.3, shows the GGA has a greater spread of sample sizes and execution times than the SAA. However, it does attain the two lowest sample sizes. This indicates that the finetuning process enabled the GGA to conduct a greater exploration of the search space. The GGA took significantly more time to find the lowest sample sizes. The

SAA has a tighter spread of sample sizes (indicating more exploitation of local search areas), however, it also has the lowest execution times. Sample sizes and execution times for the SAA appear to be clustered. This clusterisation could be as a result of the hyperparameter settings, used for the particular tests.

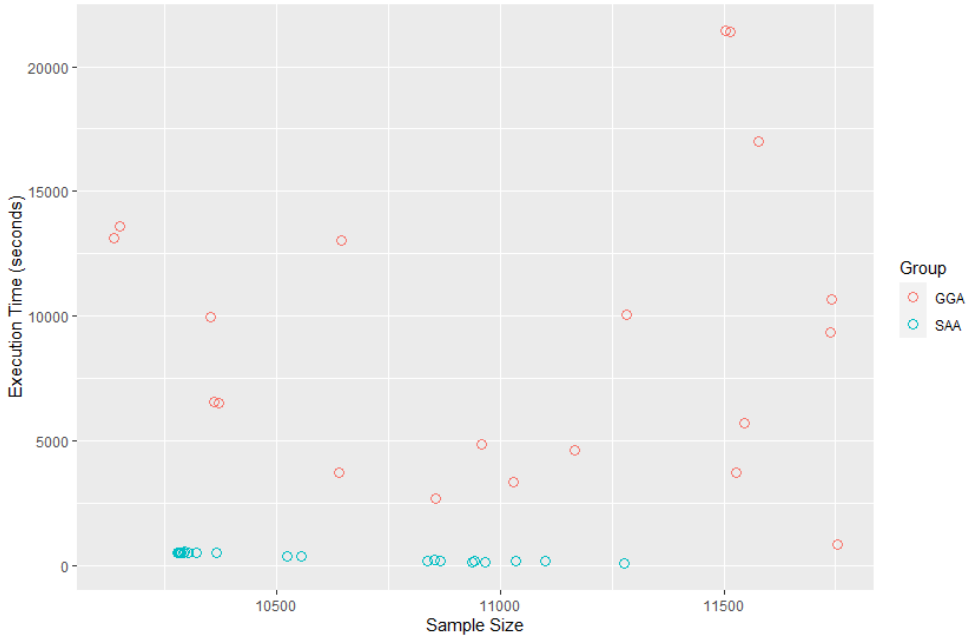


Figure 8.3: *Scatter plot of Execution Time v Sample Time for the American Community Survey, 2015 data set for GGA and SAA after 20 finetuning tests*

For the Kiva Loans data set finetuning tests, Figure 8.4, shows that the SAA has found a cluster of lower sample sizes. It also has found all sample sizes in significantly less time. As above, the GGA has a broader range of sample sizes, which also appear to be clustered.

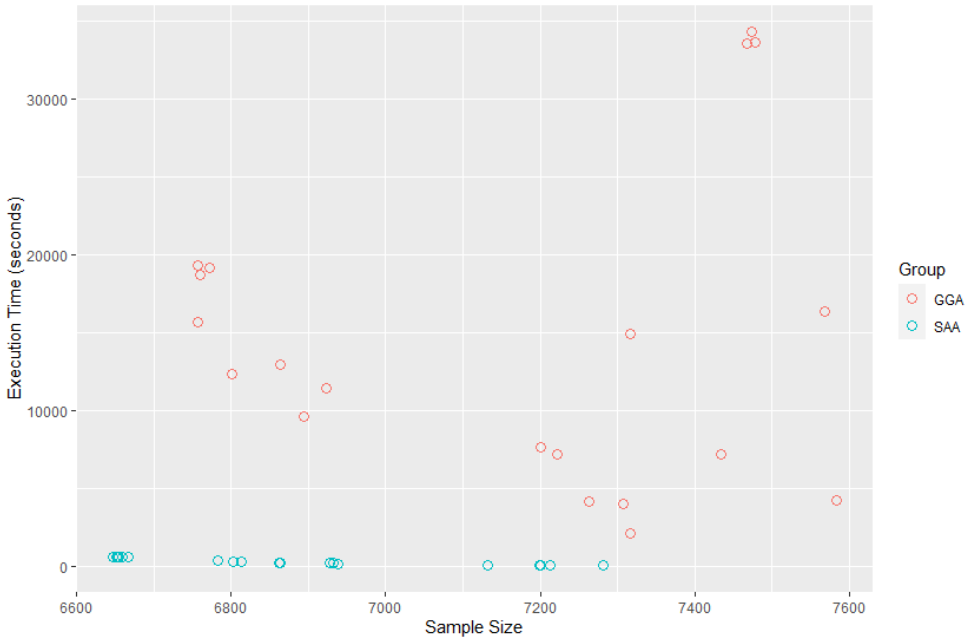


Figure 8.4: *Scatter plot of Execution Time v Sample Time for the Kiva Loans data set for GGA and SAA after 20 finetuning tests*

For the UN Commodity Trade Statistics data set finetuning tests, Figure 8.5, shows that the SAA typically finds a lower sample size, in less time, than the GGA. Again, there is clusterisation for both algorithm results, with a greater dispersion of results for the GGA.

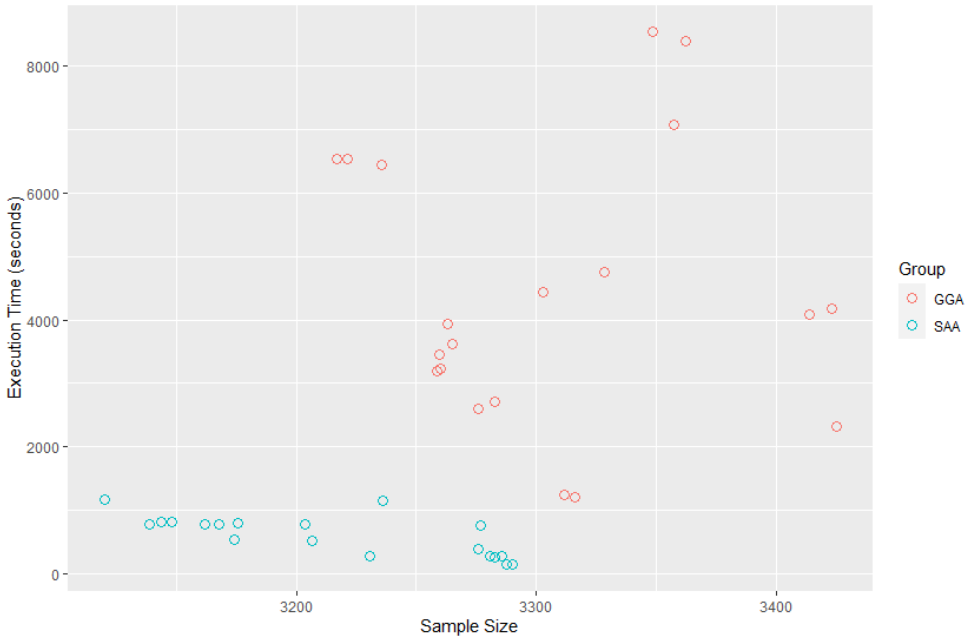


Figure 8.5: *Scatter plot of Execution Time v Sample Time for the UN Commodity Trade Statistics data set for GGA and SAA after 20 finetuning tests*

For the US Census, 2000 data set finetuning tests, Figure 8.6, shows that the SAA typically finds a lower sample size, in less time, than the GGA. Again, there is clusterisation, however the SAA results appear to have greater dispersion. In this case, the SAA hyperparametters appear to have promoted greater exploration of the search space, while still enabling exploitation of good search areas.

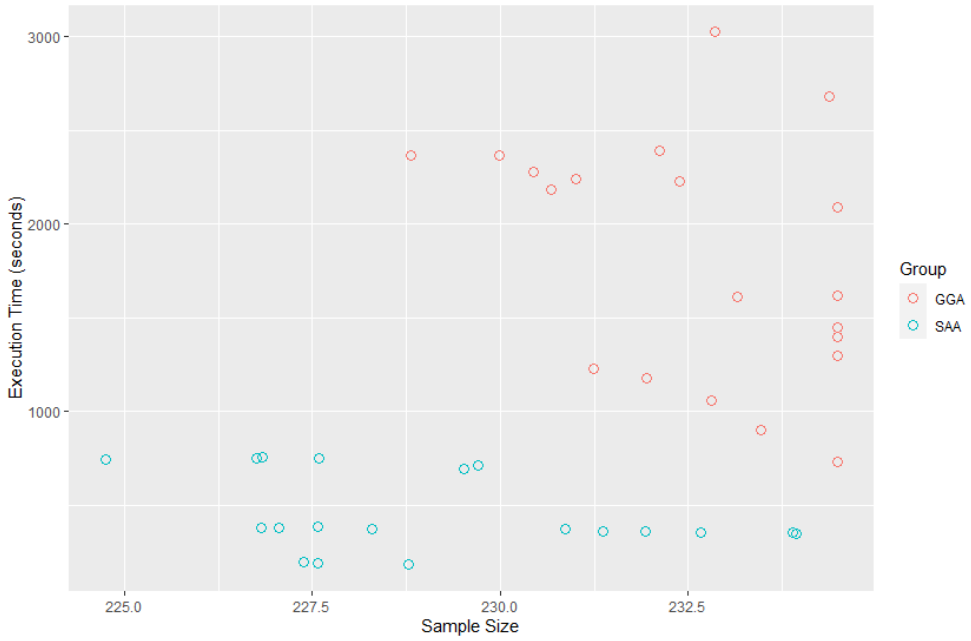


Figure 8.6: *Scatter plot of Execution Time v Sample Time for the US Census, 2000 data set for GGA and SAA after 20 finetuning tests*

8.6 Comparison with the continuous method in *SamplingStrata*

We also compared the SAA with the classical genetic algorithm which Ballin and Barcaroli (2020) have applied to partition continuous strata. We used the target variables outlined in table 8.5.1 above as both the continuous target and auxiliary variables (for clarity we outline them again in table 8.7.13 below) along with the precision constraints outlined in table A.1.1 (the appendix).

In practice, the target variable would not be exactly equal to the auxiliary variable though it is common for the auxiliary variable to be an imperfect version (for example an out-of-date or a related variable) available on the sampling frame. We invite the reader to consider this when reviewing the results of the comparisons below. It is also worth noting that initial solutions were created for both algorithms using the k-means method. Details on the training of hyperparameters for these

experiments also can be found in section 8.6.1, below.

8.6.1 Hyperparameters for the classical genetic algorithm and simulated annealing algorithm

Tables 8.6.11 and 8.6.12 outline the hyperparameters for the classical genetic algorithm and the simulated annealing algorithm. The add strata factor option is not available for the classical genetic algorithm and, therefore, is not included in table 8.6.11. More details on fine-tuning the hyperparameters are provided in section 8.6.2.

Table 8.6.11: *Hyperparameters for the classical genetic algorithm*

Data set	Iterations	Population size	Mutation chance	Elitism rate, E_R
Swiss Municipalities	4,000	50	0.0053360	0.4
American Community Survey, 2015	1,000	20	0.0009952	0.1
US Census, 2000	400	20	0.0002317	0.4
Kiva Loans	200	20	0.0817285	0.5
UN Commodity Trade Statistics data	5,000	30	0.0005599	0.2

Table 8.6.12: *Hyperparameters for the simulated annealing algorithm*

Data set	Number of sequences, maxit	Length of sequence, J	Temperature, T	Decrement constant , DC	% for maximum q value, $L_{max\%}$	Probability of new stratum, P(H+1)
Swiss Municipalities	5	5,000	0.02311057	0.9427609	0.3736443	0.0229361
American Community Survey, 2015	50	2,000	0.00000005	0.9528952	0.0001021	0.0000008
US Census, 2000	1	2,000	0.00002000	0.9665631	0.0221147	0.0160408
Kiva Loans	2	2,000	0.00053839	0.8660943	0.0014281	0.0216320
UN Commodity Trade Statistics data	2	250	0.00067481	0.9309940	0.0203113	0.0149499

8.6.2 Fine-tuning the hyperparameters for the classical genetic algorithm and simulated annealing algorithm

We fine-tuned the hyperparameters for the CGA and SAA using the same methodology described in section 8.5.6. Tables A.1.4 and A.1.5 outline the hyperparameters in each of the fine-tuning iterations.

As with the hyperparameters for the GGA and SAA comparison, the ranges for the hyperameters were set on a trial and error basis. The CGA is an exploratory algorithm and the SAA is an exploitative algorithm. Although both algorithms attain comparable solution qualities, this search of hyperparameters should not be taken as an exhaustive optimisation. This process found comparable near optimal solution qualities, while demonstrating the faster processing times of the SAA.

The first 10 sets were randomly generated using practical knowledge of the algorithms and data to define upper and lower bounds for each hyperparameter. In

the second 10 sets the *MBO* function attempts to optimise the hyperparameters using the previous solutions as a guide.

8.7 Target and auxiliary variables

Table 8.7.13 provides a summary of the target and auxiliary variables used in the experiments below.

Table 8.7.13: *Summary by data set of the target and auxiliary variable descriptions for the continuous method*

Data set	Target variables	Auxiliary variables	Description
SwissMunicipalities	Surfacebois	Surfacebois	wood area
	Airbat	Airbat	area with buildings
American Community Survey, 2015	HINCP	HINCP	Household income (past 12 months)
	VALP	VALP	Property value
	SMOCP	SMOCP	Selected monthly owner costs
	INSP	INSP	Fire/hazard/flood insurance (yearly amount)
US Census, 2000	HHINCOME	HHINCOME	total household income
Kiva Loans	term_in_months	term_in_months	duration for which the loan was disbursed
	lender_count	lender_count	the total number of lenders
	loan	loan	the amount in USD
UN Commodity Trade Statistics data	trade_usd	trade_usd	value of the trade in USD

8.8 Results

The attained sample sizes are compared in table 8.8.14 below where the sample size for the SAA is expressed as a ratio of the CGA. After the hyperparameters were fine-tuned (see section A.1.3) the resulting sample sizes are comparable.

Table 8.8.14: *Ratio comparison of the sample sizes for the simulated annealing algorithm and classical genetic algorithm on the continuous method*

Data set	CGA	SAA	Ratio
Swiss Municipalities	128.69	120.00	0.93
American Community Survey, 2015	4,197.68	3,915.48	0.93
US Census, 2000	192.71	179.89	0.93
Kiva Loans	3,062.33	3,017.79	0.99
UN Commodity Trade Statistics data	3,619.42	3,258.52	0.90

Table 8.8.15 compares the execution times for the set of hyperparameters that found the sample sizes for each algorithm in table 8.8.14 above, as well as the total execution times taken to train that set of hyperparameters.

Table 8.8.15: *Comparison of the execution times and total execution times for the simulated annealing and classical genetic algorithm algorithm on the continuous method*

	CGA		SAA		Ratio comparison	
Data set	**Execution time (seconds)**	**Total execution time (seconds)**	**Execution time (seconds)**	**Total execution time (seconds)**	**Execution time**	**Total execution time**
Swiss Municipalities	753.82	10,434.30	213.44	1,905.82	.28	.18
American Community Survey, 2015	22,016.95	227,635.51	13,351.19	169,115.92	.61	.74
US Census, 2000	3,361.90	46,801.78	51.94	1,147.36	.02	.02
Kiva Loans	3,232.78	48,746.61	300.16	4,149.06	.09	.09
UN Commodity Trade Statistics data	29,045.23	326,931.63	73.18	1,287.38	.003	.004

These results indicate a significantly lower execution time for the SAA for the attained solution quality. The computational efficiency gained by delta evaluation in the training of the recommended hyperparameters is also evident in the total execution times. For the *American Community Survey, 2015* experiment significantly more solutions were generated by the SAA than the CGA as a result of the given hyperparameters and this impacts the execution and total execution times (see also table 8.8.16). Table 8.8.16 compares the number of solutions generated by the classical genetic algorithm with the simulated annealing algorithm.

Table 8.8.16: *Comparison of the number of solutions generated by the classical genetic algorithm and simulated annealing algorithm on the continuous method*

	Number of solutions evaluated	
Data set	**CGA**	**SAA**
Swiss Municipalities	840,140	175,000
American Community Survey, 2015	918,102	5,100,000
US Census, 2000	43,272	18,000
Kiva Loans	146,730	292,000
UN Commodity Trade Statistics data	20,521,026	85,500

In all cases except for *Kiva Loans* and the *American Community Survey, 2015* the SAA has generated fewer solutions. The low number of solutions generated by both algorithms for the *US Census, 2000* experiment may indicate that the initial k-means solution was near the global minimum. The *American Community Survey, 2015* results indicate that the SAA generated significantly more solutions to get to a comparable sample size with the CGA. As we are moving, predominantly, $q = 1$ atomic strata between strata such changes in this case had limited impact on solution quality from one solution to the next. However, the gains achieved by delta evaluation meant that more solutions were evaluated per second leading to a more complete search and a lower sample size being attained.

For these experiments, the CGA took longer to find a comparable sample size in all cases.

As with the atomic strata comparison, we visualise the results for both algorithms across each of the 20 fine tuning tests.

For the SwissMunicipalities data set finetuning tests, Figure 8.7, shows that the SAA finds a lower sample size more often than the CGA. It also does so, generally, in less time. There is less dispersion of the SAA sample size and processing times.

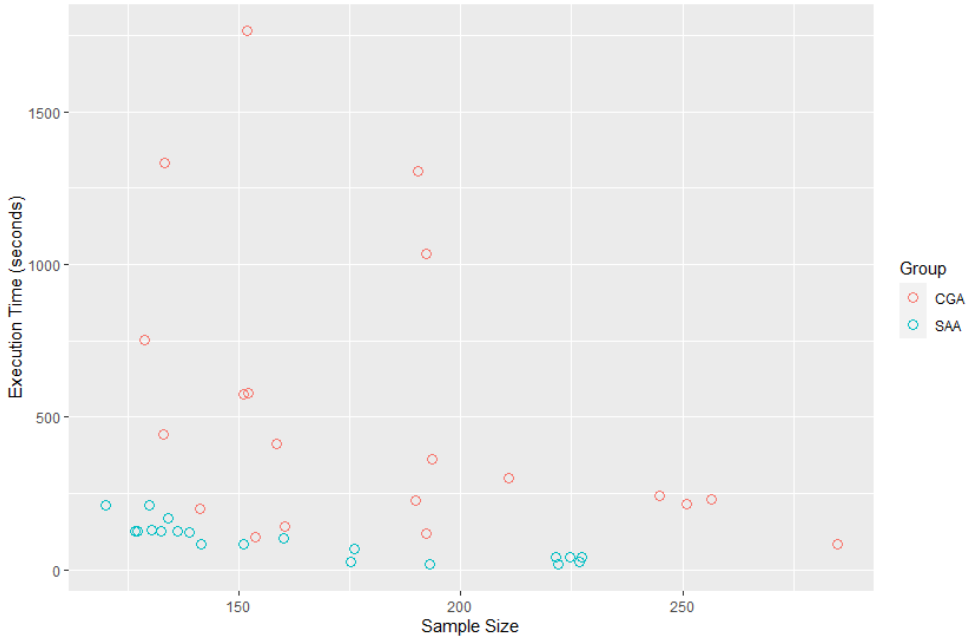


Figure 8.7: *Scatter plot of Execution Time v Sample Time for the SwissMunicipalities data set for CGA and SAA after 20 finetuning tests*

For the American Community Survey, 2015 data set finetuning tests, Figure 8.8, shows the SAA has a greater spread of sample sizes than the CGA . However, the SAA also attains a cluster of lower sample sizes. The graph also indicates a negative relationship between sample size and processing time for the SAA. In other words, as the SAA spent more time searching, the solution quality improved. There is no such relationship evident for the GGA, and there is a wide dispersion of sample sizes and execution times.

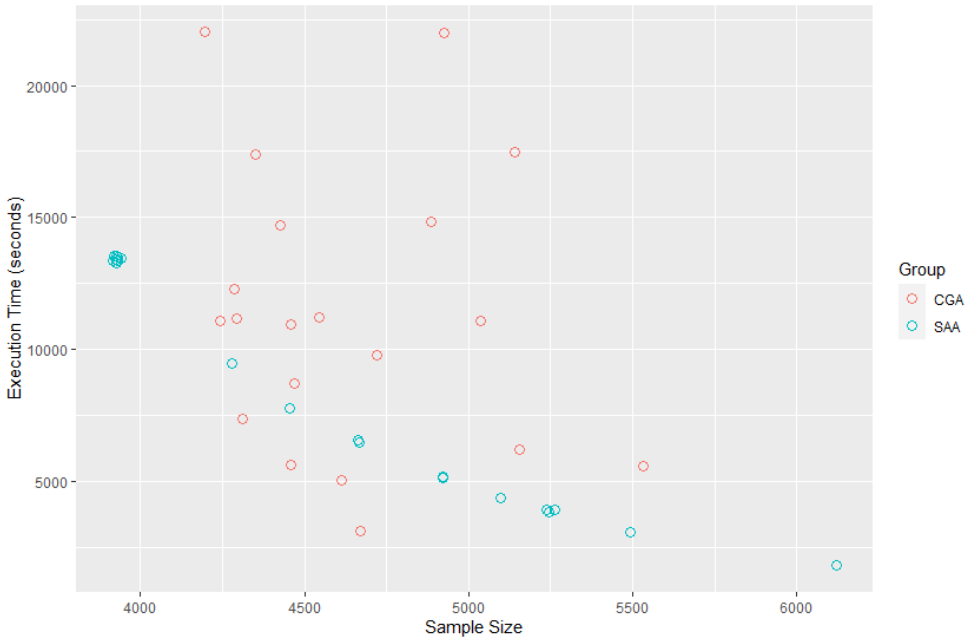


Figure 8.8: *Scatter plot of Execution Time v Sample Time for the American Community Survey, 2015 data set for CGA and SAA after 20 finetuning tests*

For the Kiva Loans data set finetuning tests, Figure 8.9, shows that, as with the atomic strata tests, the SAA has found a cluster of lower sample sizes. Likewise, it has found all sample sizes in less time. The clear clusterisation of results for the SAA, appears to be due to a combination of solutions generated (search time) and the remaining hyperparameters. The CGA has a greater dispersion of sample sizes and execution times.

Figure 8.9: *Scatter plot of Execution Time v Sample Time for the Kiva Loans data set for CGA and SAA after 20 finetuning tests*

For the UN Commodity Trade Statistics data set finetuning tests, Figure 8.10, shows that the SAA always finds a lower sample size, in less time, than the CGA. All results for the SAA are tightly clsuters, indicating that we may be close to the optimal solution. As before, there is a greater dispersion of results for the CGA.

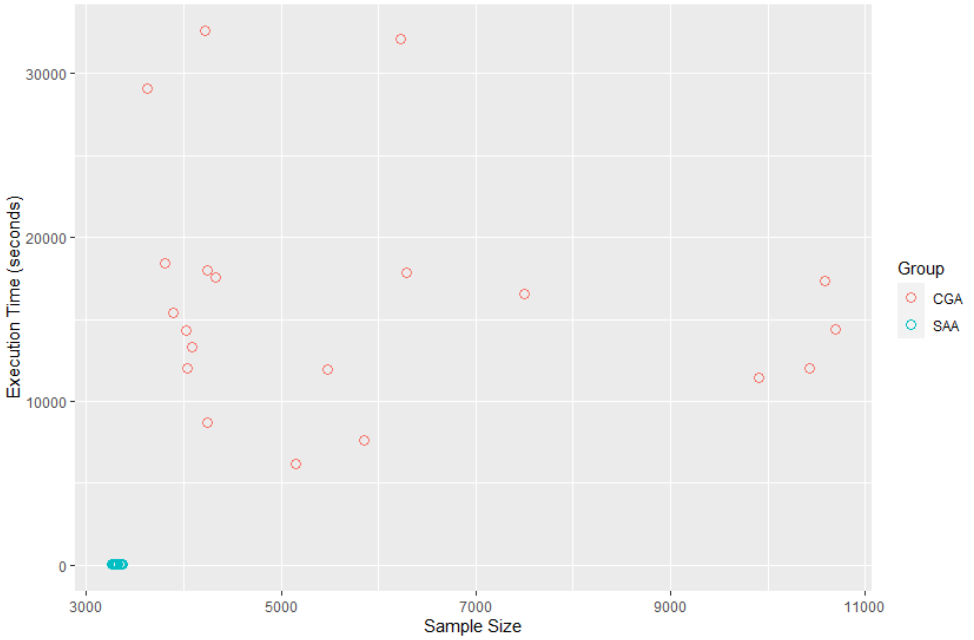


Figure 8.10: *Scatter plot of Execution Time v Sample Time for the UN Commodity Trade Statistics data set for CGA and SAA after 20 finetuning tests*

For the US Census, 2000 data set finetuning tests, Figure 8.11, shows that the SAA always finds a lower sample size in less time than the CGA. Again, the SAA sample sizes and execution times are tightly clustered. There appears to be a weak, and negative relationship between sample size and execution times of the CGA. This indicates that, in order to achieve further gains in sample sizes, the CGA will need to evaluate more solutions, i.e. needs more time.

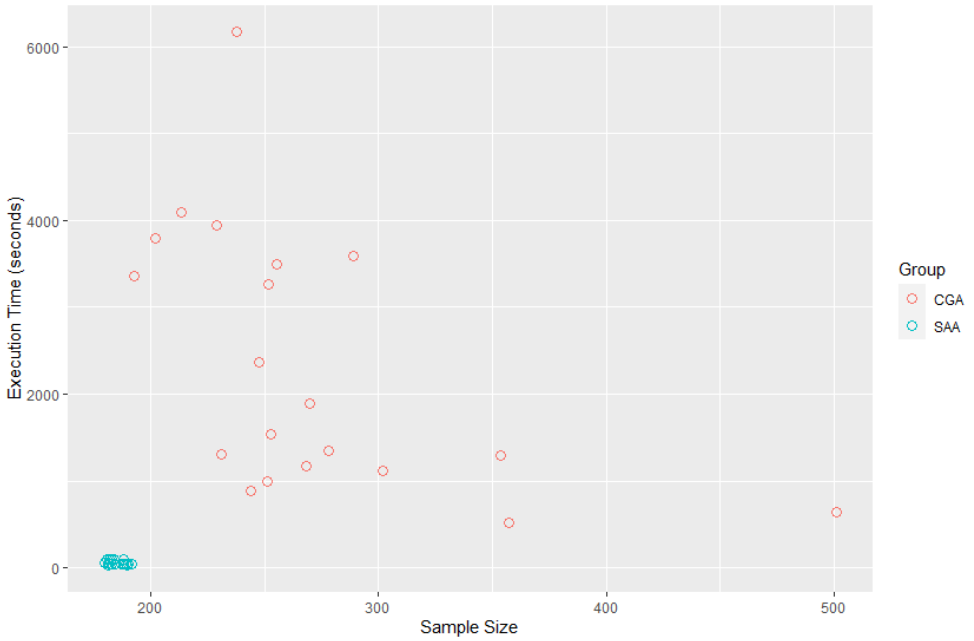


Figure 8.11: *Scatter plot of Execution Time v Sample Time for the US Census, 2000 data set for GGA and SAA after 20 finetuning tests*

As pointed out in chapter 7, classical genetic algorithms are not as efficient for grouping problems as the grouping genetic algorithm because solutions tend to have a great deal of redundancy. We would, therefore, propose that the GGA be applied also to continuous strata. On the basis of the above analysis, and the performance of SAAs in local search generally speaking along with the added gains in efficiency from delta evaluation, we would also propose that the SAA be considered as an alternative to the classical genetic algorithm.

8.9 Processing platform

Table 8.9.17 below provides details of the processing platform used for these experiments.

Table 8.9.17: Specifications of the processing platform

Specification	Details	Notes
Processor	AMD Ryzen 9 3950X 16-Core Processor, 3493 Mhz	
Cores	16 Core(s)	
Logical processors	32 Logical Processor(s)	32 cores in R
System model	X570 GAMING X	
System type	x64-based PC	
Installed physical memory (RAM)	16.0 GB	
Total virtual memory	35.7 GB	
OS name	Microsoft Windows 10 Pro	

In all cases, R version 4.0 or greater was used. We used the *foreach* (Microsoft and Weston, 2020) and *doParallel* (Corporation and Weston, 2020) packages to run the experiments in parallel. The number of cores used in the experiments was 31 (32 less 1) and this means that in the three experiments with more than 31 domains (*American Community Survey 2015, Kiva Loans, UN Commodity Trade Statistics data*) the *foreach* algorithm continued to loop through the available cores until a solution had been found for all domains.

8.10 Conclusions

We compared the SAA with the GGA in the case of atomic strata and the classical genetic algorithm in the case of continuous strata (Ballin and Barcaroli, 2020). The k-means algorithm provided good starting points in all cases. When the hyperparameters have been fine-tuned all algorithms attain results of similar quality - however, the execution times for the recommended hyperparameters are lower for the SAA than for the GGA in respect of atomic strata and classical genetic algorithm in respect of continuous strata. Delta evaluation also has advantages in reducing the training times needed to find the suitable hyperparameters for the SAA.

The GGA might benefit from being extended into a memetic algorithm (see section 4.2.3) by using local search to quickly improve a chromosome before adding it to the GGA chromosome population.

The SAA, by using local search (along with a probabilistic acceptance of inferior solutions), is well suited to navigation out of local minima and the implementation of delta evaluation enables a more complete search of the local neighbourhood than would otherwise be possible in the same computation time.

Chapter 9

Combining clustering algorithms with hill-climbing

9.1 Introduction

In chapter 7, we reported that an optimal solution (there may be more than one) may be found by computing the sample allocation for each possible stratification. In other words, this optimal solution is unknown or latent. However, this exhaustive evaluation of each candidate solution, or grid-search, quickly becomes computationally prohibitive for larger sets of possible solutions. This is further compounded by the computation time necessary to evaluate each solution. For this reason a number of heuristic algorithms have been proposed to search for solutions that are of an acceptable quality, e.g. a local minimum that is in close proximity to the global minimum, in a tolerable computing time.

In survey methodology, when clustering is referenced in relation to sampling, it usually refers to a two-step process of firstly, dividing the population into clusters, and secondly, randomly selecting a small number of these, in order to take measurements for every unit within.

In this chapter, our focus will be on the first step - as it is applied in the unsupervised machine learning aspect of clustering – where similar units of a population are grouped together into clusters. Indeed, the search for the optimal solution resembles clustering, in the sense that, we seek to group basic strata with similar characteristics together into common groups. Accordingly, for our purposes, clusters are equivalent to strata, and the similarity within each cluster will enable more precise sample allocations.

We explore staged combinations of K-means, expectation maximisation, self-organising map, fuzzy clustering and neural gas algorithms with hill-climbing to find better quality solutions to this problem, in reduced training times, when compared to our earlier experiments, on the same data, with the Grouping Genetic Algorithm and Simulated Annealing Algorithm in chapter 8.

Section 9.2 provides background details on the K-means algorithm. Section 9.7 summarises the Hartigan-Wong version of the K-means algorithm. Section 9.8 discusses expectation maximisation. Section 9.9 talks about the self-organising map. Section 9.10 describes the fuzzy clustering algorithm. Section 9.11 outlines the neural gas algorithm. The hill-climbing algorithm with delta evaluation is depicted in section 9.12. Section 9.13 provides the details of the computer specifications. This is followed by empirical comparisons for atomic and continuous strata in sections 9.14 and 9.15. The conclusions and suggestions for future work in section 9.16 closes the chapter.

9.2 Background information

The equivalency of the stratification aspect of the joint stratification and sample allocation problem to a machine learning clustering technique, the K-means method, was first noted by Lisic et al. (2018). This refers to a family of algorithms "developed as a result of independent investigations in the 1950s" (Pérez-Ortega et al., 2019), but first given the name K-means by MacQueen et al. (1967).

The K-means algorithm separates observations in to k-clusters (where k is an input parameter), so that the within sum of squares in each cluster is minimised (Hartigan and Wong, 1979). The purpose of the algorithm is not to find some unique, definitive grouping, but rather to simply aid the investigator in obtaining qualitative and quantitative understanding of large amounts of N-dimensional data by providing him with reasonably good similarity groups (MacQueen et al., 1967).

K-means is a greedy algorithm (Jain, 2010) that finds a local minimum for a problem, that is known to be NP hard (Drineas et al., 2004), and this has also led to various heuristic approaches being used to solved the problem. K-means algorithms, e.g. Forgy (1965); Hartigan and Wong (1979); Lloyd (1982); MacQueen et al. (1967), remain commonly used clustering algorithms. This is mainly due to their ease of implementation, computational efficiency and low memory consumption (Morissette and Chartier, 2013).

This is, notwithstanding, many other similar clustering techniques which

have been introduced since their inception. These techniques include expectation-maximization (EM) (Dempster et al., 1977), fuzzy K-means clustering (Bezdek et al., 1984), self-organising maps (SOM) (Kohonen, 1990) and neural gas (Martinetz et al., 1991)

K-means algorithms are local search heuristics, with a number of number of chosen inputs, such as k and the initial centroids. They are sensitive to the initial centroids chosen and, while this is what makes them more likely of finding local minima (more often than global minima), it also means that they sometimes produce counter-intuitive results (Dirgová Luptáková et al., 2016).

Secondly, K-means algorithms provide results that are robust in nature with the implication that the local minimum is harder to escape from and the structure of the solution (i.e. number of clusters and constituents of each cluster) may be far from that of the optimal solution. Thirdly, the algorithm is also known to be sensitive to the order at which constituents (basic strata) would be re-assigned to clusters (atomic strata) (Morissette and Chartier, 2013). Nonetheless, these three issues are commonly mitigated by re-starting the algorithm a number of times along with varying the number of k, and selecting the best quality solution.

Finally, K-means also has a tendency to create clusters of equal size (in area) even if this does not represent the best clustering of the data (Äyrämö and Kärkkäinen, 2006). This can be offset by clustering alternatives to the K-means algorithm, such as those discussed below in this chapter.

Following Lisic et al. (2018)'s proposal, Ballin and Barcaroli (2020) recommended using an efficient version of the K-means clustering algorithm (Hartigan and Wong, 1979), to determine initial solutions for our problem which are then improved upon or "optimised" using a grouping genetic algorithm. In chapter 8, we used the same approach to determine initial solutions which are optimised using a simulated annealing algorithm. Hartigan and Wong (1979)'s algorithm searches for a solution that is locally optimal and, although it is quick, alternative solutions to that found by the algorithm may have smaller within sums of squares.

The proximity of the local optimum to the global optimum, as well as the degree of similarity within each cluster, will have implications on the search for the former by the genetic and simulated annealing algorithms. The further the local optimum is from the global optimum, the longer it is likely to take. Furthermore, the grouping genetic algorithm and simulated annealing algorithms require that their hyperparameters be adequately suited to the problem in order to operate efficiently. Training the hyperparameters can also be computationally expensive.

Hartigan and Wong (1979)'s algorithm works well when the natural data groupings are spherical in nature and clusters are linearly separable, with a corresponding high degree of similarity within clusters and low degree of similarity between clusters. Indeed, it has been shown that K-means can converge to a global minimum when clusters are well-separated (Meilă, 2006).

However, it is not likely to work as well when there is not a clear separation between clusters and also when there an occurrence of basic strata from exogenous clusters within cluster spheres, i.e. they may be close to a particular cluster centroid, but are more suited to another cluster, e.g. see Figure 9.3). This can occur when the assignment of the information for basic strata from within the radius of a given cluster centroid to exogenous clusters improves the objective function. Recall that the objective is a function of not only the basic strata means, but their standard deviations and number of records, as well. On the other hand, cluster centroids are calculated solely from the means of the target variable means for each basic stratum.

9.3 Choosing the parameter K

The initial solution proposed by Lisic et al. (2018) is created using a K-means clustering algorithm in which atomic strata are assigned to a pre-determined number, $k = 1, ..., K$, of strata (i.e. in this case clusters) based on their similarity to other atomic strata in each stratum.

We refer to Kodinariya and Makwana (2013); Pham et al. (2005) for the following discussion. Determining the appropriate number of clusters to use is a trial and error process, made difficult by the lack of an evaluation method for determining what is appropriate. We now provide a number of methods for selecting K.

1. Rule of thumb. Set K to be $\sqrt{\frac{L}{2}}$, where L is the number of input rows (basic strata).
2. Values of K specified within a range or set. Instead of using a single predefined K, a set of values can be considered. The range of values for K should be large enough to allow well separated groupings of similar items, but the upper limit on K should be significantly lower than the number of rows in the data set (the main reason for clustering).
3. Values of K specified by the user. Over a number of iterations the clustering algorithm is evaluated using different values of K. The validity of the clustering result is assessed visually without evaluating the quality of the clustering. Such visual inspections cannot be applied in the case of multidimensional data.

4. Values of K determined in a later processing step. This implies the determination of K by the main algorithm that utilises the clustering created by the initial clustering algorithm. This is a black box scenario, as the optimal value for K cannot be determined in advance and depends on the characteristics of the main algorithm and the input data.

5. Values of K equated to the number of generators. This is useful for synthetic data sets, created using a set of normal or uniform distribution generators. Then, the number of clusters can be equated to the number of generators. The assumption being that clusters will each cover all objects created by a particular generator. However, objects created by generators could be in overlapping cluster areas. Also, using the number of generators as the number of clusters, might group dissimilar objects together. This method cannot be applied to practical problems, as the data distributions and the number of generators are unknown.

6. Values of K determined by statistical measures. There are a number statistical measures available for determining K. These comprise the Bayesian information criterion or Akeike's information criterion (which assume Gaussian distributions), the Gap Statistic, an information theoretic approach, the silhouette width or Monte Carlo techniques (e.g. cross validation). Expectation Maximisation use probablistic measures (soft clustering) to determine the number of clusters, however these measures do not apply to partitioning (hard clustering) algorithms such as K-means. Assumptions regarding the distributions of underlying data, may not hold on real data.

7. Values of K equated to the number of classes. In this method the number of clusters is simply equated to the number of classes in the data sets. This implies the creation of clusters, each containing items belonging to one class (assuming their values are similar). However, this may not be a practical assumption in real world problems.

8. Values of K determined through visualization. A visual inspection of data is used to provide expected clustering results. In particular, this is useful in order to assess whether a clustering model would be useful for the particular position, shape, size, and object distributions of the input data. However, the approach is only useful for up to three dimensions of data.

9. Values of K determined using a neighbourhood measure. This approach entails adding a neighbourhood measure to the clustering algorithm's cost function, for the purposes of determining K. As this implies a modification to the

existing cost function, it also entails a modification to the algorithm.

There is significant scope for utilising an algorithm or method for determining K mainly in a starting solution, as they tend to be found quickly and near optimal. The structure of such starting solutions will provide an indication for the number of strata in the optimal solution. The main algorithm may add or reduce the number of strata in the search for a better quality solution.

In this manner, there is also scope for refining the K-means solution algorithm process even further by using heuristics such as the Elbow method or Gap statistic method to select the starting solution, and limit the maximum number of strata, K, using a default parameter such as equating K to the maximum number of classifications in the auxiliary variable columns, or the square root of the number of atomic strata in each domain (e.g. region) or increasing k incrementally until the algorithm plateaus.

9.4 Determining K in the experiments below

Referring to item 3 in section 9.3 above, Ballin and Barcaroli (2020) utilise the *KmeansSolution* algorithm to generate clusterings for iteratively increasing values of k, and evaluate the solution quality for each clustering. Although the main goal is to select a good quality solution: that solution will have k clusters. The algorithm starts by partitioning the basic strata into the k clusters (strata), where k starts at a value, e.g. 2, based on their proximity to a stratum centroid (the mean of each of the relevant values from the target variable columns in that stratum). This solution is then evaluated using the Bethel-Chromy algorithm. The K-means algorithm then proceeds to create a solution with $k+1$ strata and again evaluates that solution. This process is repeated for each k up to a predefined maximum number of strata. The solution which provides the lowest cost is then chosen to be the initial solution. We use the *KmeansSolution* algorithm to determine k in the k-means experiments described below.

We also use the above approach in analagous algorithms (which we developed, based on the the *KmeansSolution* algorithm) to generate and evaluate clusterings for iteratively increasing values of k, and selecting the lowest cost solution found, in the expectation maximisation, the self-organising map, fuzzy clustering and neural gas experiments (in the single stage and multi-stage combinations) described below.

9.5 Use of *R* packages

The clustering algorithms below mainly have been utilised as they are found in the various cited R Core Team (2021) packages, with the exception of expectation maximisation, where the *estep* and *mstep* functions from the *mclust* package (Scrucca et al., 2016) are used sequentially in two steps iteratively until they have converged on a solution. In cases where we trained hyperparameters for an algorithm, we used the R package *mlrMBO* (Bischla et al., 2017) to do so (see the methodology described in section 6.1).

9.6 Inputs and outputs

In each of the clustering algorithms below, the input data are the basic (atomic or continuous) strata, the codebook vectors (where we cluster the SOM output) or the neural gas clustering (as a starting point where we cluster the neural gas output). We use sample to denote the input data, and sample unit to denote each row. The outputs are a matrix of probabilities, codebook vectors (output for the SOM), and/or the clustering solutions which are used as a starting solution for the hill climbing algorithm.

9.7 The Haritgan-Wong version of the K-means algorithm

As described above, Ballin and Barcaroli (2020) and chapter 8 use the Hartigan and Wong (1979) version of the K-means algorithm to generate starting solutions for the genetic and simulated annealing algorithms respectively. This version seeks to attain the locally optimal minimum sums of squares within each cluster by moving data-points from one cluster to another. This means that, although a sample unit might be closer to its current cluster centroid, it could still be re-assigned to another cluster, if in doing so the sums of squares was reduced. We have referred to the pseudocode provided by Vouros et al. (2021) to guide the summary, below.

Algorithm 7 Haritgan Wong K-means clustering algorithm

1: Initialise K initial centroids $m_{1,g} \dots m_{K,g}$ using an initialisation method.
2: Assign each $\bar{Y}_l$ to a cluster $c_{k'}$, where $k^{'} = \arg\min_k \left\{ \sum_{g=1}^{G} (\bar{Y}_{l,g} - m_{k,g})^2 \right\}$.
3: For each sample unit $\bar{Y}_l$ remove it from its cluster $c_{k'}$ and compute the centroid of $c_{k'}$ using the remaining basic strata in that cluster, $m_{k',g} = \frac{1}{n_k^{'}} \sum_{l=1}^{n_{k'}} \bar{Y}_{l,g}$ where $\bar{Y}_l$ refers to the basic strata $\in c_{k'}$.
4: Assign $\bar{Y}_l$ to cluster c_{k^*}, where $k^* = \arg\min_k \left\{ \sum_{g=1}^{G} (\bar{Y}_{l,g} - m_{k,g})^2 \right\}$.
5: Recompute the centroid of cluster c_{k^*} as follows $m_{k^*,g} = \frac{1}{n_{k^*}} \sum_{l=1}^{n_{k^*}} \bar{Y}_{l,g}$ where $\bar{Y}_l$ refers to the basic strata $\in c_{k^*}$.
6: Repeat step 3 to step 5 until convergence.

where $k = 1, 2 \dots K$ is the number of clusters (strata), n_k the number of basic strata of the k^{th} cluster and G the number of target variables; $\bar{Y}_{l,g}$ is the value of the g-th mean in sample unit l, $\bar{Y}_l$ is the vector representing the sample unit l; m_k is the vector specifying the location of the k^{th} cluster centroid.

The initialisation method used by the K-means algorithm is the random selection of data points (basic strata) as initial centroids proposed by MacQueen et al. (1967). Other initialisation methods comprise the kmeans++ method proposed by Arthur and Vassilvitskii (2006) which uses a uses a probabilistic approach in order to select as initial centroids data points that are far away from each other, and the maximin approach of Gonzalez (1985) which uses distance to select centroids which are far apart from each other.

The inputs are the sample (basic strata, codebook vectors or neural gas starting solution), and K, which is determined using the *KmeansSolution* algorithm. The algorithm is deemed to have converged when no sample unit changes cluster in the next iteration. We did not train any hyperparameters for the K-means algorithm, however we set the maximum number of iterations as 1,000 and the number of restarts as 100.

9.8 Expectation maximisation

A Gaussian Mixtures Model considers whether each sample unit belongs to one of a mixture of K Gaussian mixture components (each representing a cluster or given stratum). The Expectation-Maximisation, EM, algorithm is an iterative method that we can use to maximise the likelihood a sample unit belongs to each cluster. It is

used in clustering problems where the optimal solution is latent. EM is similar to K-means in the action of grouping similar basic strata into strata.

It differs from K-means in the sense that each sample unit has a probability of being in each cluster (i.e. soft clustering), whereas in K-means a sample unit is assigned to one cluster (hard clustering). EM is not as sensitive to the initial clustering as K-means and works well in practice. However, it is not guaranteed to find the maximum likelihood. .

The following description of how the EM algorithm works is based on that provided by Chadha (2020). As with the K-means algorithm , we start with our basic strata which are characterised by $x_1, \ldots, x_n$ and each x_i represents a basic stratum. We want to group these into K clusters, where $k = 1, 2 \ldots K$, and this clustering is denoted by z, where each $z_i \sim$ Multinomial (ϕ)(where $\phi_k \geq 0, \sum_{k=1}^{K} \phi_k = 1$, and the parameter ϕ_k gives $p(z_i = k)$, and $x_i | z_i = k \sim N(\mu_k, \sigma_k)$) is a latent random variable that can take one of K values. Thus, we posit that each x_i was drawn from one of K Gaussians depending on z_i. We model the complete data with a joint distribution $p(x_i, z_i) = p(x_i|z_i)p(z_i)$. This is called the mixture of Gaussians model.

The parameters of our model are ϕ, μ and σ. To estimate them, we calculate likelihood of our data as follows:

$$\ell(\phi, \mu, \sigma) = \sum_{i=1}^{n} log\ p(x_i; \phi, \mu, \sigma)$$

or

$$\ell(\phi, \mu, \sigma) = \sum_{i=1}^{n} log \sum_{z_i=1}^{K} p(x_i \mid z_i; \mu, \sigma) p(z_i; \phi)$$

Each z_i indicates which of the K Gaussians each x_i comes from. If we knew the values of each z_i then we could calculate the likelihood using

$$\ell(\phi, \mu, \sigma) = \sum_{i=1}^{n} log \sum_{z_i=1}^{K} p(x_i \mid z_i; \mu, \sigma) + p(z_i; \phi)$$

Maximizing this with respect to ϕ, μ and σ gives the parameters

$$\phi_k = \frac{1}{n} \sum_{i=1}^{m} 1\{z_i = j\}$$

$$\mu_k = \frac{\sum_{i=1}^{m} 1\{z_i = k\} x_i}{\sum_{i=1}^{m} 1\{z_i = k\}}$$

$$\sigma_k = \frac{\sum_{i=1}^{m} 1\{z_i = k\}(x_i - \mu_k)(x_i - \mu_k)'}{\sum_{i=1}^{m} 1\{z_i = k\}}$$

where $'$ denotes transpose.

The EM algorithm has two main steps, in the E-step, it tries to "guess" the values of the z_i's. In the M-step, it updates the parameters of the model based on these guesses, assuming they are correct, through use of maximisation. The EM algorithm starts with an initial guess for ϕ, μ and σ, and then iterates the E-step and M-step as described below until convergence. The algorithm operates as follows:

Repeat until convergence: $\{$

(E-step) For each i, k, set

$$w_{i,k} := p(z_i = k | x_i; \phi, \mu, \sigma)$$

(M-step) Update the parameters:

$$\phi_k = \frac{1}{n} \sum_{i=1}^{m} 1\{w_i = k\}$$

$$\mu_k = \frac{\sum_{i=1}^{m} 1\{w_i = k\} x_i}{\sum_{i=1}^{m} 1\{w_i = k\}}$$

$$\sigma_k = \frac{\sum_{i=1}^{m} 1\{w_i = k\}(x_i - \mu_k)(x_i - \mu_k)'}{\sum_{i=1}^{m} 1\{w_i = k\}}$$

$\}$

In the E-step, the posterior probability of each z_i given the x_i is calculated using Bayes rule:

$$p(z_i = k | x_i; \phi, \mu, \sigma) = \frac{p(x_i | z_i = k; \mu, \sigma) p(z_i = k; \phi)}{\sum_{k=1}^{K} p(x_i | z_i = k; \mu, \sigma) p(z_i = k; \phi)}$$

Note that $p(x_i | z_i = k; \mu, \sigma)$ is given by evaluating the Gaussian density with mean μ_k and covariance σ_k at x_i; where $p(z_i = k; \phi)$ is given by ϕ_k. The values $w_{i,j}$ calculated in the E-step and M-step represent our guesses for the values of $z_i = k$ as $w_{i,j}$ is the expected value of $z_i = k$.

To generalise the description of the algorithm, we use θ to represent the parameters of our model (ϕ, μ and σ) and for each i, let Q_i be the posterior distribution of the z_i's ($\sum_z Q_i(z) = 1, Q_i(z) \geq 0$) given x and the setting of the parameters θ and it can be shown that the optimal choice of Q is $p(z|x;\theta)$ for

computing the posterior distribution of z. The EM algorithm can then summarised as follows:

Algorithm 8 Expectation-Maximisation clustering algorithm

1: Let $t = 0$. Initialize $\hat{\theta}^t$. We randomly generate z to estimate θ^t.
2: For $t = 1, 2, \ldots,$ repeat:
3: (E-step)

$$Q_{i,j} = p(z_i = k | x_i; \hat{\theta}^{t-1}) \tag{9.1}$$

4: Set, (M-step)

$$\hat{\theta}^t = \arg\max_{\theta} \sum_i \sum_{z_i} Q_i(z_i) \log \frac{p(x_i, z_i; \theta)}{Q_i(z_i)} \tag{9.2}$$

5: Let $t = t + 1$
6: Repeat until convergence $(\hat{\theta}^t - \hat{\theta}^{t-1} < \epsilon)$, where ϵ denotes some threshold..

We used the *estep* and *mstep* functions from the *mclust* package (Scrucca et al., 2016) in R (R Core Team, 2021) iteratively until they converged on a solution. Convergence is measured by the difference in the proportional distributions of basic strata allocated to the K clusters between each iteration of steps 2 and 3 being less than 1×10^{-15}. As a starting point, we randomly generated a clustering. However, it is also possible to generate a clustering using a method, such as, but not limited to, any of those discussed here.

We did not train any hyperparameters for EM. The inputs are the basic strata and the intitial clustering. The output is a matrix of probabilities, where each row provides the probabilities of a sample unit belonging to each cluster. The highest probability in each row determines the final (hard) clustering.

9.9 The self-organising map (SOM)

In a process that again is similar to K-means, we can use self-organising maps (SOMs), to group similar data together into units (or clusters) on a grid (or map). Indeed SOMs are also known as constrained K-means because in SOMs the centroids are fixed, whereas in K-means they vary according the changing data in each cluster. In SOMs each cluster is arranged in a rectangular or hexagonal grid. The clusters, however, are different to the clusters/strata discussed up to now.

A SOM is an artificial neural network algorithm that takes the data as input and transcribes them into an output of lower dimensionality (usually bi-dimensional) (Morissette and Chartier, 2013). They are similar to dimension reduction techniques

such as principal component analysis and multidimensional scaling. In this respect, K-means is harder to visualise if the clusters are multi-dimensional.

The SOM typically consists of two layers of units, the input layer (basic strata) and the output layer (clusters- which have weight vectors denoting cluster centroids). Clusters in the output layer are arranged in form of a topological architecture which is usually two-dimensional and a grid consisting of rows and columns. The number of clusters in the output layer represents the maximum number of clusters (i.e. not need contain basic strata). This maximum number of clusters determines whether the map provides a suitable similarity clustering of the data. The topological structure can also be one-dimensional, where the clusters are visualised as linear row or column. However, it is generally agreed that a two-dimensional map provides a better representation of the clusters (Asan and Ercan, 2012). We use a two-dimensional hexagonal grid in the SOM experiments below.

SOMs use a Hebbian competitive learning algorithm to build the map. Each basic stratum is represented by an m-dimensional input vector, $x = (x_1, x_2 \dots x_m)'$. The algorithm begins with initialising the weight vectors $w_i = (w_{i,1}, w_{i,2} \dots w_{i,m})$ where $i = 1, 2, \dots, n$ and denotes the number of clusters. The weights connect the basic strata to the clusters and, and are updated through the learning process. The final results are subject to the choice of initial weights, $w_{i(t)}$ for $t = 0$, where t represents the iteration number, and can be determined either arbitrarily or systematically (e.g. linearly). The most common initialisation method assigns random values to the weight vectors. However, it is also possible to randomly select data points. This is the approach we use.

It then proceeds to find the nearest (winning) cluster to a randomly selected sample unit. To find the best matching cluster, we compute the distances between a basic stratum (x) and all the weight vectors (w_i) of the SOM using the sum of squares distance. The distance between x, chosen randomly from the basic strata, and all the weight vectors at iteration t is calculated using the following formula:

$$d_{i(t)} = \sum_{j=1}^{m} (x_{tj} - w_{tj,i})^2 \quad i = 1, 2, \dots, n$$

At the end of this process, the best-matching (winning) cluster c at iteration t is determined by using the minimum distance criterion:

$$c(t) = \arg\min_{i} \sum_{j=1}^{m} (x_{tj} - w_{tj,i})^2$$

The weights of winning clusters, and those of their neighbourhood, are updated. The weight vectors of the winner and its neighboring units in the output space are adjusted to become more representative of the basic stratum. This updating requires the consideration of the following two parameters: learning rate and neighborhood size. The learning rate, $\alpha(t)$, controls the rate of change of the weight vectors and ranges between 0 and 1. The learning rate gradually decreases as a function of the iteration step index t. Given the weight vector $w_i(t)$ of the winning cluster i at iteration t, the updated weight vector $w_i(t+1)$ at iteration $t+1$ is defined as:

$$w_i(t+1) = w_i(t) + \alpha(t)[x(t) - w_i(t)]$$

The clusters in the neighborhood of the best matching cluster also participate in the learning process. The rate of adaptation of the weights decreases away from the winning node, according to the bubble neighborhood function used by Wehrens, Buydens, et al. (2021) $h_{ci}(t)$ where c denotes the winning cluster and i denotes the index of the neighbouring unit, and $h_{ci}(t)$ is assigned a value of 1 for neighbours within an iteratively decreasing radius:

$$h_{ci}(t) = \begin{cases} 1, & \text{if } d_{ci(t)} \leq \sigma(t) \\ 0, & \text{otherwise} \end{cases}$$

where $\sigma(t)$ denotes the radius of the neighborhood. The following formula presents the update rule for the weight vector of unit i:

$$w_i(t+1) = w_i(t) + \alpha(t)h_{ci}(t)[x(t) - w_i(t)]$$

The above three steps are referred to as competition, cooperation and adaptation. The learning rate (or rate of change of the weight vectors (Asan and Ercan, 2012)) decreases linearly (as a function of the increase in iterations) until the algorithm converges and the neighbourhood radius reduces also linearly from a starting value to a stop value. Kohonen (1990) suggests that clusters have a large neighbourhood size to begin with, e.g. a radius of more than half the diameter of the grid.

The final weights are the referencing *codebook* vectors between the clusters and their connected basic strata. Asan and Ercan (2012); Honkela (1997); Kohonen (1990) were referred to in preparing the above summary and the following algorithm description.

Algorithm 9 Self-Organising Maps

1: Set $t = 0$. Initialise a set of weights, the radius and the learning rate.
2: Set $t = t + 1$. Randomly selecting an input (basic stratum) $x(t)$.
3: Select the nearest (winning) cluster using the sum of squares distance $c(t) = \arg\min_i \sum_{j=1}^{m} (x_{tj} - w_{tj,i})^2$.
4: Update the weights for the winning cluster and clusters in its neighbourhood $w_i(t+1) = w_i(t) + \alpha(t) h_{ci}(t)[x(t) - w_i(t)]$. Note: For the winning node the neighborhood function $h_{ci}(t)$ will be equal to 1.
5: Set $t = t + 1$. Adjust the radius and the learning rate.
6: Repeat steps 2 to 5 for a set number of iterations.

SOMs better explore the search space and are less prone to local optima than K-means (Bação et al., 2005). They are particularly useful in solving problems where there are multiple optima, such as our problem. However, the number of grid clusters in the SOM can be quite large. The visualisations of such grids are more of qualitative importance than quantitative.

The basic principle of our research is that similar basic strata be grouped together. In a detailed grid similar basic strata can be mapped to different grid clusters. It is therefore appropriate that similar grid clusters are grouped together into larger clusters.

Clustering SOMs was first proposed by Vesanto and Alhoniemi (2000) using both agglomerative and partitive clustering techniques, in particular K-means. However, they also refer to expectation maximisation and fuzzy clustering of the SOM. They also suggest using a neural gas algorithm to initially abstract the data (instead of the SOM).

Vesanto and Alhoniemi (2000) found that a two-step process of using the SOM to create a prototype which are then clustered creating better quality output in terms of quantitative analysis (e.g. means, medians and ranges) and computation times when compared with direct clustering of the data.

Their goal was not to find the optimal clustering for the data but to get a good insight into the cluster structure of the data for data mining purposes. We compare the outputs when the SOM and neural gas (see section 9.11) are combined with K-means, fuzzy K-means clustering and expectation maximisation. We compare the resulting outputs of this approach, with the outputs from direct clustering. We then consider outputs when a third step is added: the hill-climbing algorithm.

Similarly our goal is not to find the optimal solution, but to get better quality clusterings (strata) in terms of objective function than those attained by the GGA and

SAA - in less computation and training times. For the SOM we use the *Kohonen* package (Wehrens, Buydens, et al., 2021). The inputs are the sample and the following hyperparameters: number of iterations, upper and lower α values and the neighbourhood radius, which we train using the R package *mlrMBO* (Bischla et al., 2017) (see the methodology described in section 6.1).

The SOM outputs needed for the subsequent clustering are a matrix of the codebook vectors (which we subsequently cluster) and a vector with the position of the grid unit onto which each sample unit is mapped. In other words, the codebook vectors are clustered using either K-means clustering, fuzzy K-means clustering or expectation maximisation, and then referenced back to the basic strata. This results in a clustering which is used as a starting solution for the hill-climbing algorithm discussed in section 9.12.

9.10 Fuzzy K-means clustering

In hard clustering, an algorithm such as K-means algorithm attempts to partition a finite collection of elements (in this case sample units) $X = x_1, x_2, \ldots, x_N$ into a collection of K clusters with respect to their similarity, x_i is the i^{th} sample unit; $x_{i,j}$ the j^{th} variable of x_i. If K is an integer, $2 \leq K < N$, the K-clustering of X amounts to subsets of X $(X_1, X_2, \ldots X_K)$, where each cluster has at least one element and is independent of another cluster, and all elements are contained within a cluster.

In soft clustering a sample unit belongs to all clusters (or strata) at varying probabilities. In the fuzzy K-means algorithm (also called the fuzzy c-means algorithm) each centroid of a cluster is the mean of all cases (basic strata) in the data set, weighted by their degree of belonging to that cluster. Like K-means the algorithm is affected by the initial clusters and tends to find local minima (Morissette and Chartier, 2013).

Nonetheless, the algorithm is useful in scenarios where there is some degree of overlap between clusters and it would be useful to consider to what degree a sample unit belongs to various clusters. The soft clustering helps to smooth out the likelihood of the algorithm finding inferior quality local minima (Klawonn, 2004).

The fuzzy K-means clustering algorithm attempts to partition elements (which in this case are basic strata) probabilistically into all sets. The probability of each basic stratum belonging to a cluster in clustering X_k is represented by a $K \times N$

probability matrix U with components $u_{k,i},\ k = 1,\ldots,K, i = 1,\ldots,N$ such that:

$$u_{k,i} = \frac{1}{\sum_{l=1}^{K}\left(\frac{\|\mathbf{x}_i - \mathbf{v}_k\|}{\|\mathbf{x}_i - \mathbf{v}_l\|}\right)^{\frac{2}{m-1}}}$$

where $u_{k,i}$ is a numerical value in $[0,1]$ giving the probability of sample unit x_i belonging to the k^{th} cluster.

The centroid of a cluster (v_k) (and $v = (v_1, v_2 \ldots v_k)$ = vectors of centers) is the mean of all basic strata in that cluster, weighted by their degree of belonging to that cluster ($u_{k,i}$), i.e.

$$\hat{v}_k = \frac{\sum_{i=1}^{N}(\hat{u}_{k,i})^m x_i}{\sum_{i=1}^{N}(\hat{u}_{k,i})^m}$$

where m is a weighting exponent, a hyperparameter, controlling how hard the cluster will be. The higher m is, the softer the cluster will be. In fuzzy clustering we aim to minimise the weighted distance between the sample unit and each centroid and our objective function is:

$$J_m(U,v) = \sum_{i=1}^{N}\sum_{k=1}^{K}(u_{k,i})^m \|x_i - v_k\|^2$$

Using the description provided by Bezdek et al. (1984) as a guide the algorithm operates as follows:

Algorithm 10 Fuzzy K-means

Step 1: Select the number of clusters K , exponential weight $m\ (1 < m < \infty)$, randomly generate clustering and partition matrix U^l, and set the termination criterion ϵ. Also, set the iteration index l to 0.
Step 2: Calculate the fuzzy cluster centroids $v^{l+1} = (v_1, v_2 \ldots v_k)$ using U^l.
Step 3: Calculate the new partition matrix U^{l+1} by using v^{l+1}.
Step 4: If $\left\|U^{l+1} - U^l\right\| > \epsilon$, then set $l = l + 1$ and go to step 2, otherwise stop.

We use the fuzzy clustering method in the *cmeans* function in the *e1071 package* (Meyer et al., 2019) in R. For this the inputs are the basic strata, m the weighting exponent - which we trained using the R package *mlrMBO* (Bischla et al., 2017) (see the methodology described in section 6.1). The output is a matrix of probabilites for each sample unit belonging to a cluster, from these the cluster with the highest probability is taken as the hard clustering.

9.11 Neural gas

Developed by Martinetz et al. (1991), the neural gas algorithm is inspired by the artificial neural network SOM algorithm, using a Hebb-like learning rule, along with a memory decay term, to map multi-dimensional data to typically one or two dimensions. In SOMs the grid for the clusters is fixed and updates are made to the weights of winning clusters in their neighbourhood. In neural gas (NG) however, cluster centroids are not fixed, and their positions move around abruptly during the training, similar to Brownian motion of gas molecules, which helped give the algorithm its name.

The NG algorithm sorts the clusters according to the distance of their reference weights to the basic strata. The level of the adjustment to each weight is determined by their "rank" in this order (Li, 2014). The winning weight takes the largest adjustment. The strength of the adjustment decreases according to a fixed schedule. Because prior knowledge of the grid structure is not required for neural gas, this in tandem with the soft nature of the clusterings may assist the neural gas find better quality clusters than the SOM.

More specifically, the algorithm operates as follows, we initialise a randomly generated set $w = (w_1, \ldots, w_N)$ of codebook vectors (also called cluster centroids and are of the same dimension as the sample units) $w_i \in R^D, i = 1, \ldots, N$. These can be randomly selected points from the data.

A basic stratum vector $v \in V$ is assigned to the cluster represented by the best-matching or "winning" codebook vector $w_{i,v}$ of w for which the distortion error $d(v, w_{i,v})$, e.g., the squared error $\|v - w_{i,v}\|^2$, is minimal. This procedure divides the basic strata V into a number of clusters

$$V_i = v \in \|v - w_i\| \leq \|v - w_j\| \; \forall j$$

out of which each sample unit (basic stratum) v is described by the corresponding codebook vector w_i.

Each time a randomly selected sample unit v is presented, we determine the "neighborhood-ranking" $(w_{i,0}, w_{i,1} \ldots, w_{i,N-1})$ of the codebook vectors with $w_{i,0}$ being closest to v , $w_{i,1}$ being second closest to v , and $w_{i,k}$, where $k = 0, \ldots, N-1$ being the reference vector for which there are k vectors w_j with $\|v - w_j\| < \|v - w_{i,k}\|$. If we denote the number k associated with each vector w_i by k_i, which depends on v and the whole set $w = (w_1, \ldots, w_N)$ of reference vectors, then each

weight vector of the ordered sequence is adapted by

$$w_i^t = w_i^{t-1} + \epsilon\, e^{\frac{-k_i}{\lambda}}(v - w_i^{t-1}); \quad i = 1, ..., N$$

where t is the iteration and T the total number of iterations. This adaptation step is a "winner-takes-most" rather than a "winner-takes-all" rule. The step size $\epsilon \in [0,1]$ decreases with increasing distances $\|\|v - w_i\|$, and $e^{\frac{-k_i}{\lambda}} \in [0,1]$, where $e^{\frac{-k_i}{\lambda}}$ is 1 for $k_i = 0$, and decays to zero for increasing k_i; with a decay constant λ. If λ decays to 0 then only the winner $w_{i,0}$ is updated. The parameters ϵ and λ are iteratively reduced using the following:

$$\epsilon_t = \epsilon_0 \left(\frac{\epsilon_T}{\epsilon_0}\right)^{(t/T)}$$

$$\lambda_t = \lambda_0 \left(\frac{\lambda_T}{\lambda_0}\right)^{(t/T)}$$

At the final iteration, each basic stratum assigned to the nearest w_i, which leads to final clustering. We referred to Daszykowski et al. (2002); Martinetz et al. (1991, 1993); Pena and Fyfe (2006) in preparing the following algorithm summary:

Algorithm 11 Neural gas

1: Set $t = 0$. Assign initial values to the weights $w_i \in R^n$.
2: Set $t = t + 1$. Select a sample unit, or vector v, from the input data.
3: State For each cluster centroid i determine the number k_i of centroids j with $\|v - w_j\| < \|v - w_i\|$ by determining the sequence $(i_0, i_1, ..., i_{N-1})$ of cluster centroids with $\|v - w_{i_0}\| < \|v - w_{i_1}\| < \cdots < \|v - w_{i_{N-1}}\|$
4: Adapt the weights accordingly: $w_i^{t+1} = w_i^t + \epsilon\, e^{\frac{-k_i}{\lambda}}(v - w_i^t);\ i = 1, ..., N$.
5: if $t < T$ return to step 2, else assign each sample unit to its closest centroid.

We implement the neural gas method using the *cclust* function in the *cclust* package (Dimitriadou et al., 2020). The inputs are the sample and the upper and lower values of the λ and ϵ hyperparameters, which we trained using the R package *mlrMBO* (Bischla et al., 2017) (using the methodology described in section 6.1). The output is a clustering is further clustered using either K-means clustering, fuzzy K-means clustering or expectation maximisation. This clustering is utilised as a starting solution for the hill-climbing algorithm discussed in section 9.12.

9.12 Hill-climbing algorithm with delta evaluation

The term *hill-climbing* implies a maximisation problem, however the equivalent optimisation of the objective function can be applied in a downward sense also, i.e. for minimisation problems. Hill-climbing is used to describe both types of problems without loss of generality (Michalewicz and Fogel, 2013). We apply hill-climbing to search for better neighbouring solutions in our problem. In doing so, we create a new solution by randomly taking a basic stratum from one stratum in the current solution and moving it to another stratum.

We then evaluate the quality of that new state and keep the change if its quality is found to be better. If not we return to the previous solution state. This process continues until there has been no further improvement in solution quality after a set number of iterations. The similarity between solutions was used in chapter 8 to achieve gains in computation times by using delta evaluation for a simulated annealing algorithm.

Hill-climbing is analogous with the advanced state of a simulated annealing algorithm when the tunable probability of accepting inferior solutions has decayed to a sufficient degree that the algorithm only accepts better quality solutions. The hill-climbing algorithm outlined below uses the same perturbation approach as the simulated annealing algorithm in chapter 8. We also apply delta evaluation in the context of hill-climbing to help speed up computation times.

The reason for using hill-climbing is our assumption that the starting solutions attained by combining SOM or neural gas with K-means clustering, fuzzy K-means clustering or expectation maximisation clustering will be of a sufficient quality that the hill-climbing algorithm need only converge on a local minimum (which will be in close proximity to the global minimum).

We have fixed the number of basic strata to be moved from one stratum to another as $q = 1$ and set 1,000 as the *stopping-point* for the number of iterations without an improvement in solution quality, after which each consecutive pair of solution qualities 1,000 iterations apart are rounded to 2 decimal places and compared. Therefore there will be no training of hyper-parameters required for this algorithm. We base the description of the algorithm below on the summary in (Lin and Kernighan, 1973):

Algorithm 12 Hill-climbing algorithm

1: Initialise a starting solution, S
2: Attempt to find an improved solution S' by randomly assigning $q = 1$ basic strata from stratum h to stratum h'.
3: If an improved solution is found using delta evaluation, i.e., $f(S') < f(S)$, then replace S by S'.
4: If no improved solution can be found after $1,000$ iterations, S is a locally optimum solution, alternatively repeat from Step 2.

9.13 Computer specifications

Table 9.13.1 below is a reproduction of the equivalent table in chapter 8 and provides details of the processing platform used for these experiments. Further details including including the R version, packages used and number of cores is available in the description provided with the original version of the table.

Table 9.13.1: *Specifications of the processing platform*

Specification	Details	Notes
Processor	AMD Ryzen 9 3950X 16-Core Processor, 3493 Mhz	
Cores	16 Core(s)	
Logical Processors	32 Logical Processor(s)	32 cores in R
System Model	X570 GAMING X	
System Type	x64-based PC	
Installed Physical Memory (RAM)	16.0 GB	
Total Virtual Memory	35.7 GB	
OS Name	Microsoft Windows 10 Pro	

9.14 Empirical comparisons for atomic strata

As discussed in chapter 3, atomic strata are constructed from the Cartesian product of discrete auxiliary (X) variables. Each atomic stratum contains a number of records with values for their constituent numerical target (Y) variables. These are summarised by calculating the estimates of the means and standard deviations for each target variable. We used the *mlrMBO* package to train the hyperparameters (for more details on this methodology see section 6.1) for the experiments below in all cases except for grid size in the SOM which we adjusted according to data set size.

9.14.1 Benchmark results - atomic strata

Table 9.14.2 is a reproduction of the summary by the above data sets of the evaluation time, sample size and number of strata for the Grouping Genetic Algorithm (GGA) and Simulated Annealing Algorithm (SAA) as found by the experiments conducted

in chapter 8. We compare these results with the results below to provide a benchmark for comparing solution quality and total training time.

Table 9.14.2: *Summary by data set of the evaluation time, sample size and number of strata for the Grouping Genetic Algorithm (GGA) and Simulated Annealing Algorithm (SAA)*

	GGA			SAA		
Data set	**Sample Size**	**Execution Time**	**Total Training Time**	**Sample Size**	**Execution Time**	**Total Training Time**
Swiss Municipalities	128.69	753.82	10,434.30	125.17	248.91	8,808.63
American Community Survey, 2015	10,136.50	13,146.25	182,152.46	10,279.44	517.76	6,822.42
Kiva Loans	6,756.19	15,669.11	288,946.79	6,646.67	664.30	7,549.87

9.14.2 Swiss municipalities

Recall that the Swiss municipalities data set (Barcaroli et al., 2014) has 2,896 records, referring to Swiss municipalities in 2003. Each municipality belongs to one of 7 regions (domains). We stratify and select a sample for each of these regions, with the combined stratification for all regions being the solution, and the total sample size indicating the quality of the solution. The target and auxiliary variables are outlined in table 9.14.3. The auxiliary variables *POPTOT.cat* and *Hapoly.cat* are categorical versions of the continuous *POPTOT* and *Hapoly* variables. For further details on the target and auxiliary variables refer to (Ballin and Barcaroli, 2020).

Table 9.14.3: *Summary of the variables and number of records for the Swiss Municipalities data set*

Data set	**Target Variables**	**Auxiliary Variables**	**Number of Records**
Swiss Municipalities	Surfacesbois, Airbat	POPTOT.cat , Hapoly.cat	2,896

In setting a maximum number of clusters of 30 we obtained an initial K-means solution ranging between 8 and 20 clusters for the seven domains. For each domain the solution is broken down as follows:

Domain/Region	1	2	3	4	5	6	7	
Number of Clusters (Strata)	8	20	13	17	17	15	14	Total
Sample Size	15.96	45.15	28.16	46.18	45.74	29.31	36.41	246.90

A visual examination of the cluster plot for Domain 1 in Figure 9.1 indicates 8 distinct clusters, i.e. they are linearly separated. However, the majority of clusters (i.e. those in the lower left hand corner) are not well-separated (clearly dissimilar). The sample size of 246.90 may be further improved by considering some degree of overlap and soft clustering, i.e. assigning probabilities for the basic strata to belong

to each cluster, before choosing the cluster with the highest probability for each basic stratum as the hard clustering.

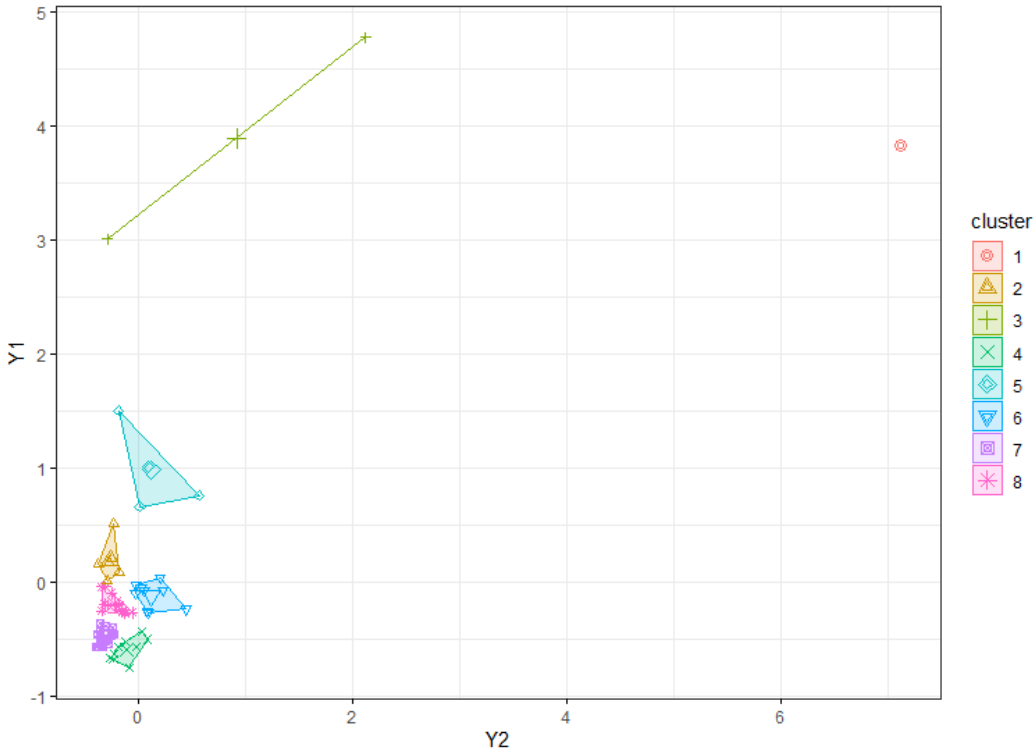


Figure 9.1: *Cluster plot of K-means solution for Domain 1*

Table 9.14.4 below provides the sample size, execution time, total execution time and aggregated total time for fine tuning the hyperparameters for each algorithm (or staged combination of algorithms). The **Stage** variable indicates whether the sample size and times were obtained for the algorithm in one stage or whether the results were obtained through a staged combination of various algorithms. For example the sample size of 246.90 was obtained after 1 stage by the *KmeansSolution* algorithm (Ballin and Barcaroli, 2020). This corresponds to the largest sample size. The time of 1.80 seconds was also the quickest total execution time.

On the other hand the smallest sample size of 130.92 was obtained after combining three stages of the SOM (stage 1), EM (stage 2) and hill-climbing (stage 3) algorithms. This was achieved in a time of 178.76 seconds (109.84 + 68.92). The was also the longest time.

The smallest sample size in table 9.14.4 should be considered in the context of the sample size of 125.17 obtained in 248.91 seconds after a fine-tuning time of 8,808.63 seconds by the simulated annealing algorithm (table 9.14.2).

Depending on the initial quality of the solution the hill-climbing algorithm

converges on diverse local minima resulting in varying final solution quality. Furthermore, the hill-climbing algorithm search for local minima is stochastic in nature - meaning that for the same starting solution the hill-climbing algorithm may also find diverse local minima on each attempt.

Notwithstanding the larger differences in final solution quality the same point could be made for training and the multi-stage combination of the above algorithms (or other clustering algorithms) with hill-climbing. These results should not be taken as deterministic, rather as a means of providing greater scope for selecting a solution that is good enough in a computing time that is small enough (Sörensen and Glover, 2013).

Table 9.14.4: *Comparison of Sample Size, Time and Total Execution Time for the various single algorithm or multi-algorithm combinations for the Swiss Municipalities Data set experiment*

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	KmeansSolution	246.90	1.80	1.80	
2	Hill Climbing	165.05	33.17	33.17	34.97

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and Kmeans	198.03	3.34	65.30	
3	HillClimbing	141.46	69.25	69.25	134.55

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and Kmeans	204.00	0.73	14.56	
3	HillClimbing	146.25	58.37	58.37	72.93

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Expectation Maximisation	196.12	1.34	1.34	
2	HillClimbing	133.21	68.98	68.98	70.32

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and EM	183.13	5.39	109.84	
3	HillClimbing	**130.92**	68.92	68.92	**178.76**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and EM	178.19	1.47	28.42	
3	HillClimbing	134.84	79.07	79.07	107.49

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Fuzzy Clustering	175.38	0.27	4.89	
2	HillClimbing	133.91	91.31	91.31	96.20

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and FC	170.64	3.33	64.92	
3	HillClimbing	139.87	43.47	43.47	108.39

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and FC	174.34	0.66	13.63	
3	HillClimbing	136.71	71.76	71.76	85.39

Figure 9.2 plots the aggregated total time against sample size by algorithm combination for the Swiss Municipalities dataset. The graph demonstrates that the SOM, EM and hill climbing combination attains the lowest sample size but with the second largest time. On the other hand, the EM and hill climbing combination attained a similar sample size but in less time. The K-means and hill climbing

combination was the fastest, but the sample size was of the least quality (highest value). A number of combinations are clustered around similar sample sizes and times.

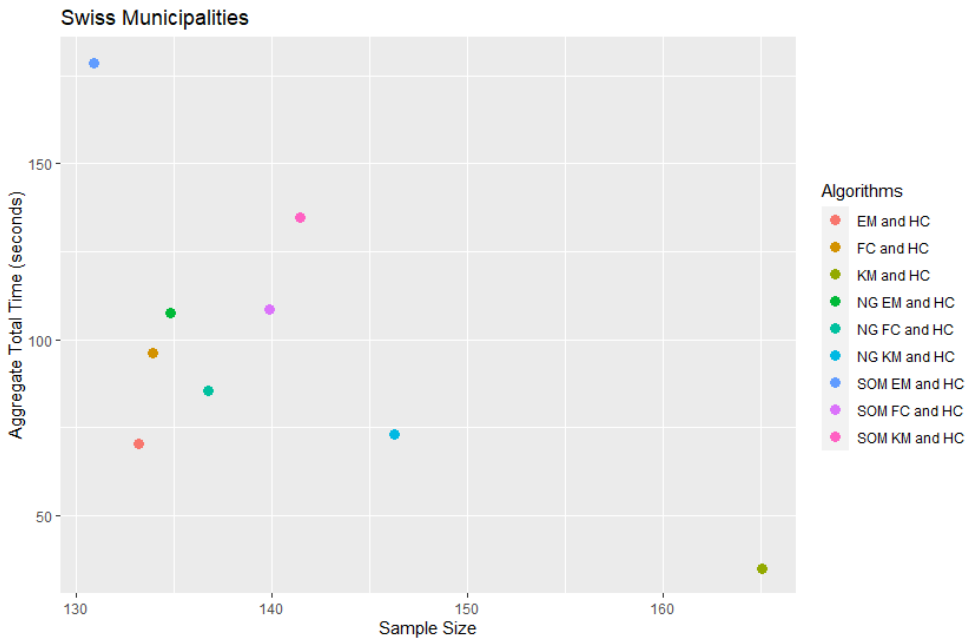


Figure 9.2: Plot of Aggregated Total Time versus Sample Size by Algorithm combination for the Swiss Municipalities dataset

Table 9.14.5 summarises the fine-tuned hyper-parameters for the algorithms where fine-tuning was implemented.

Table 9.14.5: *Summary of the fine-tuned hyperparameters for the various single-stage or multi-stage algorithm combinations for the Swiss Municipalities Data set*

SOM and K-means	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	1081	0.809753218616724	0.031088352131124	0.83249740019602
	Lambda High	**Lambda Low**	**Stepsize High**	**Stepsize Low**
NG and K-means	10.6110575335055	0.959806751475908	0.499173569748991	0.016996916064339
	Iterations	**Alpha High**	**Alpha Low**	**Radius**
SOM and EM	7695	0.113771222653394	0.04425198940754	0.061954274246033
	Lambda High	**Lambda Low**	**Stepsize High**	**Stepsize Low**
NG and EM	8.23248667684384	0.67836881950614	0.146985992956907	0.032059727899148
	m			
Fuzzy Clustering	3			
	Iterations	**Alpha High**	**Alpha Low**	**Radius**
SOM and Fuzzy	1462	0.148577394044338	0.030700312647367	0.008935026169402
	Lambda High	**Lambda Low**	**Stepsize High**	**Stepsize Low**
NG and Fuzzy	8.23248667684384	0.67836881950614	0.146985992956907	0.032059727899148

Figure 9.3 visualises the clusters (strata) for Domain 1 of the Swiss Municipalities Data set for the solution found after combining SOM, EM and hill-climbing when represented as scaled data. This solution has 6 clusters in contrast to Figure 9.1 which has 8 clusters for the K-means solution. Clusters 5 and 7 show evidence of

the overlap that occurs with soft clustering. Cluster 8 stretches to take in an outlier, which is unlikely to have occurred with hard clustering, or without the hill-climbing algorithm.

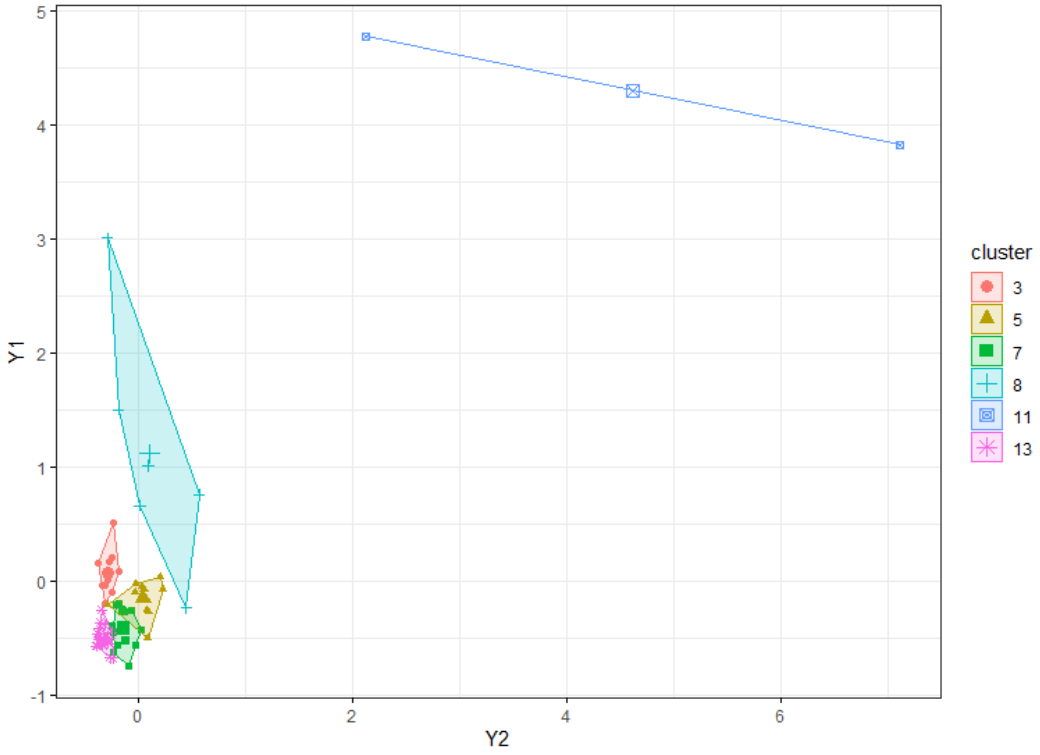


Figure 9.3: *Cluster plot of solution after combining SOM, EM and hill-climbing*

9.14.3 American Community Survey, 2015

A summary of the Target and Auxiliary variables and number of records for the American Survey 2015 data set is provided in table 9.14.6 below. For further details on the data set refer to chapter 7.

Table 9.14.6: *Summary of the variables, number of records and domains for the American Survey 2015 data set*

Data set	Target Variables	Auxiliary Variables	Number of Records	Number of Domains
American Community Survey, 2015	HINCP, VALP, SMOCP, INSP	BLD, TEN, WKEXREL, WORKSTAT, HFL, YBL	619,747	51

The sample size, execution time and total execution time for fine tuning the hyperparameters for each algorithm (or staged combination of algorithms) are outlined in table 9.14.7 below. The sample size of 9,065.45 in a time of 8,426.67 seconds for the self-organising maps, fuzzy clustering and hill-climbing combination compares favourably with the lowest sample size of 10,136.50 and a total training

time of 182,152.46 seconds attained by the grouping genetic algorithm in table 9.14.2.

Table 9.14.7: *Comparison of Sample Size, Time and Total Execution Time for the various single stage or multi-stage algorithm combinations for the American Community Survey, 2015 Data set*

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	KmeansSolution	11,769.53	44.27	44.27	
2	HillClimbing	9,322.07	5,652.85	5,652.85	**5,697.12**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and Kmeans	12,787.19	86.48	2,142.86	
3	HillClimbing	9,232.31	6,590.34	6,590.34	**8,733.20**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and Kmeans	12,398.84	107.58	2,181.06	
3	HillClimbing	9,290.53	6,019.93	6,019.93	**8,200.99**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Expectation Maximisation	13,136.49	32.94	32.94	
2	HillClimbing	9,755.61	5,641.87	5,641.87	**5,674.81**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and EM	13,018.89	84.94	12,231.26	
3	HillClimbing	9,592.40	6,777.39	6,777.39	**19,008.65**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and EM	12,050.93	133.64	2,636.87	
3	HillClimbing	9,368.93	5,755.20	5,755.20	**8,392.07**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Fuzzy Clustering	13,350.35	4.89	352.46	
2	HillClimbing	9,255.64	6,245.89	6,245.89	**6,598.35**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and FC	12,088.80	88.87	2,127.86	
3	HillClimbing	**9,065.45**	6,298.81	6,298.81	**8,426.67**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and FC	12,580.33	123.31	2,570.40	
3	HillClimbing	9,249.05	5,506.89	5,506.89	**8,077.29**

Figure 9.4 demonstrates that the SOM, FC and hill climbing combination attains the lowest sample size and with a greater time. On the other hand, the EM and hill climbing combination attained the largest sample size but in one of the least times. A number of combinations are clustered around similar sample sizes and times.

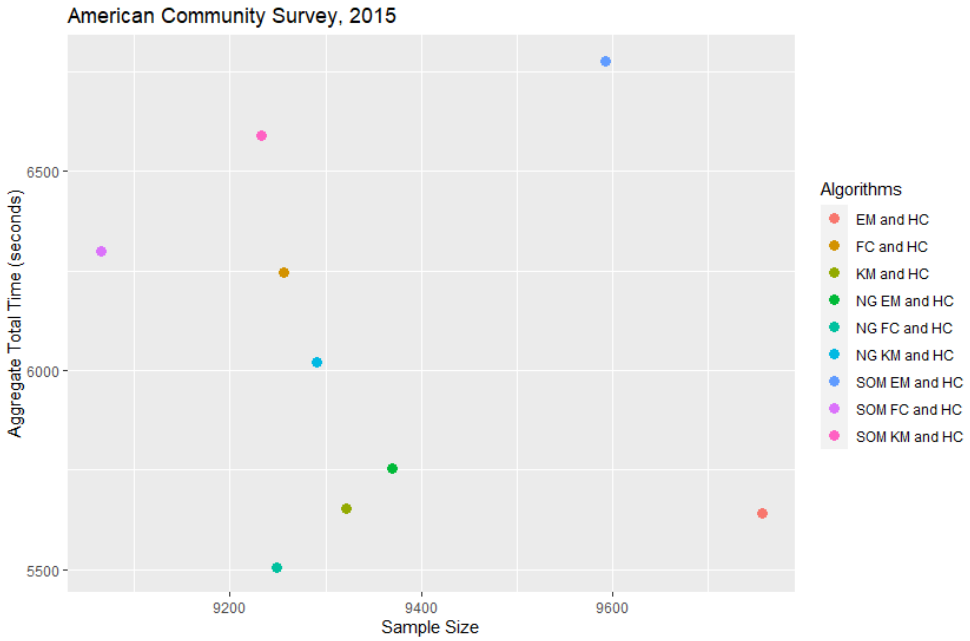


Figure 9.4: Plot of Aggregated Total Time versus Sample Size by Algorithm combination for the American Community Survey, 2015 dataset

Table 9.14.8 provides the summary of the fine-tuned hyper-parameters for the relevant algorithms.

Table 9.14.8: *Summary of the fine-tuned hyperparameters for the various single-stage or multi-stage algorithm combinations for the American Community Survey, 2015 Data set*

SOM and K-means	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	1490	0.211798547655744	0.014647548302976	0.569154786943982
NG and K-means	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	27.9195691507683	0.953798242114481	0.22725191488931	0.005327383472354
SOM and EM	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	351	0.165795508368377	0.003116705920671	0.272235491431915
NG and EM	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	6.40801811285783	0.410143870242871	0.11268728621304	0.005130624421989
Fuzzy Clustering	**m**			
	7			
SOM and FC	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	351	0.134122729571279	0.002215617049982	0.552123088934338
NG and FC	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	13.0783552149509	0.329139476246046	0.384919628095097	0.005166985979477

9.14.4 Kiva Loans

A summary of the target and auxiliary variables, number of records and number of domains for the Kiva Loans data set is provided in table 9.14.9 below.

Table 9.14.9: *Summary of the Target and Auxiliary variables, number of records and number of domains for the Kiva Loans data set*

Data set	Target Variables	Auxiliary Variables	Number of Records	Number of Domains
Kiva Loans	term_in_months, lender_count, loan_amount	sector, currency, activity, region, partner_id	614,361	73

The sample size, execution time and total execution times for fine tuning the hyperparameters for each algorithm (or staged combination of algorithms) are outlined in table 9.14.10 below. The lowest sample size of 6,326.35 and total training time of 6,390.36 seconds for the fuzzy clustering and hill-climbing combination compares favourably with the lowest sample size of 6,646.67 and total training time of 7,549.87 seconds attained by the simulated annealing algorithm in table 9.14.2.

Table 9.14.10: *Comparison of Sample Size, Time and Total Execution Time for the various single stage or multi-stage algorithm combinations for the Kiva Loans Data set*

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	KmeansSolution	7,609.97	36.33	36.33	
2	HillClimbing	6,355.91	5,201.12	5,201.12	5,237.45

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and Kmeans	8,235.49	46.69	1,376.00	
3	HillClimbing	6,686.45	4,594.16	4,594.16	5,970.16

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and Kmeans	7,521.78	68.12	1,368.16	
3	HillClimbing	6,352.77	5,379.39	5,379.39	6,747.55

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Expectation Maximisation	8,016.05	34.89	34.89	
2	HillClimbing	6,696.26	9,826.94	9,826.94	9,861.83

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and EM	9,040.16	622.99	11,717.33	
3	HillClimbing	7,173.30	12,342.69	12,342.69	**24,060.02**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and EM	7,433.73	92.04	1,861.40	
3	HillClimbing	6,378.58	5,139.70	5,139.70	7,001.10

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Fuzzy Clustering	7,532.95	17.81	371.75	
2	HillClimbing	**6,326.35**	6,018.61	6,018.61	6,390.36

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and FC	7,707.41	50.03	1,357.19	
3	HillClimbing	6,487.29	5,818.46	5,818.46	7,175.65

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and FC	7,470.76	84.57	1,671.55	
3	HillClimbing	6,361.31	5,896.17	5,896.17	7,567.72

Figure 9.5 demonstrates that a number of combinations are clustered around similar low sample sizes and times. The FC and hill climbing combination attained the lowest sample size and with one of the lowest times. On the other hand, the SOM, EM and hill climbing combination attained the largest sample size in the most time.

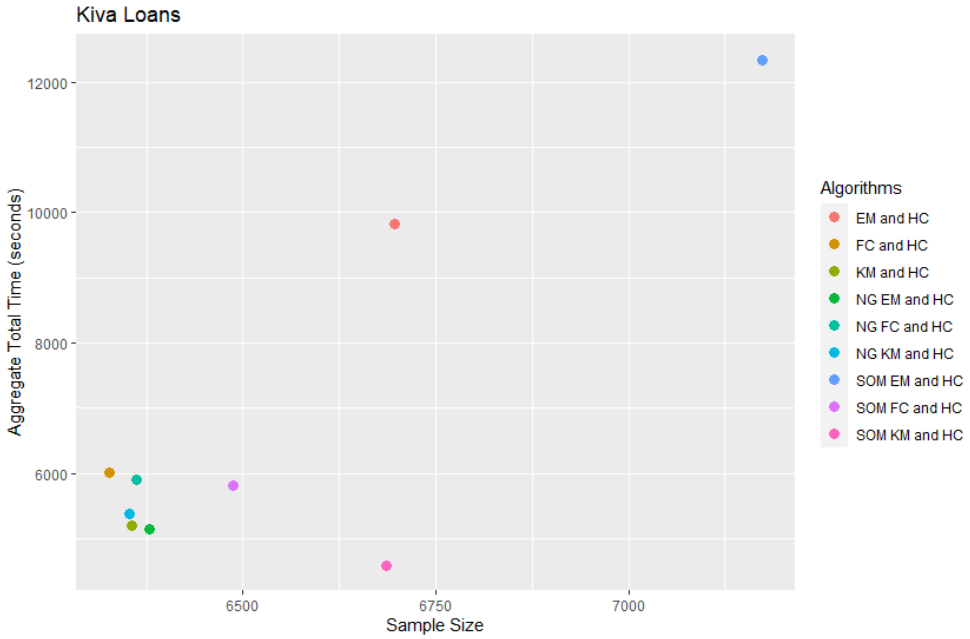


Figure 9.5: Plot of Aggregated Total Time versus Sample Size by Algorithm combination for the Kiva Loans dataset

Table 9.14.11 provides the summary of the fine-tuned hyper-parameters for the relevant algorithms.

Table 9.14.11: *Summary of the fine-tuned hyperparameters for the various single-stage or multi-stage algorithm combinations for the American Community Survey, 2015 Data set*

SOM and K-means	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	101	0.92384807669335	0.097029295450863	0.680610206832993
NG and K-means	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	16.2579368586406	0.068847852889988	0.258579228835623	0.005948537735878
SOM and EM	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	9864	0.878176548467018	0.033124931246478	0.462181919813156
NG and EM	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	15.6835266864393	0.906532369809407	0.341722599971882	0.041738065618349
Fuzzy Clustering	**m**			
	2			
SOM and FC	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	325	0.178468058524982	0.015586729150577	0.00040918374223
NG and FC	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	20.0046674131416	0.865994203313719	0.23247297228314	0.04890728238225

9.15 Empirical comparisons for continuous strata

Unlike the atomic strata above, continuous strata are not numerical or discrete. We use the same methodology and algorithms outlined above to the continuous strata and the results are presented below (i.e. we apply the same procedures on the continuous strata). For each of the experiments below the target and auxiliary variables are the same.

9.15.1 Benchmark Results - Continuous Strata

For comparison purposes table 9.15.12 contains the evaluation time, sample size and number of strata for the Grouping Genetic Algorithm (GGA) and Simulated Annealing Algorithm (SAA) for the data sets utilised in the experiments below.

Table 9.15.12: *Summary by data set of the evaluation time, sample size and number of strata for the GGA and SAA*

Data set	Sample Size	Execution Time	Total Training Time	Sample Size	Execution Time	Total Training Time
SwissMunicipalities	128.69	753.82	10,434.30	120.00	213.44	1,905.82
American Community Survey, 2015	4,197.68	22,016.95	227,635.51	3915.48	13,351.19	169,115.92
Kiva Loans	3,062.33	3,232.78	48,746.61	3,017.79	300.16	4,149.06

9.15.2 Swiss municipalities

The target and auxiliary variables for the Swiss Municipalities data set are recorded in table 9.15.13.

Table 9.15.13: *Summary of the Target and Auxiliary variables for the Swiss Municipalities data set*

Data set	Target Variables	Auxiliary Variables
Swiss Municipalities	Surfacesbois, Airbat	Surfacesbois, Airbat

In table 9.15.14 the multi-stage combination of the neural gas, expectation maximisation and hill-climbing algorithms provides the lowest sample size of 110 for both the atomic and continuous methods in an total training time of 95.33 seconds. This compares favourably with the lowest sample size of 120 and total training time of 1,905.82 seconds attained by the simulated annealing algorithm. The longest total training time of 547.11 seconds was for the multi-stage combination of the self-organising maps, expectation maximisation and hill-climbing algorithms.

Table 9.15.14: *Comparison of Sample Size, Time and Total Execution Time for the various single stage or multi-stage combinations for the Swiss Municipalities Data set experiment*

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	KmeansSolution	271.07	1.89	1.89	
2	K-means and Hill Climbing	229.00	30.05	30.05	31.94

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and Kmeans	150.58	4.03	96.55	
3	HillClimbing	125.31	139.69	139.69	236.24

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and Kmeans	229.09	3.30	66.10	
3	HillClimbing	189.00	26.53	26.53	92.63

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Expectation Maximisation	174.01	2.02	2.02	
2	HillClimbing	132.66	131.44	131.44	133.46

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and EM	165.35	14.12	342.86	
3	HillClimbing	124.48	204.25	204.25	**547.11**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and EM	124.96	2.90	56.72	
3	HillClimbing	**110.00**	38.61	38.61	95.33

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Fuzzy Clustering	144.97	0.47	8.55	
2	HillClimbing	125.17	49.88	49.88	58.43

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and FC	129.64	4.59	94.87	
3	HillClimbing	118.09	130.38	130.38	225.25

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and FC	138.23	1.76	35.85	
3	HillClimbing	123.95	67.91	67.91	103.76

The total training time for the self-organising maps algorithm in the self-organising maps, expectation maximisation and hill-climbing combination is significantly higher than total training times for the other combinations. This along with the run-time for the would contribute to explaining the longer total training time for the this multi-stage combination.

Figure 9.6 demonstrates that the NG, EM and hill climbing algorithm attains the smallest sample size in a relatively low time. A number of combinations return similar sample sizes but with varying times. The KM and hill climbing combination returns the largest sample size in the least time.

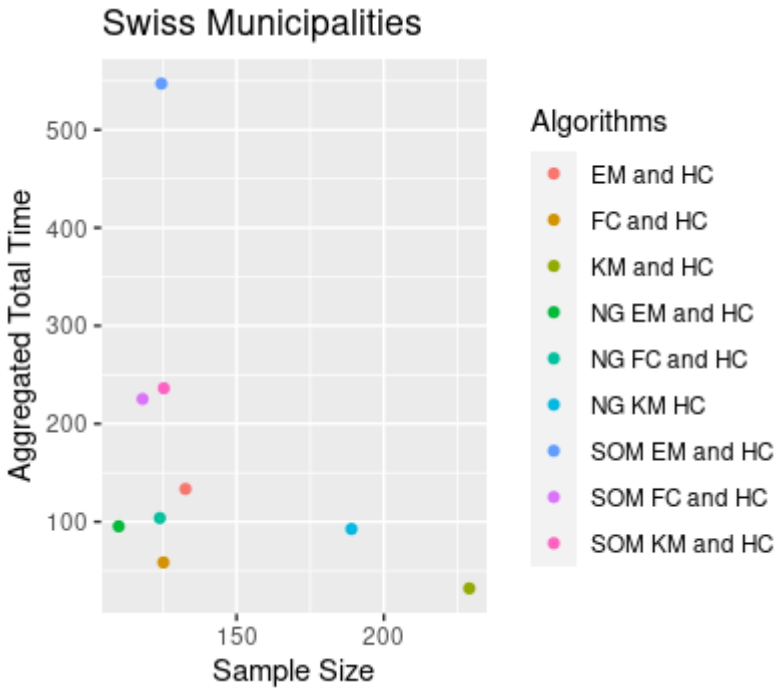


Figure 9.6: Plot of Aggregated Total Time versus Sample Size by Algorithm combination for the Swiss Municipalities dataset

Table 9.15.15 contains the the fine-tuned hyperparameters for the various single-stage or multi-stage algorithm combinations for the Swiss Municipalities dataset.

Table 9.15.15: *Summary of the fine-tuned hyperparameters for the various algorithm combinations for the Swiss Municipalities data set*

SOM and K-means	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	146	0.955179772565607	0.097093850745587	0.648499123915099
NG and K-means	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	1.92036896656643	0.408302785790299	0.483817725175832	0.016913660635487
SOM and EM	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	5,314	0.860526216045479	0.036380460266579	0.266383153636882
NG and EM	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	1.02031964830343	0.965802849282413	0.49419928787723	0.046602493289255
Fuzzy Clustering	**m**			
	2			
SOM and FC	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	361	0.463767154766247	0.003181504371105	0.00137499391567
NG and FC	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	21.617934961514	0.764578349819259	0.133934543593412	0.080908668275684

9.15.3 American Community Survey, 2015

A summary of the target and auxiliary variables for the American Survey 2015 data set is provided in table 9.15.16 below.

Table 9.15.16: *Summary of the Target and Auxiliary variables, number of records and number of domains for the American Survey 2015 data set*

Data set	Target Variables	Auxiliary Variables	Number of Records	Number of Domains
American Community Survey, 2015	HINCP, VALP, SMOCP, INSP	BLD, TEN, WKEXREL, WORKSTAT, HFL, YBL	619,747	51

The results of the various single stage or multi-stage algorithm combinations for the American Community Survey, 2015 data set experiment are presented in table 9.15.16. Note that for the K-means Hartigan-Wong method the R version of the algorithm encountered difficulties and failed to converge for either 100 (initially) or $1,000$ iterations. We used the MacQueen et al. (1967) method instead - moving from 100 to $1,000$ iterations in order to attain convergence.

The minimum sample size attained was 2,753.55 with a total training time of 41,740.96 seconds for the staged combination of the neural gas, fuzzy clustering and hill-climbing algorithms. This compares favourably with the lowest sample size of 3,915.48 and total training time of 169,115.92 seconds attained by the simulated annealing algorithm in table 9.15.12. The longest total training time was 78,344.99 seconds for the K-means and hill-climbing combination.

Table 9.15.17: *Comparison of Sample Size, Time and Total Execution Time for the various single stage or multi-stage algorithm combinations for the American Community Survey, 2015 Data set*

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)	Note
1	KmeansSolution	8,864.89	109.72	109.72		
2	HillClimbing	3,015.11	78,235.27	78,235.27	**78,344.99**	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1 and 2	SOM and Kmeans	3,819.99	313.32	8,322.50		
3	HillClimbing	3,310.36	9,882.47	9,882.47	18,204.97	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1 and 2	NG and Kmeans	3,168.46	657.87	13,449.13		*MCQUEEN
3	HillClimbing	2,802.93	6,655.40	6,655.40	20,104.53	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1	Expectation Maximisation	5,520.72	141.66	141.66		
2	HillClimbing	3,443.64	46,108.23	46,108.23	46,249.89	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1 and 2	SOM and EM	4,441.14	287.36	9,301.32		
3	HillClimbing	3,350.65	24,693.78	24,693.78	33,995.10	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1 and 2	NG and EM	3,137.56	670.08	13,658.88		
3	HillClimbing	2,808.22	9,281.39	9,281.39	22,940.27	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1	Fuzzy Clustering	5,775.91	23.27	2,090.77		
2	HillClimbing	2,825.38	51,268.38	51,268.38	53,359.15	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1 and 2	SOM and FC	3,798.98	260.48	7,904.15		
3	HillClimbing	3,193.08	12,691.15	12,691.15	20,595.30	
Stage	**Algorithm**	**Sample Size**	**Time (Seconds)**	**Total Execution Time (Seconds)**	**Aggregate Total Time (seconds)**	**Note**
1 and 2	NG and FC	3,650.26	653.14	13,108.87		
3	HillClimbing	**2,753.55**	28,632.09	28,632.09	41,740.96	

Figure 9.7 plots the aggregated total time against Sample Size by algorithm combination for the Swiss Municipalities dataset. The graph demonstrates that the EM and hill climbing algorithm attains the smallest sample size in a relatively low time. A number of combinations return similar sample sizes but with varying times. SOM, EM and hill climbing combination returns the largest sample size in the longest time.

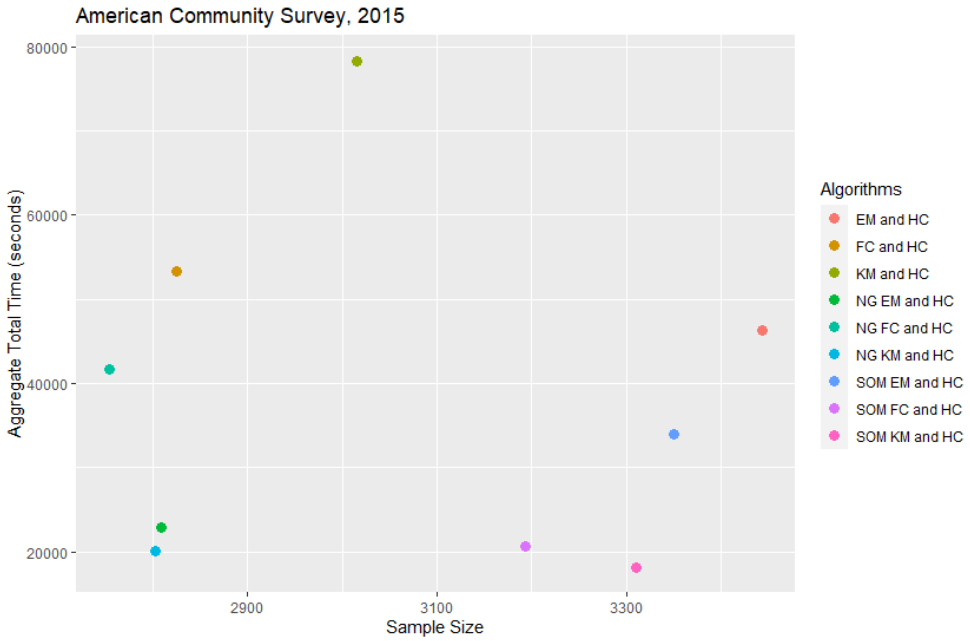


Figure 9.7: Plot of Aggregated Total Time versus Sample Size by Algorithm combination for the American Community Survey, 2015 dataset

Table 9.15.18 summarises the hyper-parameters and relevant algorithms for which the above results.

Table 9.15.18: *Summary of the fine-tuned hyperparameters for the various single-stage or multi-stage algorithm combinations for the American Community Survey, 2015 Data set*

SOM and K-means	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	362	0.235757044486596	0.012774618075309	0.484338735244991
NG and K-means	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	22.8202557664311	0.916501772178628	0.499675245659364	0.008470696479746
SOM and EM	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	268	0.170893752936649	0.004722280048395	0.557606799123925
NG and EM	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	29.1800880556926	0.25795004530251	0.266119958357885	0.019287368137506
Fuzzy Clustering	**m**			
	11			
SOM and FC	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	249	0.124999162557325	0.052358677482419	0.073764869553736
NG and FC	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	5.09321314570995	0.984287188866795	0.287914934710588	0.00526985258796

9.15.4 Kiva Loans

In table 9.15.19 the lowest sample size of 1,752.82 was attained using a combination of expectation maximisation and hill-climbing, in a total training time of 7,428.64 seconds. This compares favourably with the lowest sample size of 3,017.79 and total training time of 4,149.06 seconds attained by the simulated annealing algorithm in table 9.15.12. The longest total training time was 14,613.90 seconds for the self-organising maps, expectation maximisation and hill-climbing combination.

Table 9.15.19: *Summary of the variables, number of records and domains for the Kiva Loans data set*

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	KmeansSolution	3,901.98	43.60	43.60	
2	HillClimbing	2,001.55	5,941.14	5,941.14	5,984.74

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and Kmeans	3,005.53	108.91	2,801.87	
3	HillClimbing	1,876.29	4,692.34	4,692.34	7,494.21

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and Kmeans	2,636.01	160.70	3,269.08	
3	HillClimbing	2,043.52	3,171.44	3,171.44	6,440.52

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Expectation Maximisation	3,496.17	58.11	58.11	
2	HillClimbing	**1,752.82**	7,370.53	7,370.53	7,428.64

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and EM	4,452.22	188.69	3,655.43	
3	HillClimbing	2,150.91	10,958.47	10,958.47	**14,613.90**

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and EM	3,198.13	125.70	2,484.06	
3	HillClimbing	1,840.88	6,517.68	6,517.68	9,001.74

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1	Fuzzy Clustering	3,885.66	20.27	452.67	
2	HillClimbing	1,879.47	6,559.08	6,559.08	7,011.75

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	SOM and FC	2,547.45	98.68	2,878.54	
3	HillClimbing	1,818.98	4,501.45	4,501.45	7,379.99

Stage	Algorithm	Sample Size	Time (Seconds)	Total Execution Time (Seconds)	Aggregate Total Time (seconds)
1 and 2	NG and FC	3,351.17	93.50	1,875.48	
3	HillClimbing	1,869.71	9,528.57	9,528.57	11,404.05

Figure 9.8 plots the aggregated total time against Sample Size by algorithm combination for the Swiss Municipalities dataset. The graph demonstrates that the EM and hill climbing algorithm attains the smallest sample size in a relatively low time. A number of combinations return similar sample sizes but with varying times. SOM, EM and hill climbing combination returns the largest sample size in the longest time.

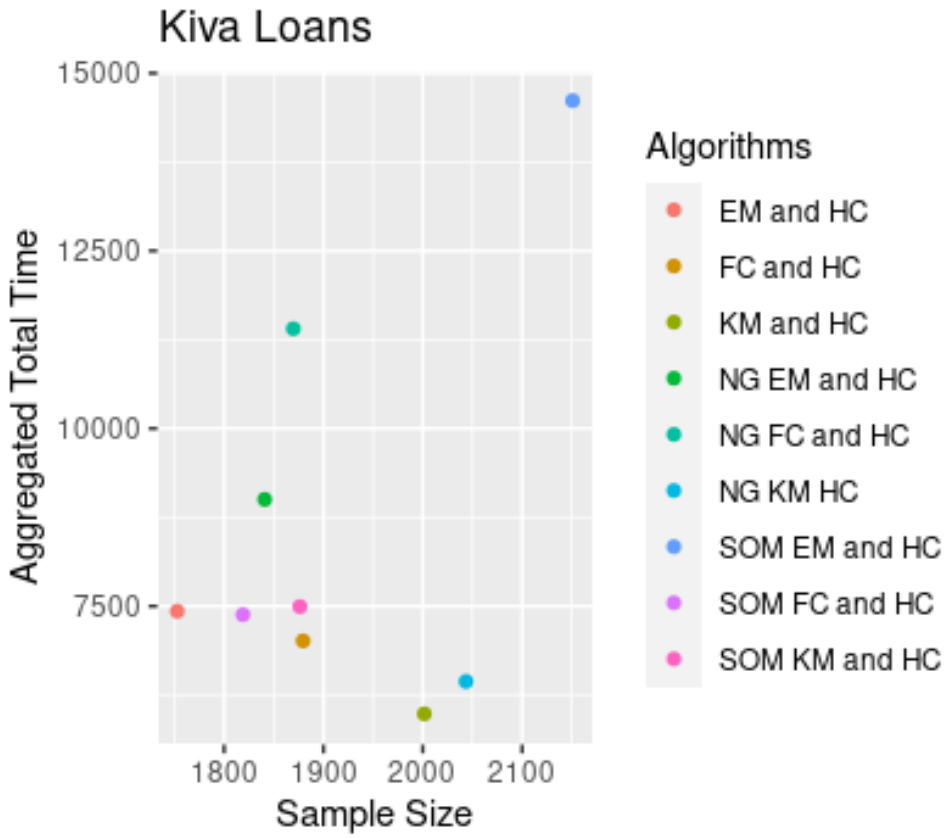


Figure 9.8: Plot of Aggregated Total Time versus Sample Size by Algorithm combination for the Kiva Loans dataset

In table 9.15.20 we provide details of the fine-tuned hyper-parameters for the relevant algorithms.

Table 9.15.20: *Summary of the fine-tuned hyperparameters for the various algorithm combinations for the American Community Survey, 2015 Data set*

SOM and K-means	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	371	0.277877372112223	0.012899151606656	0.694571047057514
NG and K-means	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	15.1897164925445	0.822356168348829	0.10500027609291	0.005030299354611
SOM and EM	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	914	0.162357069495344	0.040045055053234	0.535640629408619
NG and EM	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	20.9163182787887	0.804910189211671	0.183011407460269	0.076024879621262
Fuzzy Clustering	**m**			
	2			
SOM and FC	**Iterations**	**Alpha High**	**Alpha Low**	**Radius**
	264	0.113939729200586	0.00350977823364	0.003857035322069
NG and FC	**Lambda High**	**Lambda Low**	**Stepsize High**	**Stepsize Low**
	18.9297597394552	0.643763823658483	0.136589161984179	0.067314184908353

9.16 Conclusions

The multi-stage combination of clustering algorithms with hill-climbing performed better, generally, than the grouping genetic algorithm and the simulated annealing algorithm for atomic and continuous strata. The exception being the case of atomic strata for the Swiss Municipalities data set, where the grouping genetic algorithm performed better- although the survey designer may wish to consider solution quality in the context of training times (as with all the other experiments).

The continuous strata provide lower sample samples than the atomic strata method. This may, in part, be because the stratification variable is the target variable - which results in better efficiency.

The performance of each multi-stage combination of algorithms depends on the data set and the auxiliary and stratification variables. Therefore there is not one combination that suits all experiments. Nonetheless, we have provided the survey designer with a choice of combinations, which can be applied in the context expert knowledge of the particular sampling problem.

Chapter 10

A hybrid estimation of distribution algorithm

10.1 Introduction

In this chapter, we propose a hybrid estimation of distribution algorithm (HEDA) to solve this problem. EDAs are stochastic black-box optimization algorithms which can be used to estimate, build and sample probability models in the search for an optimal solution. We, therefore, describe the search for the lowest cost stratification as that of black-box optimization, in order to tie the problem in with existing EDA literature.

Rather than other methodologies where the strata have already been determined (e.g. administrative strata or the cumulative root frequency method) before evaluating the optimal sample allocation - the basic premise of this problem is that the optimal solution is unknown. It is a non linear problem with a rugged search space for which there are many near-optimal sample allocations (or local minima) and also perhaps more than one (i.e. attainable through multiple stratifications) optimal sample allocation (global minimum).

However, we cannot see the sample allocation for each stratification in advance, and thus cannot say *a priori* which stratification provides the optimal allocation. We could say that we are dealing with a black box. The only sure way of determining the optimal stratification is to evaluate each solution. This is known as grid-search and is computationally prohibitive for large problems. Indeed, for representative surveys, especially in official statistics, for any practical sized sampling frame the number of basic strata to be stratified (given that they may be derived from multiple auxiliary variables) can be quite large.

A black-box optimization procedure explores the search space by generating solutions, evaluating them, and processing the results of this evaluation in order to generate new promising solutions (Gonzalez-Fernandez and Soto, 2012). Such procedures tend to find local minima of varying proximity to a global minimum. However they tend to be faster alternatives to grid-search. Some are deterministic, e.g. the direct search approach of (Hooke and Jeeves, 1961) or the simplex method of (Nelder and Mead, 1965) and some stochastic, e.g. k-means (Hartigan and Wong, 1979), grouping genetic algorithm (John, 1975), simulated annealing (Černỳ, 1985; Kirkpatrick et al., 1983) and hill-climbing (Lin, 1965; Lin and Kernighan, 1973) . The latter set of procedures provide a means of attaining a solution that is "good enough" in a computing time that is "small enough" (Sörensen and Glover, 2013).

That has been our focus in earlier chapters in which we presented evolutionary (a grouping genetic algorithm -chapter 7) and local search (simulated annealing algorithm - chapter 8) algorithms and explored multi-stage combinations of clustering algorithms with a hill-climbing algorithm (chapter 9). We have evaluated the performance of these algorithms by the solution quality and computation time taken to find a local minimum. In chapters 8 and 9 we also considered training times. The performance has varied according to each data stratification problem.

To expand on the details mentioned above, we explore a new paradigm of evolutionary algorithms named estimation of distribution algorithms (EDAs) (Baluja, 1994; Larrañaga and Lozano, 2001; Mühlenbein and Paaß, 1996; Pelikan et al., 2003) in the context of this problem. In particular, we consider the application of a hybridised version of an EDA and compare it against the best results achieved so far with the algorithms discussed in chapters 7, 8 and 9. Section 10.2 provides some background details on EDAs. Section 10.3 motivates the use of a HEDA for this problem. Section 3.2 summarises the objective function.

Section 10.4 outlines the HEDA we propose in this chapter. Section 10.5 discusses the evaluation approach. Section 10.6 demonstrates how the EDA component of the HEDA works. Sections 10.7 and 10.8 give results of empirical comparisons of the HEDA for atomic and continuous strata. Section 10.9 and describes our conclusions and suggestions for further work are included in section 11.7.

10.2 Estimation of distribution algorithms (EDAs)

EDAs, which are also known as probabilistic model-building genetic algorithms (PMBGAs), belong to a family of evolutionary algorithms. They are characterized by iteratively estimating, building and sampling probability models in the search for optimal solutions.

To put it another way, EDAs work by selecting promising solutions from a population (similar to elitism in evolutionary algorithms) and building probabilistic models of those solutions. They then sample from the corresponding probability distributions to obtain new solutions (Lima et al., 2011). It is these characteristics of probabilistic model building and sampling from that model that most differentiates the EDA from other evolutionary algorithms such as the genetic algorithm.

If the model built in each generation captures the important features of selected solutions and generates new solutions with these features, then the EDA should be able to quickly converge to the optimum (Mühlenbein and Mahnig, 1999). Indeed, assuming that the population size is large enough to ensure reliable convergence (Goldberg, 2002; Harik et al., 1999), the EDA based on the probability model provides an efficient and reliable approach to solving many optimization problems (Hauschild and Pelikan, 2011).

Furthermore, as EDAs iteratively refine the probabilistic model using elite solutions, thereby increasing the probability of generating the global optimum, after a reasonable number of iterations, the procedure should locate the global optimum or its accurate approximation (Hauschild and Pelikan, 2011). Empirically, EDAs have been shown to outperform other optimisation techniques in problems such as the multi-objective knapsack problem (Shah and Reed, 2011), multi-objective monitoring network design (Kollat et al., 2008) or optimising cancer chemotherapy (Petrovski et al., 2006).

Basic EDAs use a simple probability model that has a fixed structure and learns the parameters for that model, e.g. the univariate marginal distribution algorithm (UMDA) (Mühlenbein and Paaß, 1996), population-based incremental learning (PBIL) (Baluja, 1994) and compact genetic algorithm (cGA) (Harik et al., 1999). Such EDAS are suitable for univariate problems.

On the other hand, for more complex problems there are EDAs with adaptive multivariate models such as Bayesian networks (BNs) (Pearl, 1988), which can model complex multivariate interactions. Examples of Bayesian EDAs include the Bayesian optimization algorithm (BOA) (Pelikan et al., 1999, 2003), the estimation

of Bayesian networks algorithm (EBNA) Etxeberria (1999), and the learning factorized distribution algorithm (LFDA)(Mühlenbein and Mahnig, 1999).

For this problem, as we are dealing with mutually exclusive basic strata, we assume that the probability distribution of any basic stratum belonging to a particular stratum is independent of that for another basic stratum, and for this reason we will implement a univariate EDA based closely on the UMDA.

10.3 A hybrid estimation of distribution algorithm (HEDA)

Even though we pointed out in section 10.2 that EDAs should locate the global optimum or its near approximation in a reasonable amount of iterations, they have two main computational bottlenecks: (1) fitness evaluation and (2) model building. These must be addressed by efficiency enhancement techniques suitable for EDAs (Hauschild and Pelikan, 2011).

The latter bottleneck we have addressed by using the simple model building approach of the UMDA - thus avoiding the computational overhead of more complex model building. The former bottleneck, clearly, is not only an issue for the EDA as it relates not only to the size of the search space but also the complexity of the problem. However, we can address this using hybridisation (Hauschild and Pelikan, 2011). This is an efficiency technique the purpose of which is to speed up the performance of the EDA.

In many real-world applications, evolutionary algorithms are combined with other search optimization algorithms such as local search, tabu search, simulated annealing or hill-climbing algorithms. Such metaheuristic hybrids combine the advantages of population based exploration methods (which search regions of the search space) with trajectory methods (which optimise locally and are often very successful). Simulated annealing is one example of a trajectory method which we focus on in this chapter.

Simulated annealing resembles hill-climbing in that it makes small adjustments to a candidate solution - keeping those which lead to an improvement in solution quality and gradually morphing the solution towards the closest local optimum. Hill-climbing algorithms generally attain good results especially when combined with population based search procedures (Lima et al., 2009).

As an example, to solve the minimal switching graph problem (Tang and

Lau, 2005) combined the exploration properties of the UMDA and the exploitation properties of the hill-climbing algorithm into a hybrid EDA to find an optimal or near-optimal solution efficiently and effectively. They compared the hybrid algorithm with the UMDA and the hill-climbing algorithm separately, and found that the performance of the hybrid EDA is significantly better than both the UMDA and the hill-climbing algorithm. Similarly, a hybrid EDA with simulated annealing was implemented on a production scheduling problem by Gao et al. (2020) in order to enhance the algorithm's search ability.

For our problem, we expect a hybrid EDA using simulated annealing to perform well. This is because hill-climbing solutions gravitate towards the nearest local minima, where they become trapped until the algorithm reaches a pre-defined end-point. On the other hand, simulated annealing allows for a probabilistic acceptance of inferior quality solutions, enabling escape from local minima and thus attaining better quality solutions than hill-climbing. For the reasons outlined in this section and in section 10.2, combining an EDA with simulated annealing should generate less solutions and thus translate into calling the Bethel-Chromy evaluation algorithm less often to attain optimal or near-optimal solutions.

10.4 Outline of the HEDA

We base our description of the hyrbid estimation of distribution aglorithm on that found in Gonzalez-Fernandez and Soto (2012). We start with a population of N_p solutions either generated at random, or a starting solution accompanied by solutions generated with random permutations that solution. The population is evaluated and the best solutions are selected. Any selection method can be used (e.g. roulette wheel, tournament selection, Boltzmann selection, etc.) however for our algorithm we use the elitism approach described in chapter 7.

We construct a probability model of the best solutions. We then use that model to generate new solutions to replace those not selected at the elitism stage. After a tunable number of iterations a simulated annealing algorithm (for the purpose of approximating a local optimum) is applied to a tunable number of solutions from the population (for the experiments in sections 10.7 and 10.8 we apply simulated annealing to the top ranked solution). This process continues until a stopping rule has been reached.

Algorithm 13 Hybrid EDA

1: Set $i \leftarrow 1$ and generate initial population P_1 of N_p solutions
2: Apply simulated annealing algorithm ▷ tunable conditions
3: Select N_{Elite} promising solutions S_i from P_i
4: Construct a probability model M_i of S_i (see section 10.6 below).
5: Create a new population P_{i+1} by sampling a set of $N_p - N_{Elite}$ new solutions according to the distribution encoded by M_i
6: set $i \leftarrow i + 1$
7: If stopping criteria (number of iterations, I) not met repeat steps 2 to 6

10.5 Evaluation approach

Each of the N_p solutions in the population are evaluated in a two step-approach using the *aggrStrata* function (Ballin and Barcaroli, 2020) and *Bethel* algorithm (Ballin and Barcaroli, 2020; De Meo and De Meo, 2009). The *aggrStrata* function obtains summary statistics (including point estimates) for the basic strata. The *Bethel* function uses those statistics to compute the optimal sample allocation for that stratification.

We found this two-step approach to be a more useful evaluation metric for the HEDA than the Bayesian Information Criterion (BIC) or total within sums of squares (TWSS) which are more suited to clustering problems and faster than the evaluation algorithm. We initially considered these as "surrogate" functions. However, solutions which attain better BIC or TWSS scores might not attain better sample allocations. This is because minimising TWSS eventually leads towards one cluster per atomic stratum whereas maximising BIC scores is useful for selecting the optimal number of clusters, but not necessarily the optimal arrangement of items within clusters.

10.6 Example of EDA component of the HEDA on the iris data set

To demonstrate how an estimation of distribution algorithm works we use the iris data set (Anderson, 1935; Fisher, 1936; R Core Team, 2021) and atomic strata. We select Petal Length and Petal Width as the target variables. We use Sepal Length and Species as auxiliary variables. We convert Sepal Length to a categorical variable with 3 bins using the k-means algorithm (Hartigan and Wong, 1979). We use an upper coefficient of variation level of 0.05 for the target variables. The Cartesian product of the categorical version of Sepal Length with Species results in 8 atomic

strata. We group these 8 atomic strata into H strata, in this example $H = 3$ or $H = 4$, i.e. some of the solutions have three strata and some have four.

Table 10.6.1: *Summary statistics for the L atomic strata*

Atomic Stratum, l	N	M1	M2	S1	S2
a	40	1.46	0.24	0.17	0.10
b	5	3.40	1.10	0.30	0.15
c	1	4.50	1.70	0.00	0.00
d	10	1.48	0.29	0.17	0.09
e	31	4.23	1.31	0.36	0.19
f	12	5.07	1.88	0.22	0.27
g	14	4.64	1.45	0.21	0.11
h	37	5.74	2.08	0.50	0.25

We initialise a population size of N_p solutions (in this case $N_p = 5$) each of size L, i.e. the number of atomic strata, where $l = 1, 2, \ldots, L$. As can be seen in table 10.6.1 we initially use letters of the alphabet to differentiate between atomic strata, however in subsequent tables an integer denotes to which one of the H strata each atomic stratum belongs. Each solution is evaluated and ranked.

Table 10.6.2: *Population of size N_p with solution quality and rank*

Initial Population								Quality	Rank
3	2	2	3	2	1	2	2	19.19	3
3	2	4	3	2	1	2	3	46.77	5
3	2	4	1	2	1	2	4	14.48	2
3	2	4	3	2	1	1	4	10.65	1
3	3	2	3	2	1	2	2	30.30	4

We create a new selected population of the best solutions (in this example we select 2 solutions) in a process that is analgous with elitism in evolutionary algorithms.

Table 10.6.3: *Selected population of best elite solutions*

Selected Population							
3	2	4	3	2	1	1	4
3	2	4	1	2	1	2	4

We then construct a probabilistic model of the solutions in the selected population with the aim of estimating the probability distribution for each of the H strata for each atomic stratum. The probability of a candidate solution (or vector, V) where $(V = V_1, V_2, ..., V_L)$ is the product of probabilities of individual strata for each atomic stratum:

$$p(V_1, V_2, ..., V_L) = p(V_1) \times p(V_2), ..., \times p(V_L) \tag{10.1}$$

where $p(V_l)$ is the probability of variable $(V_l = v_l)$, v_l is an integer between 1 and H and $p(V_1, V_2, ..., V_L)$ is the probability of the candidate solution

$(V_1, V_2, ..., V_L)$ and H represents the number of strata which are labelled individually by an integer between 1 and H. The univariate model for the L variables consists of L vectors containing probabilities of an atomic stratum belonging to the different strata. The probabilities different strata sum to 1.

The joint probability distribution is factorized as a product of independent univariate marginal distributions, which are estimated from marginal frequencies (Paul and Iba, 2002):

$$p(V_l) = \frac{\sum_{j=1}^{N_{Elite}} \delta_j(V_l = v_l | N_{Elite}^{iter-1})}{N_{Elite}^{iter-1}} \tag{10.2}$$

where $N_{Elite}^{iter-1}$is the number of elite solutions from the previous iteration, $\delta_j(V_l = v_l | N_{Elite}^{iter-1}) = 1$ if $V_l = v_l$ in the j^{th}case of N_{Elite}^{iter-1}, and 0 otherwise.

Table 10.6.4: *Model estimating the probability of each atomic stratum l belonging to each stratum h*

Stratum, h	**Probability Model**							
1	0	0	0	0.5	0	1	0.5	0
2	0	1	0	0	1	0	0.5	0
3	1	0	0	0.5	0	0	0	0
4	0	0	1	0	0	0	0	1
Total	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

We generate new solutions from this model to replace the non-elite solutions by sampling from that distribution. We evaluate the new solutions and rank all solutions by their quality. This offspring population returns a solution quality of 9.34 which is the global minimum (to find the global minimum we evaluated each of the 4,140 possible partitions of the 8 atomic strata).

Table 10.6.5: *Offspring population with solution quality and rank*

Offspring Population								**Quality**	**Rank**
3	2	4	3	2	1	1	4	10.65	2
3	2	4	1	2	1	2	4	14.48	3
3	2	4	3	2	1	1	4	10.65	2
3	2	4	3	2	1	2	4	9.34	1
3	2	4	1	2	1	2	4	14.48	3

Normally the algorithm repeats the above process until a stopping criterion has been met.

The extreme probabilities — 0 and 1 — mean that unless some mitigating measure is introduced the atomic stratum l would remain fixed forever to stratum h at a probability of either 0 or 1, obstructing some regions of the search space (Dang et al., 2019). However, at advanced stages of the search, e.g. when using a starting

solution, the stratifications are of good quality and atomic stratum may already be in the correct stratum (i.e. the stratum it would be allocated to in the optimal solution).

In the above experiment it is clear that for all solutions in the population there are a number of cases where there is only one stratum choice available for an atomic stratum (i.e. it is assigned to that stratum with a probability of 1). Here we are very close to a global minimum and we get there on the second step - without adjusting the probability of strata assignments. On a more general level however, the hybrid EDA uses mutation and add strata components (see chapter 7), as well as simulated annealing (see chapter 8) which enable escape from this occurrence. The UMDA also has alternative approaches of avoiding $1,0$ probabilities (such as capping the probability interval to between $\frac{1}{L}$ and $(1-\frac{1}{L})$) e.g. (Doerr and Krejca, 2020), however, for our purposes they are not required.

10.7 Empirical comparisons for atomic strata

10.7.1 Background details

We compare the performance of the HEDA with the best solution qualities from the experiments we carried out on the same data sets, with the grouping genetic algorithm, simulated annealing algorithm and combinations of clustering algorithms with hill-climbing. We use the best results (solution qualities with execution times and total execution times) for all algorithms from these experiments as a benchmark for comparing the HEDA. More details on these experiments are provided in chapters 7, 8 and 9.

The data sets including target and auxiliary variables are summarised in table 10.7.6. We explain the meaning of each variable in table B.1.1 in section B.1.

Table 10.7.6: *Summary by data set of the variables, number of records and atomic strata*

Data set	Target Variables	Auxiliary Variables	Number of Records	Number of Atomic Strata
Swiss Municipalities	Surfacesbois, Airbat	POPTOT , HApoly	2,896	579
American Community Survey, 2015	HINCP, VALP, SMOCP, INSP	BLD, TEN, WKEXREL, WORKSTAT, HFL, YBL	619,747	123,007
Kiva Loans	terminmonths, lendercount, loanamount	sector, currency, activity, region, partnerid	614,361	84,897

Table 10.7.7 provides details of the best sample sizes and the corresponding algorithms from the aforementioned which they were attained with.

Table 10.7.7: *Summary by data set of the best sample size with evaluation time for the final stage in multi stage combinations of clustering algorithms with the simulated annealing algorithm (table 8.5.8) and hill-climbing algorithm (tables 9.14.7 and 9.14.10)*

Data set	Initial Stage(s)	Final Stage	Sample Size	Execution Time	Total Execution Time
Swiss Municipalities	Kmeans	SAA	125.17	248.91	8,808.63
American Community Survey, 2015	SOM, FC	HC	9,065.45	6,298.81	6,298.81
Kiva Loans	FC	HC	6,326.35	6,018.61	6,018.61

However, as we have sought to develop algorithms that provide solutions that are good enough in a computing time that is small enough we also report evaluation (execution) and training (total execution) times to provide context to the results. In table 10.7.7, *SAA* refers to simulated annealing algorithm, *SOM* applies to self-organising map, *FC* relates to Fuzzy Clustering and *HC* indicates hill-climbing.

For comparison purposes execution time and total execution time relates to the final stage algorithms only. Note: there was no fine-tuning required for the hill-climbing algorithm. Details of the fine-tuning time for the initial stage solutions are available in chapters 8 and 9.

10.7.2 Hyperparameters

The hyperparameters used in the experiments for the HEDA are provided in Table 10.7.8.

Table 10.7.8: *Hyperparameters for the hybrid estimation of distribution algorithm (HEDA)*

Data set	Iterations, I	SAA Frequency	Mutation Chance	Elitism Rate	Add Strata Factor	Temperature, T
Swiss Municipalities	3,000	100	0.000185221	0.20	0.000000146	0.000000027
American Community Survey, 2015	200	10	0.000000016	0.30	0.000000054	0.000655857
Kiva Loans	600	100	0.000038838	0.30	0.000000739	0.001400962
Data set	**Decrement Constant,DC**	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**
Swiss Municipalities	0.941411680	2	20,000	0.10	0.009183601	20
American Community Survey, 2015	0.815042754	20	5,000	0.01	0.000000609	20
Kiva Loans	0.954218273	2	10,000	0.02	0.000052373	20

The hyperparameters operate in this way, *Iterations, I* indicates the number of iterations the algorithm runs for. The algorithm stops at the last iteration. The HEDA builds and samples from a *probability model, M_i* once every iteration. The simulated annealing algorithm also runs after a tunable number of iterations, i.e. for the Swiss Municipalities and Kiva Loans experiments it runs once every 100 iterations whereas

for the American Community Survey experiment it runs once every 10 iterations. The simulated annealing algorithm runs each time for a maximum number of sequences with the length of each sequence varying for each experiment. See chapter 8 for more details on the hyperparameters for the simulated annealing algorithm.

The following hyperparameters are described in more detail in chapter 7. However in brief, *Elitism rate* indicates the proportion of the ranked population that is carried over to the next iteration, *Mutation chance* indicates the probability of each atomic stratum being moved to another stratum every iteration. Similarly, *Add Strata Factor* is the probability of an atomic stratum moving to a new stratum. *Population size,* N_p is the number of solutions used for exploration and evolution each iteration.

The hyperparameters often need to be fine-tuned to suit the characteristics of a particular data set. We fine-tuned the hyperparameters for these experiments using the methodology described by Bischla et al. (2017) in the R software language (R Core Team, 2021) and for more details on the methodology see section 6.1. Finally, although we have stated the hyperparameters, the solution quality attained with them in the HEDA may not be reproduced exactly on their first application. See table B.2.2 for details of the hyperaparameters, execution and total execution times in the training process.

10.7.3 Results

The results of these experiments are provided in Table 10.7.9.

Table 10.7.9: *Summary by data set of the benchmark sample size and evaluation time for the multi stage combinations of clustering algorithms with the HEDA*

Data set	Initial Stage(s)	Final Stage	Sample Size	Execution Time	Total Execution Time
Swiss Municipalities	Kmeans	HEDA	122.39	1,316.11	47,045.87
American Community Survey, 2015	SOM and FC	HEDA	8,800.53	28,857.52	550,676.58
Kiva Loans	Kmeans	HEDA	6,246.74	3,292.97	53,006.37

We compare the sample sizes attained in table 10.7.7 and table 10.7.9 and present the results below in table 10.7.10.

Table 10.7.10: *Ratio comparison of the sample sizes for the the HEDA results and the benchmark results*

Data set	Benchmark results	HEDA Results	Ratio
Swiss Municipalities	125.17	122.39	0.98
American Community Survey, 2015	9,065.45	8,800.53	0.97
Kiva Loans	6,326.35	6,246.74	0.99

The ratio of the HEDA results to the benchmark results indicate that, although they are similar, the HEDA results are of better quality in all cases. However,

as our focus has been on obtaining results that are good enough in a time that is small enough these results should be considered in the context of execution and total execution times.

Table 10.7.11 compares the execution times for the HEDA and those for the final stage algorithm in the benchmark results.

Table 10.7.11: *Comparison of the execution times and total execution times for the benchmark results and the HEDA results*

	Benchmark Results	**HEDA Results**		**Benchmark Results**	**HEDA Results**	
Data set	**Execution Time**	**Execution Time**	**Ratio**	**Total Execution Time**	**Total Execution Time**	**Ratio**
Swiss Municipalities	248.91	1,316.11	5.29	8,808.63	47,045.87	5.34
American Community Survey, 2015	6,298.81	28,857.52	4.58	6,298.81	550,676.58	87.43
Kiva Loans	6,018.61	3,292.97	0.55	6,018.61	53,006.37	8.81

In brief the results indicate that overall, for these experiments, the HEDA takes longer to execute and fine-tune than the SAA for the Swiss Municipalities, or the combination of clustering algorithms with hill-climbing for the American Community Survey, 2015 and Kiva Loans experiments.

10.8 Empirical comparisons for continuous strata

Using the same methodology applied in section 10.7 we ran experiments using the same data sets comparing the performance of the HEDA on the stratification of continuous strata with the benchmark results from experiments we described in chapters 7, 8 and 9. As before we first provide a summary of the target and auxiliary variables in table 10.8.12.

Table 10.8.12: *Target and auxiliary variables for the Continuous method*

Data set	**Target variables Auxiliary variables**			
Swiss Municipalities	Airbat	Surfacesbois		
American Community Survey, 2015	HINCP	VALP	SMOCP	INSP
US Census, 2000	HHINCOME			
Kiva Loans	term_in_months	lender_count	loan_amount	
UN Commodity Trade Statistics data	trade_usd			

Likewise table 10.8.13 provides details of the best sample sizes and the corresponding algorithms from the aforementioned which they were attained with.

Table 10.8.13: *Summary by data set of the sample size and evaluation time for the final stage of multi stage combinations of clustering algorithms with the hill-climbing algorithm*

Data set	Initial Stage(s)	Final Stage	Sample Size	Execution Time	Total Execution Time
Swiss Municipalities	NG and EM	HC	110	38.61	38.61
American Community Survey, 2015	NG and FC	HC	2,753.55	28,632.09	28,632.09
Kiva Loans	EM	HC	1,752.82	7,370.53	7,370.53

10.8.1 Hyperparameters

The hyperparameters used in the HEDA are provided below in Table 10.8.14.

Table 10.8.14: *Hyperparameters for the hybrid estimation of distribution algorithm (HEDA)*

Data set	Iterations, I	SAA Frequency	Mutation Chance	Elitism Rate	Add Strata Factor	Temperature, T
Swiss Municipalities	1,000	100	0.0000008	0.2	0.0000000	0.0000058
American Community Survey, 2015	30	10	0.0000003	0.3	0.0000010	0.0000010
Kiva Loans	90	10	0.0000004	0.3	0.0000001	0.0000001
Data set	**Decrement Constant, DC**	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**
Swiss Municipalities	0.6266293	10	3,000	0.0195154	0.0000377	20
American Community Survey, 2015	0.9849021	2	50,000	0.0113557	0.0000001	20
Kiva Loans	0.9844605	2	20,000	0.0156086	0.0000008	20

Table B.3.3 outlines the hyperaparameters used, along with the execution and total execution times resulting from the training process.

10.8.2 Results

The results of the HEDA experiments on continuous strata are provided in Table 10.8.15.

Table 10.8.15: *Summary by data set of the sample size and evaluation time for the final stage in multi stage combinations of clustering algorithms with the HEDA*

Data set	Initial Stage(s)	Final Stage	Sample Size	Execution Time	Total Execution Time
Swiss Municipalities	Kmeans	HEDA	104.93	207.11	6,792.40
American Community Survey, 2015	NG and FC	HEDA	2,619.06	22,663.52	247,314.63
Kiva Loans	NG and Kmeans	HEDA	1,576.80	8,791.12	88,951.97

As before, we compare the sample sizes attained in table 10.8.13 and table 10.8.15 and present the results below in table 10.8.16.

Table 10.8.16: *Ratio comparison of the sample sizes for the HEDA results and the benchmark results*

Data set	Benchmark Results	HEDA Results	Ratio
Swiss Municipalities	110	104.930419822273	0.95
American Community Survey, 2015	2,753.55	2619.06326917666	0.95
Kiva Loans	1,752.82	1,576.80	0.90

The ratio of the HEDA results to the benchmark results indicate that, the HEDA results are of better quality in all three experiments.

Table 10.8.17 indicates that the execution times and total execution times are, overall, significantly greater for the HEDA.

Table 10.8.17: *Ratio comparison of the execution times and total execution times for the HEDA results and the benchmark results*

	Benchmark Results	HEDA Results		Benchmark Results	HEDA Results	
Data set	**Execution Time**	**Execution Time**	**Ratio**	**Total Execution Time**	**Total Execution Time**	**Ratio**
Swiss Municipalities	38.61	207.11	5.36	95.33	6,792.40	71.25
American Community Survey, 2015	28,632.09	22,663.52	0.79	41,740.96	247,314.63	5.92
Kiva Loans	7,370.53	8,791.12	1.19	7,428.64	88,951.97	11.97

10.9 Conclusions

Although these experiments are intended to be demonstrative rather than exhaustive, the multi-stage combination of an initial solution with the HEDA generally attains better sample sizes than the combinations of initial solutions with either the SAA or HC algorithm which had attained the best results in earlier experiments (that also included the GGA).

The execution times and total execution are, generally, significantly higher for the HEDA than some of the best alternative performing algorithms (especially the clustering algorithms considered in chapter 9). However, it is worth recalling that the HC algorithm in the above experiments had converged on a solution and would not have attained better results without adjusting the stopping conditions for the algorithm or adjusting the solution. We should also bear in mind that the HEDA has found the best results so far for these experiments, and the execution and total times for the benchmark algorithms to find solutions of that quality (or better), using either the same or alternative starting solution, have not been evaluated.

That said, the HEDA is useful for a survey designer with expert knowledge of the sampling frame and the stratification of basic strata (atomic and continuous)- for whom the requirement to fine-tune hyperparameters is mitigated by prior knowledge.

It is also useful for survey designers for whom the fine-tuning times can be included in their computation budget.

Nonetheless, because of the combination of probability models, explorative and exploitative search we feel the HEDA is well placed to generally outperform when it comes to attaining results which are good enough in a time that is small enough.

Chapter 11

Conclusions and further work

11.1 Conclusions

In this thesis, we consider the problem of joint stratification and sample allocation in survey design. There are two main difficulties associated with this problem. The first being that as the problem grows in size the search space of candidate solutions becomes too great for each solution to be evaluated and thus determine the optimal solution. The second being that the evaluation time for each solution also increases in size with the complexity of the problem.

In the development of the above techniques we considered faster surrogate objective functions, such as the Bayesian Information Criterion (BIC) or total within sums of squares (TWSS). However, minimising TWSS eventually leads towards one cluster per atomic stratum whereas maximising BIC scores is useful for selecting the optimal number of clusters, but not necessarily the optimal arrangement of items within clusters. That said, there may be scope to consider alternative convex approximation algorithms as a substitute for the evaluation algorithm.

However, we opted to use the existing Bethel-Chromy algorithm and focus on proposing a number of metaheuristic and machine learning algorithms both to create starting solutions and to find solutions that are good enough in a computing time that is small enough.

Our first algorithm the grouping genetic algorithm (GGA) compares favourably with the classical GA at finding the correct solution (for a toy example using the iris data set) and on finding similar quality solutions on smaller data sets, but significantly outperforms the GA on larger data sets - in cases where the number of iterations was restricted.

We compared the next algorithm we developed, the simulated annealing algorithm (SAA) with the GGA in the case of atomic strata and the classical GA in the case of continuous strata. In both cases we used the k-means algorithm to generate a starting solution before training hyperparameters for which both algorithms attained results of similar quality. However, the algorithm execution times and training times for the recommended hyperparameters were lower for the SAA (which benefited from delta evaluation) than for the GGA.

After this we combined the k-means and clustering algorithms with a hill-climbing algorithm to search for solutions. The combination performed better, generally speaking, than the grouping genetic algorithm and the simulated annealing algorithm for atomic and continuous strata. While there isn't one combination that suits all experiments we have provided the survey designer with a choice of combinations to draw from when using this approach.

We conclude our research with the hybrid estimation of distribution algorithm (HEDA). The HEDA combines the capability of the EDA to model the probability distribution of an atomic stratum belonging to different strata with the exploitative search properties of a simulated annealing algorithm. Comparisons with the best solution qualities from the experiments in earlier chapters show that the HEDA finds better solution qualities but requires longer total execution times than the alternative methods we described.

11.2 Unseen data sets

We drew from the same selection of data sets for each of the above experiments. These experiments demonstrate that each algorithm finds better solutions than the initial solutions. The use of model-based optimisation to train the hyperparameters in experiments where the algorithms are unlikely to find the optimal solution in a small number of trials, is more likely to lead to better solutions, than a simple trial and error approach, or a one-off selection. Therefore, while there are added computation costs involved in the training step, it ultimately could save time and return a better quality solution than would other wise be the case, thus resulting in cost savings both in terms of computation time and a smaller sample size.

As the algorithms have been implemented to solve the problem, they can, therefore, be used on unseen data sets as well. New data sets could be generated from the existing data sets by randomly perturbing the target and auxiliary variables. Further testing data sets can also be retrieved from open source data hubs such

as IPUMS or Kaggle which are used in this thesis. Alternatively, there is the option of randomly generating datasets, using knowledge of existing or experimental distributions.

11.3 Hyperparameters

Hyperparameters are integral to the computational efficiency of the clustering algorithms and, more particularly, the GGA, SAA and HEDA, as well as their capacity to find near optimal or optimal solutions. The choice of hyperparameters is specific to each algorithm, and each experiment. We don't know which set of hyperparameters to use from the outset, and their selection is also an optimisation problem.

The advantage of using model based optimisation over trial and error fine-tuning, is that the optimisation has been shown, empirically, to progress sequentially towards the optimal set of hyperparameters.

However, we only randomly generate 10 sets of hyperparameters from within the ranges specified. From these, 10 tests are carried out, and the results of these tests are used as a starting point to sequentially fine-tune the hyperparameters in 10 further tests. This choice of 20 tests is arbitrary and due to practical consideration of the computation times for each of the tests.

Similarly, the selection of ranges for the hyperparameters is based on knowledge of each algorithm and the data. However, it may happen that different ranges of hyperparameters may prove more effective. Therefore, the hyperparameters for each of the above experiments have been fine-tuned, but are more than likely not optimal. Clearly, the survey designer may be starting from a clear slate when deciding which hyperparameters to use, in order to solve a given instance of this problem. Nonetheless, we have presented a framework for training in either the above or other hyperparameterised algorithms.

11.4 Number of Strata

As mentioned in chapter 3, the optimal stratification will have H strata. However, the size of H can vary in cases where there are more than one optimal solutions. For example, as we saw in chapter 7 for the iris data set experiment, there are three optimal solutions with 3, 4, or 5 strata. On the other hand, there may be more than one optimal solution all with H being the same size, but where the arrangement

of basic strata within varies. We cannot tell in advance the number of strata that optimal solution(s) will have, unless we have completed a grid search of all possible solutions.

We need to allow for this when generating starting solutions, either randomly or with a clustering algorithm. The Elbow method or Gap statistic method may be useful in determining a good starting solution (and thus a starting number of strata) from the clustering algorithms. However, due to the movement of basic strata between strata, the final number of strata in the optimal solution may differ from the number in the starting solution.

11.5 Parallelisation

For the experiments in chapters 7, 8 and 9 we used parallelisation. As noted in chapter 8, in some cases there are more domains than cores. In these cases, the algorithms loop through the block of available cores until all domains have been searched. In cases where the domains vary greatly in size, this means waiting until the algorithm has finished searching the largest domain, before moving on to the next set of domains.

However, recall that each domain is independent. Prior to parallelisation, we propose splitting the domains in to approximately equal sized sets (where a set is less than or equal in size to the number of available cores), or else sorting them by size in descending order. In this way, we expect that the parallelisation step the variation between computation times for a set of domains in the available cores will be minimised. Therefore, it will run with less 'idle-time' and thus achieve gains in total computation times. For more information on implementing this type of parallelisation in *R*, refer to the *parallel* package (R Core Team, 2021).

11.6 Recommendations

As discussed in the relevant chapters, in principle, the GGA, SAA and HEDA should all find an optimal solution. However, the hyperparameters may need to be trained, in order to do so. The hill climbing algorithm has no hyperparameters to be trained and will converge on a locally optimal solution, but this may not be the globally optimal solution.

The clustering algorithms, should find the optimal solution on small problems, like the iris data set experiment described in chapter 7. However, they are likely to

be more useful in finding good quality starting solutions in larger experiments. This is because of their speed and they seek to minimise a different objective function.

The combination of the initial solutions found by the clustering algorithms with the hill climbing algorithm will be the best choice in cases where the survey designer is satisfied with local minima solutions that are of varying proximities to the global minima. However, for cases where the survey designer is satisfied to train the hyperparameters of the final stage algorithms (i.e. GGA, SAA or HEDA), we expect that the HEDA will more efficiently find better quality solutions.

This is because of its hybrid nature combining a probabilistic model-building evolutionary algorithm with simulated annealing and delta evaluation. Our results indicate that it efficiently explores and exploits good solutions in the search space, thus calling the Bethel-Chromy algorithm less often to find better quality solutions. It is also because of the optimisation of the HEDA and evaluation function code, described in section 10.5.

Nonetheless, we are also confident in the three final stage algorithms as part of a multi stage combination with initial solutions generated by the clustering algorithms, as an implementation of the GGA is already publicly in use, and they all perform well. Indeed, the net gain from this research is a greater range of algorithms, which exploit various computational efficiency methods, to choose from. This is along with a multi-stage approach to efficiently solve this statistical problem and a recommendation to fine tune the hyperparameters so that ultimately the search time for near-optimal solutions can be shortened.

11.7 Further work

As pointed out in chapter 3, we have set $C_h = 1$, and thus we have focused on solution qualities measured by sample size rather than sample cost, i.e., where $C_h > 1$. Although we don't know H in advance, we do know the domain that each h would be in, and therefore we can estimate sample cost, where C_h is the average cost of surveying a unit in a given domain. Therefore, the algorithms can be used to search for better qualities solutions where quality is a function of sample cost rather than size. This measurement of sample cost merely entails a manipulation of the cost vector in the Bethel Chromy algorithm, however, as we haven't reported results on this application of the GGA, SAA and HEDA, we include it as further work.

All three algorithms can be applied in an alternative stratification problem, where instead of minimising sample size, there is an upper bound on n, and the

aim is to minimise the CV values. In these cases the algorithms behave in the same way, searching for the correct arrangement of basic strata within strata, but a different objective function (using the specified criteria) would measure solution quality.

Similarly, all three algorithms can be applied to consider cases where the response rates are less than 100%. In these situations, the solution quality is sample size adjusted for non response, in the manner indicated in chapter 1.

The GGA can be developed in several ways. Alternative evaluation techniques to speed up the algorithm could be considered. Further research could also be undertaken into other machine learning techniques for solving this problem.

The GGA could be applied to other problems which tackle more general sampling designs with modifications required only for the algorithm evaluating the fitness of chromosomes (i.e., the Bethel-Chromy algorithm). In other words, the basic concept of grouping items in a manner which best minimises (or maximise) the objective function, remains. Instead, a different evaluation algorithm would be used in accordance with requirements of the grouping problem.

For example, instead of searching for a stratified simple random sample to meet precision constraints based on population totals, the GGA could consider stratified probability proportional to size sampling with an evaluation algorithm that uses more general estimators (e.g., regression or ratio estimators) or more general parameters (e.g., a correlation coefficient). In such circumstances, solution quality would be a function of the regression or ratio estimators, or the correlation coefficient.

The evaluation algorithm might also be modified to look at scenarios in which the population variances are not known. In these cases, data from previous censuses, administrative records, or proxy surveys can be used to estimate the population variance. However, estimation of the population variance in a large number of atomic strata requires more careful research.

Finally, the groupings of atomic strata by the GGA can be difficult to interpret. For instance, an ordinal auxiliary variable taking values 1 to 4 may be unnaturally separated, where the atomic strata corresponding to values 1 and 3 are grouped in one design stratum and those with values 2 and 4 are grouped in another design stratum. It might be interesting to explore less-than-optimal sample sizes for stratifications that are easier to interpret. For instance, one may impose constraints on the admissible groupings. This would require research into the formulation of appropriate admissibility constraints and their effective implementation in the GGA.

The perturbation used by the SAA randomly moves q atomic strata, where

mainly $q = 1$, from one stratum to another. This stochastic process is standard in default simulated annealing algorithms. However, as we are using a starting solution where there is already similarity within the strata, this random process could easily move an atomic stratum ($q = 1$) to a stratum where it is less suited than the stratum it was in. This suggests that a certain amount of redundancy in the search for the global minimum.

Lisic et al. (2018) conjecture that the introduction of nonuniform weighting in atomic strata selection could greatly improve performance of (their proposed) simulated annealing method by exchanging atomic strata near stratum boundaries more frequently than more important atomic strata. We agree that for this algorithm it would be more beneficial if there was a higher probability that an atomic stratum that was dissimilar to the other atomic strata was selected. We could then search for a more suitable stratum to move this atomic strata to.

To achieve this we could first randomly select a stratum, and then measure the Euclidean distance of each atomic stratum from that stratum medoid, weighting the chance of selection of the atomic strata in accordance with their distance from the medoid. At this point an atomic stratum is selected using these weighted probabilities.

The next step would be to use a K-nearest-neighbour algorithm to find the stratum medoid closest to that atomic stratum and move it to that stratum. This simple machine learning algorithm uses distance measures to classify objects based on their K nearest neighbours. In this case, $k = 1$, so the algorithm in practice is a closest nearest neighbour classifier.

This additional degree of complexity to the algorithm offsets the gains achieved by using delta evaluation, particularly as the problem grows in size, thus reducing the number of solutions evaluated in the same running time. It might be effective to use the column medians as an equivalent to the medoids. This could better assist the algorithm find better quality solutions. However, this may only be effective at an advanced stage of the search where the l in H are already quite similar.

In the case of our multi-stage combinations of k-means and clustering algorithms with the hill-climbing algorithm further combinations of the hill-climbing algorithm with other clustering algorithms such as affinity propagation clustering, cobweb clustering algorithm, density-based clustering, divisive hierarchical clustering, simple featureless clustering, farthest first clustering algorithm, mean shift clustering, kernel k-means clustering, mini batch k-means clustering, k-means with automatic determination of k, partitioning around medoids, k-medians clustering, etc. may be

considered. Alternative local search algorithms such as tabu search, stochastic hill-climbing, local beam search or simulated annealing may also be considered for the final stage. The grouping genetic algorithm may also be applied.

Finally in the case of the HEDA adjustments can be made in order to use a seed function for reproducibility of results. Optimised versions of the GGA, SAA and basic EDA can also be implemented in *Rcpp*. It might be useful to extend the functionality of the HEDA to the stratification approach used for spatial sampling by Ballin and Barcaroli (2020). Furthermore, as we kept the population size, and the number of solutions to which we applied simulated annealing to every iteration, the same for all experiments, there is scope to fine those hyperparameters. Lastly, we could investigate more complex techniques such as the Bayesian optimisation algorithm, the estimation of Bayesian networks algorithm, the learning factorized distribution algorithm or mutual information maximising input clustering (MIMIC) (De Bonet et al., 1997) on this problem.

Appendix A

A.1 Background details on the comparisons of the SAA with the GGA and CGA

A.1.1 Precision constraints

The target upper precision levels for these experiments, i.e. coefficients of variation, for each of the five experiments are provided in table A.1.1 below.

Table A.1.1: *Summary by data set of the upper limits for the coefficients of variation*

Data set	CV
Swiss Municipalities	0.1
American Community Survey, 2015	0.05
US Census, 2000	0.05
Kiva Loans	0.05
UN Commodity Trade Statistics data	0.05

We selected an upper precision level of 0.1 for the *Swiss Municipalities* data set in keeping with the level set for the experiment in Ballin and Barcaroli (2020). We used an upper precision level of 0.05 for the remaining experiments, given that the upper CV levels generally set by national statistics institutes (NSIs) tend to be between 0.01 and 0.1, and, for this reason, results for CVs in the mid-point of this range are of interest.

A.1.2 Hyperparameters for the grouping genetic algorithm and simulated annealing algorithm

Table A.1.2: *Hyperparameters for the grouping genetic algorithm (GGA)*

Swiss Municipalities							
Fine-tuning iteration No.	**GGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Sample Size**	**Execution Time**
1	3,500	50	0.0371727	0.3	0.0815300	151.83	1,764.27

A.

2	500	50	0.0975120	0.4	0.0284839	250.82	215.19
3	4,000	40	0.0231718	0.4	0.0423903	133.28	1,331.79
4	5,000	20	0.0450855	0.1	0.0756457	190.34	1,307.18
5	1,500	40	0.0101463	0.2	0.0508996	150.98	576.89
6	1,000	30	0.0007005	0.1	0.0066850	284.80	84.20
7	2,000	10	0.0614262	0.2	0.0319617	256.37	230.53
8	3,000	10	0.0531087	0.3	0.0189619	210.83	302.02
9	4,500	30	0.0813408	0.5	0.0610529	192.14	1,034.98
10	2,500	20	0.0702672	0.5	0.0974212	193.57	364.11
11	4,000	40	0.0028391	0.4	0.0068729	133.03	443.85
12	4,000	40	0.0000609	0.4	0.0002167	192.10	120.53
13	1,000	40	0.0015045	0.2	0.0379533	189.82	225.42
14	**4,000**	**50**	**0.0053360**	**0.4**	**0.0037620**	**128.69**	**753.82**
15	500	50	0.0063685	0.4	0.0038509	160.27	140.54
16	4,000	30	0.0051340	0.1	0.0037047	152.04	577.14
17	4,000	30	0.0026961	0.1	0.0060359	158.61	412.21
18	1,000	50	0.0008386	0.4	0.0064984	153.78	107.89
19	4,000	30	0.0005100	0.1	0.0102913	244.81	241.40
20	1,000	30	0.0108830	0.4	0.0066943	141.36	200.34
					Minimum	**128.69**	
						Total	**10,434.30**

American Community Survey, 2015							
Fine-tuning iteration No.	**GGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Sample Size**	**Execution Time**
1	4,000	20	0.0003717	0.3	0.0815300	10,645.70	13,048.22
2	1,000	20	0.0009751	0.4	0.0284839	11,030.60	3,343.32
3	4,000	20	0.0002317	0.4	0.0423903	10,354.37	9,979.46
4	5,000	10	0.0004509	0.1	0.0756457	11,737.36	9,358.08
5	2,000	20	0.0001015	0.2	0.0508996	11,546.75	5,696.86
6	1,000	10	0.0000070	0.1	0.0066850	11,753.83	837.38
7	2,000	10	0.0006143	0.2	0.0319617	11,527.01	3,713.57
8	3,000	10	0.0005311	0.3	0.0189619	10,958.55	4,870.35
9	**5,000**	**20**	**0.0008134**	**0.5**	**0.0610529**	**10,136.50**	**13,146.25**
10	3,000	10	0.0007027	0.5	0.0974212	10,641.01	3,722.84
11	5,000	20	0.0007753	0.1	0.0673091	11,515.73	21,389.95
12	5,000	20	0.0000217	0.5	0.0635569	11,168.07	4,621.45
13	4,000	20	0.0007946	0.1	0.0573782	11,577.85	16,983.77
14	5,000	10	0.0008209	0.5	0.0599857	10,370.84	6,493.16
15	5,000	10	0.0008311	0.1	0.0595253	11,741.48	10,654.80
16	5,000	10	0.0008342	0.5	0.0549796	10,360.82	6,567.07
17	5,000	20	0.0008727	0.1	0.0603612	11,505.02	21,426.06
18	2,000	10	0.0008064	0.5	0.0608317	10,856.85	2,658.84
19	5,000	10	0.0008207	0.2	0.0615856	11,281.42	10,044.17
20	5,000	20	0.0008618	0.5	0.0592633	10,151.25	13,596.86
					Minimum	**10,136.50**	
						Total	**182,152.46**

US Census, 2000							
Fine-tuning iteration No.	**GGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Sample Size**	**Execution Time**
1	50	20	0.0000007	0.5	0.0000469	234.48	1,295.05
2	100	10	0.0000010	0.2	0.0000966	231.94	1,176.81
3	50	20	0.0000008	0.1	0.0000374	234.48	1,618.25
4	100	10	0.0000005	0.2	0.0000188	231.24	1,228.17
5	100	10	0.0000001	0.3	0.0000658	232.81	1,058.13
6	100	20	0.0000003	0.5	0.0000846	234.48	2,090.42
7	100	20	0.0000006	0.4	0.0000090	230.68	2,181.77
8	50	10	0.0000003	0.1	0.0000775	234.48	729.07
9	50	10	0.0000001	0.3	0.0000241	233.47	899.24
10	50	20	0.0000007	0.4	0.0000590	234.48	1,447.75
11	50	20	0.0000006	0.2	0.0000034	233.16	1,613.14
12	100	20	0.0000007	0.4	0.0000091	231.01	2,238.11
13	100	20	0.0000008	0.1	0.0000114	234.38	2,677.98
14	100	20	0.0000005	0.1	0.0000084	232.86	3,023.89
15	100	20	0.0000006	0.4	0.0000356	230.44	2,276.24
16	**100**	**20**	**0.0000007**	**0.4**	**0.0000472**	**228.81**	**2,367.36**
17	100	20	0.0000007	0.4	0.0000495	232.12	2,388.77
18	50	20	0.0000004	0.4	0.0000474	234.48	1,394.61
19	100	20	0.0000004	0.4	0.0000473	229.98	2,363.21
20	100	20	0.0000008	0.4	0.0000472	232.38	2,230.38

					Minimum	**228.81**	
						Total	**36,298.35**

Kiva Loans							
Fine-tuning iteration No.	**GGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Sample Size**	**Execution Time**
1	1,000	20	0.0006495	0.3	0.0704822	7,222.08	7,229.63
2	3,000	10	0.0009548	0.1	0.0875907	7,568.52	16,361.34
3	1,000	10	0.0008895	0.3	0.0573803	7,307.78	4,067.61
4	2,000	10	0.0000439	0.1	0.0204694	7,583.08	4,260.30
5	1,000	10	0.0002903	0.4	0.0946378	7,317.01	2,158.01
6	**3,000**	**20**	**0.0007221**	**0.5**	**0.0685005**	**6,756.19**	**15,669.11**
7	3,000	20	0.0001983	0.4	0.0074480	6,923.39	11,485.64
8	3,000	20	0.0003937	0.5	0.0391558	6,801.63	12,378.56
9	2,000	10	0.0004802	0.2	0.0188587	7,433.70	7,245.60
10	2,000	20	0.0005150	0.2	0.0459069	7,316.76	14,944.48
11	3,000	10	0.0000981	0.5	0.0408975	7,263.70	4,223.41
12	3,000	20	0.0009322	0.5	0.0811859	6,760.24	18,719.82
13	3,000	20	0.0009843	0.5	0.0858498	6,771.46	19,160.63
14	3,000	20	0.0009715	0.1	0.0852278	7,467.85	33,530.93
15	3,000	20	0.0009896	0.5	0.0901260	6,756.67	19,303.41
16	3,000	10	0.0009893	0.5	0.0878967	6,894.50	9,617.54
17	3,000	20	0.0009986	0.1	0.0877172	7,474.19	34,294.95
18	2,000	20	0.0009959	0.5	0.0233675	6,863.03	12,945.70
19	3,000	20	0.0009848	0.1	0.0804226	7,477.55	33,675.61
20	3,000	20	0.0000585	0.5	0.0876780	7,200.64	7,674.51
					Minimum	**6,756.19**	
						Total	**288,946.79**

UN Commodity Trade Statistics data							
Fine-tuning iteration No.	**GGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Sample Size**	**Execution Time**
1	500	10	0.0005263	0.5	0.0126579	3,311.72	1,233.60
2	500	10	0.0002485	0.4	0.0207811	3,315.97	1,207.35
3	1,000	10	0.0007747	0.5	0.0435968	3,282.74	2,703.46
4	1,000	20	0.0000404	0.2	0.0570049	3,302.79	4,437.15
5	1,000	10	0.0003551	0.4	0.0700879	3,275.97	2,598.38
6	1,000	20	0.0001919	0.1	0.0007253	3,357.35	7,072.05
7	500	10	0.0006449	0.1	0.0806051	3,425.09	2,309.18
8	1,000	20	0.0004488	0.3	0.0313464	3,221.46	6,531.61
9	500	20	0.0008277	0.3	0.0937325	3,263.08	3,934.72
10	500	20	0.0009655	0.2	0.0673167	3,328.53	4,747.78
11	1,000	10	0.0005969	0.3	0.0831234	3,259.55	3,452.19
12	1,000	10	0.0004305	0.1	0.0790410	3,414.13	4,081.89
13	1,000	20	0.0004839	0.1	0.0937807	3,348.51	8,538.75
14	1,000	10	0.0004525	0.3	0.0809503	3,258.43	3,196.89
15	1,000	10	0.0004490	0.1	0.0855682	3,423.10	4,184.20
16	1,000	10	0.0004402	0.3	0.0877865	3,260.04	3,225.69
17	1,000	20	0.0004130	0.3	0.0794136	3,235.75	6,454.73
18	1,000	20	0.0004404	0.1	0.0293864	3,362.21	8,389.22
19	1,000	10	0.0006261	0.3	0.0777867	3,264.95	3,624.41
20	**1,000**	**20**	**0.0004493**	**0.3**	**0.0866266**	**3,216.68**	**6,535.97**
					Minimum	**3,216.68**	
						Total	**88,459.22**

Table A.1.3: *Hyperparameters for the simulated annealing algorithm (SAA)*

Swiss Municipalities								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probabilty of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	30	2,000	0.0002460	0.7029232	0.0246979	0.0736992	128.41	504.40
2	50	3,000	0.0006329	0.6373256	0.0197281	0.0433342	127.85	1,251.80
3	50	1,000	0.0005252	0.9310385	0.0156089	0.0830287	130.08	427.28
4	20	2,000	0.0008548	0.9607493	0.0210961	0.0530876	131.50	338.66
5	20	1,000	0.0004880	0.8347270	0.0097119	0.0019942	130.19	154.84
6	10	2,000	0.0000720	0.5149204	0.0033907	0.0214234	133.97	160.69

A.

7	40	3,000	0.0009095	0.6967157	0.0140365	0.0654427	131.12	1,051.31
8	10	2,000	0.0007935	0.7996514	0.0100971	0.0996045	134.48	164.61
9	40	1,000	0.0003585	0.8716979	0.0062788	0.0324787	132.23	326.64
10	30	3,000	0.0001746	0.5684019	0.0022779	0.0159946	132.70	774.41
11	30	2,000	0.0002806	0.6045232	0.0230599	0.0925182	128.04	480.18
12	30	2,000	0.0000722	0.5174297	0.0222745	0.0996578	126.15	459.47
13	10	2,000	0.0000066	0.6282987	0.0247290	0.0999727	126.51	157.94
14	**10**	**3,000**	**0.0000720**	**0.5083686**	**0.0183356**	**0.0997907**	**125.17**	**248.91**
15	10	3,000	0.0003082	0.5082227	0.0114061	0.0997231	132.04	243.53
16	10	2,000	0.0001031	0.5116027	0.0180108	0.0275291	130.04	162.72
17	40	3,000	0.0000377	0.5850357	0.0123964	0.0999682	132.44	898.16
18	10	3,000	0.0000565	0.7552034	0.0189523	0.0969085	131.20	246.24
19	10	3,000	0.0000361	0.5016947	0.0184217	0.0998266	131.65	236.72
20	30	2,000	0.0000725	0.6238461	0.0119237	0.0999920	129.49	520.12
						Minimum	**125.17**	
							Total	**8,808.63**

American Community Survey, 2015								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probabilty of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	3	3,000	0.0003317	0.7006875	0.0206152	0.0070649	10,366.59	527.02
2	1	3,000	0.0009052	0.8282285	0.0053825	0.0489284	10,943.88	178.48
3	3	3,000	0.0002090	0.8759595	0.0110168	0.0807005	10,321.25	528.49
4	3	1,000	0.0004926	0.5735955	0.0178851	0.0154427	10,865.74	179.47
5	1	3,000	0.0001389	0.6155435	0.0129485	0.0689619	11,101.47	177.23
6	1	2,000	0.0000134	0.5065807	0.0003622	0.0551158	10,967.49	119.97
7	1	1,000	0.0006310	0.6871060	0.0076583	0.0356508	11,277.27	63.27
8	2	1,000	0.0005134	0.7662841	0.0029761	0.0212996	10,939.02	120.03
9	3	2,000	0.0008713	0.9550385	0.0174063	0.0712182	10,556.15	347.62
10	2	2,000	0.0007946	0.9139578	0.0228174	0.0979716	10,852.16	236.10
11	3	3,000	0.0002558	0.7178641	0.0092048	0.0061117	10,291.61	519.55
12	3	3,000	0.0001224	0.6557205	0.0083747	0.0097838	10,295.77	532.50
13	1	3,000	0.0002393	0.6740939	0.0093276	0.0088139	11,035.19	176.25
14	3	3,000	0.0000868	0.6898501	0.0078855	0.0360372	10,303.33	519.06
15	**3**	**3,000**	**0.0002347**	**0.6873029**	**0.0076477**	**0.0291729**	**10,279.44**	**517.76**
16	3	3,000	0.0002642	0.6334884	0.0076700	0.0366039	10,281.53	518.11
17	2	3,000	0.0001786	0.6831962	0.0077415	0.0096088	10,525.35	347.17
18	3	3,000	0.0002182	0.6155311	0.0076564	0.0583217	10,287.99	520.78
19	3	3,000	0.0005624	0.5201874	0.0065580	0.0334981	10,284.12	517.37
20	3	1,000	0.0002862	0.6871434	0.0074680	0.0415287	10,837.77	176.19
						Minimum	**10,279.44**	
							Total	**6,822.42**

US Census								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probabilty of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	2	2,000	0.0003125	0.9571083	0.0247182	0.0527988	229.70	712.70
2	2	1,000	0.0002758	0.7437050	0.0213571	0.0430350	227.57	382.66
3	2	1,000	0.0008829	0.7516500	0.0183035	0.0951732	227.06	380.95
4	1	2,000	0.0009509	0.6405754	0.0035327	0.0084009	232.66	354.72
5	1	2,000	0.0007498	0.6533559	0.0067660	0.0736139	232.67	355.26
6	1	2,000	0.0006646	0.8587779	0.0017975	0.0325162	233.89	356.05
7	2	1,000	0.0005276	0.8462786	0.0160413	0.0110241	228.29	371.27
8	1	1,000	0.0004513	0.5506075	0.0090108	0.0878499	227.57	192.82
9	1	1,000	0.0001098	0.5410013	0.0123346	0.0650869	227.39	195.55
10	2	2,000	0.0000536	0.9010516	0.0146951	0.0273742	227.59	748.00
11	2	1,000	0.0005978	0.5442084	0.0020166	0.0958725	233.94	347.92
12	2	1,000	0.0000537	0.7254659	0.0070534	0.0970539	231.93	357.03
13	2	1,000	0.0007707	0.5115015	0.0130304	0.0959253	226.82	382.13
14	2	1,000	0.0007991	0.5111586	0.0082039	0.0957792	230.86	369.78
15	1	1,000	0.0007058	0.5845086	0.0095704	0.0317438	228.78	181.76
16	1	2,000	0.0009459	0.7938437	0.0113436	0.0959424	231.36	362.98
17	**2**	**2,000**	**0.0006706**	**0.5457192**	**0.0189395**	**0.0806919**	**224.75**	**741.75**
18	2	2,000	0.0006222	0.5439523	0.0198413	0.0958830	226.76	751.71
19	2	2,000	0.0006492	0.5449156	0.0088022	0.0799868	229.52	694.97
20	2	2,000	0.0006672	0.5455705	0.0197144	0.0979236	226.84	756.84
						Minimum	**224.75**	

							Total	**8,996.85**
Kiva Loans								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probabilty of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	5	1,000	0.0001599	0.5020936	0.0229094	0.0173918	6,813.39	329.02
2	2	2,000	0.0008544	0.9570320	0.0168327	0.0922868	6,926.94	264.29
3	4	1,000	0.0003293	0.7161904	0.0076080	0.0536694	6,863.14	267.52
4	4	1,000	0.0002856	0.6819111	0.0016192	0.0821119	6,861.72	263.55
5	2	2,000	0.0000752	0.8993816	0.0211231	0.0490094	6,931.46	264.07
6	1	1,000	0.0007084	0.9238656	0.0182417	0.0328898	7,280.65	69.94
7	3	1,000	0.0005266	0.5846864	0.0045428	0.0794436	6,938.87	199.16
8	5	2,000	0.0009284	0.7694508	0.0148094	0.0677245	6,653.89	657.36
9	1	2,000	0.0006809	0.8296335	0.0073496	0.0007920	7,131.55	134.95
10	3	2,000	0.0004487	0.6303538	0.0121813	0.0259957	6,783.32	393.91
11	1	2,000	0.0009295	0.7439049	0.0139223	0.0298497	7,201.11	134.30
12	**5**	**2,000**	**0.0009935**	**0.7806557**	**0.0143925**	**0.0317491**	**6,646.67**	**664.30**
13	1	2,000	0.0009972	0.7646035	0.0144326	0.0744305	7,199.54	134.53
14	5	2,000	0.0009959	0.7513781	0.0133791	0.0335832	6,651.29	658.62
15	5	1,000	0.0009990	0.7668263	0.0145028	0.0388398	6,802.15	329.86
16	5	2,000	0.0009811	0.9136393	0.0139105	0.0304594	6,654.05	665.28
17	5	2,000	0.0009348	0.9207675	0.0175411	0.0319848	6,654.40	664.43
18	5	2,000	0.0009306	0.9344035	0.0179341	0.0329171	6,658.51	664.31
19	1	2,000	0.0009769	0.9160663	0.0175526	0.0384021	7,212.87	134.34
20	5	2,000	0.0009664	0.9231640	0.0183190	0.0070315	6,666.28	656.13
						Minimum	**6,646.67**	
							Total	**7,549.87**
UN Commodity Trade Statistics data								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probabilty of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	2	3,000	0.0008702	0.5844437	0.0132398	0.0363670	3,175.43	788.73
2	3	3,000	0.0000968	0.7653975	0.0027153	0.0718499	3,235.94	1,146.20
3	3	2,000	0.0003649	0.7484819	0.0150550	0.0517558	3,167.86	777.55
4	1	1,000	0.0006688	0.5050122	0.0190043	0.0216363	3,287.42	140.44
5	2	1,000	0.0007369	0.9988117	0.0110597	0.0126288	3,230.83	267.59
6	1	1,000	0.0001813	0.8555969	0.0226920	0.0025923	3,290.31	136.42
7	2	3,000	0.0004423	0.6852189	0.0016653	0.0665130	3,276.73	763.36
8	3	2,000	0.0005544	0.6376329	0.0073737	0.0887542	3,203.70	768.80
9	2	2,000	0.0009626	0.8121415	0.0098445	0.0471295	3,206.36	512.63
10	1	3,000	0.0002775	0.9291000	0.0224728	0.0945319	3,275.77	389.01
11	1	2,000	0.0006155	0.5469176	0.0168792	0.0343758	3,282.92	262.13
12	3	2,000	0.0008268	0.5176092	0.0191755	0.0416852	3,161.93	767.85
13	3	2,000	0.0008405	0.5104915	0.0242267	0.0292217	3,138.71	777.94
14	1	2,000	0.0007888	0.5107481	0.0237482	0.0183538	3,285.62	266.09
15	1	2,000	0.0008772	0.5054989	0.0243308	0.0325604	3,280.53	269.00
16	3	2,000	0.0005351	0.5098311	0.0240315	0.0270329	3,147.95	810.00
17	3	2,000	0.0007949	0.5032587	0.0242244	0.0066236	3,143.54	807.20
18	3	2,000	0.0007772	0.5126007	0.0244247	0.0209834	3,143.75	804.37
19	2	2,000	0.0006909	0.5009057	0.0249583	0.0234605	3,174.16	537.23
20	3	3,000	0.0007902	0.5072737	0.0234728	0.0013775	3,120.07	1,169.26
						Minimum	**3,120.07**	
							Total	**12,161.80**

A.1.3 Hyperparameters for the classical genetic algorithm and simulated annealing algorithm

Table A.1.4: *Hyperparameters for the CGA*

Swiss Municipalities						
Fine-tuning iteration No.	**CGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Sample Size**	**Execution Time**
1	3,500	50	0.0371727	0.3	151.83	1,764.27
2	500	50	0.0975120	0.4	250.82	215.19
3	4,000	40	0.0231718	0.4	133.28	1,331.79
4	5,000	20	0.0450855	0.1	190.34	1,307.18
5	1,500	40	0.0101463	0.2	150.98	576.89
6	1,000	30	0.0007005	0.1	284.80	84.20
7	2,000	10	0.0614262	0.2	256.37	230.53
8	3,000	10	0.0531087	0.3	210.83	302.02
9	4,500	30	0.0813408	0.5	192.14	1,034.98
10	2,500	20	0.0702672	0.5	193.57	364.11
11	4,000	40	0.0028391	0.4	133.03	443.85
12	4,000	40	0.0000609	0.4	192.10	120.53
13	1,000	40	0.0015045	0.2	189.82	225.42
14	**4,000**	**50**	**0.0053360**	**0.4**	**128.69**	**753.82**
15	500	50	0.0063685	0.4	160.27	140.54
16	4,000	30	0.0051340	0.1	152.04	577.14
17	4,000	30	0.0026961	0.1	158.61	412.21
18	1,000	50	0.0008386	0.4	153.78	107.89
19	4,000	30	0.0005100	0.1	244.81	241.40
20	1,000	30	0.0108830	0.4	141.36	200.34
				Minimum	**128.69**	
					Total	**10,434.30**

American Community Survey, 2015						
Fine-tuning iteration No.	**CGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Sample Size**	**Execution Time**
1	1,000	20	0.0003717	0.3	4,352.32	17,378.06
2	500	20	0.0009751	0.4	4,309.63	7,388.80
3	1,000	20	0.0002317	0.4	4,427.53	14,705.11
4	1,000	10	0.0004509	0.1	4,460.27	10,936.87
5	500	20	0.0001015	0.2	4,721.48	9,811.80
6	500	10	0.0000070	0.1	5,532.79	5,583.36
7	500	10	0.0006143	0.2	4,612.39	5,054.34
8	1,000	10	0.0005311	0.3	4,470.13	8,708.80
9	1,000	20	0.0008134	0.5	4,287.84	12,283.05
10	500	10	0.0007027	0.5	4,671.85	3,114.68
11	1,000	20	0.0000373	0.4	4,886.08	14,818.32
12	**1,000**	**20**	**0.0009952**	**0.1**	**4,197.68**	**22,016.95**
13	1,000	20	0.0000075	0.1	4,926.75	21,979.72
14	1,000	20	0.0000028	0.3	5,140.69	17,480.65
15	500	20	0.0009716	0.1	4,244.86	11,080.55
16	500	20	0.0009679	0.1	4,295.00	11,161.50
17	500	20	0.0000059	0.1	5,037.11	11,084.17
18	500	10	0.0008502	0.1	4,459.32	5,618.89
19	500	20	0.0000041	0.5	5,154.22	6,215.87
20	500	20	0.0001935	0.1	4,545.09	11,214.02
				Minimum	**4,197.68**	
					Total	**227,635.51**

US Census, 2000						
Fine-tuning iteration No.	**CGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Sample Size**	**Execution Time**
1	400	20	0.0003717	0.3	213.48	4,088.52
2	100	20	0.0009751	0.4	243.99	882.86
3	**400**	**20**	**0.0002317**	**0.4**	**192.71**	**3,361.90**
4	500	10	0.0004509	0.1	251.55	3,270.99
5	200	20	0.0001015	0.2	247.50	2,372.14
6	100	10	0.0000070	0.1	500.75	641.81
7	200	10	0.0006143	0.2	268.06	1,170.59
8	300	10	0.0005311	0.3	252.60	1,538.53
9	500	20	0.0008134	0.5	202.31	3,788.72
10	300	10	0.0007027	0.5	302.11	1,115.88
11	400	20	0.0000298	0.4	289.20	3,586.16
12	400	20	0.0000170	0.3	229.12	3,950.58
13	100	20	0.0000051	0.3	251.27	996.59

14	500	20	0.0000117	0.1	238.01	6,168.77
15	100	20	0.0000091	0.1	353.79	1,296.33
16	500	20	0.0000080	0.5	255.12	3,502.78
17	100	10	0.0002115	0.3	357.12	521.23
18	400	10	0.0000024	0.3	269.87	1,891.18
19	100	20	0.0008485	0.1	231.21	1,310.39
20	100	20	0.0000020	0.1	278.16	1,345.83
				Minimum	**192.71**	
					Total	**46,801.78**

Kiva Loans						
Fine-tuning iteration No.	**CGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Sample Size**	**Execution Time**
1	200	20	0.0192864	0.5	3,652.36	3,436.42
2	200	20	0.0282890	0.4	3,525.36	3,922.12
3	100	10	0.0917037	0.1	3,090.45	1,485.47
4	100	10	0.0564565	0.2	3,383.89	1,311.05
5	100	20	0.0360430	0.1	3,477.38	2,940.12
6	100	10	0.0095998	0.3	4,380.52	1,164.07
7	200	10	0.0612849	0.2	3,401.27	2,609.40
8	**200**	**20**	**0.0817285**	**0.5**	**3,062.33**	**3,232.78**
9	200	20	0.0788767	0.3	3,156.47	4,540.05
10	100	10	0.0466330	0.4	4,358.50	1,029.31
11	100	20	0.0923780	0.3	3,189.80	2,297.47
12	100	10	0.0920346	0.4	3,841.25	987.86
13	100	20	0.0065226	0.3	4,094.16	2,210.59
14	200	10	0.0138694	0.3	4,187.53	2,295.96
15	200	10	0.0827815	0.3	3,408.99	2,253.28
16	100	10	0.0080602	0.1	4,781.26	1,456.57
17	100	10	0.0806664	0.1	3,335.59	1,461.18
18	100	10	0.0916848	0.1	3,699.34	1,473.89
19	200	10	0.0087988	0.1	5,023.73	2,908.91
20	200	20	0.0180915	0.1	3,429.61	5,730.11
				Minimum	**3,062.33**	
					Total	**48,746.61**

UN Commodity Trade Statistics						
Fine-tuning iteration No.	**CGA Iterations, I**	**Chromosome Population Size, N_p**	**Mutation Chance**	**Elitism Rate**	**Sample Size**	**Execution Time**
1	5,000	10	0.0008185	0.3	4,241.30	8,670.78
2	4,000	20	0.0002775	0.1	4,321.59	17,573.86
3	5,000	30	0.0001544	0.5	4,236.66	18,024.99
4	4,000	20	0.0006314	0.3	4,080.98	13,338.40
5	4,000	10	0.0003255	0.2	5,850.95	7,599.14
6	4,000	10	0.0007229	0.4	5,146.34	6,212.83
7	5,000	20	0.0009345	0.4	3,890.98	15,424.27
8	5,000	30	0.0004745	0.5	3,803.70	18,453.79
9	4,000	20	0.0000177	0.1	10,586.47	17,367.77
10	**5,000**	**30**	**0.0005599**	**0.2**	**3,619.42**	**29,045.23**
11	5,000	30	0.0000828	0.1	4,214.45	32,576.08
12	5,000	20	0.0000319	0.5	10,426.94	12,035.86
13	5,000	30	0.0000151	0.1	6,224.89	32,065.47
14	5,000	20	0.0000493	0.3	7,496.26	16,562.43
15	5,000	20	0.0004824	0.5	4,033.86	12,052.30
16	5,000	30	0.0000498	0.5	6,284.36	17,838.79
17	5,000	20	0.0001468	0.5	5,481.04	11,941.31
18	5,000	20	0.0000280	0.4	10,693.88	14,368.42
19	5,000	20	0.0005598	0.4	4,028.03	14,350.27
20	4,000	20	0.0000374	0.4	9,902.05	11,429.64
				Minimum	**3,619.42**	
					Total	**326,931.63**

Table A.1.5: *Hyperparameters for the simulated annealing algorithm (SAA)*

Swiss Municipalities								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	3	1,000	0.052285432	0.7534290	0.8717312	0.0777226	175.21	25.39
2	4	4,000	0.012453166	0.8707287	0.5744123	0.0100954	130.46	132.39
3	1	2,000	0.002051267	0.6045356	0.9179763	0.0024819	221.90	16.95
4	5	3,000	0.069538414	0.5685597	0.4337133	0.0285745	127.17	127.80
5	1	5,000	0.026313403	0.9284987	0.3584902	0.0365486	227.26	40.90
6	4	3,000	0.090783653	0.8330255	0.6858114	0.0624310	159.97	102.19
7	5	2,000	0.030121215	0.6711942	0.1623457	0.0515584	151.03	84.36
8	2	4,000	0.083494333	0.7467368	0.5406799	0.0948874	175.94	68.31
9	2	1,000	0.072956660	0.9577828	0.7657191	0.0413879	193.04	17.19
10	3	5,000	0.047123230	0.5209623	0.1974387	0.0852182	126.75	126.33
11	3	5,000	0.001357700	0.5425735	0.3169109	0.0058420	136.14	126.19
12	1	5,000	0.017851046	0.5212289	0.2446014	0.0213283	221.33	41.76
13	4	5,000	0.015349897	0.5169772	0.3567790	0.0200415	134.02	169.81
14	1	3,000	0.012220632	0.5162845	0.3432167	0.0133785	226.70	25.57
15	3	5,000	0.007170361	0.5057170	0.4377089	0.0039782	138.89	124.02
16	5	5,000	0.019719259	0.9219409	0.3387116	0.0246884	129.93	210.96
17	**5**	**5,000**	**0.023110573**	**0.9427609**	**0.3736443**	**0.0229361**	**120.00**	**213.44**
18	3	5,000	0.024918819	0.9674243	0.3650243	0.0216164	132.42	126.64
19	2	5,000	0.022197888	0.9387425	0.4002330	0.0313204	141.58	83.86
20	1	5,000	0.039290910	0.9516527	0.4486647	0.0237672	224.75	41.76
						Minimum	**120.00**	
							Total	**1,905.82**

American Community Survey, 2015								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	50	2,000	0.000000332	0.7006875	0.0002236	0.0000001	4,278.98	9,472.16
2	30	2,000	0.000000905	0.8282285	0.0001318	0.0000005	4,455.26	7,798.33
3	50	2,000	0.000000209	0.8759595	0.0001658	0.0000008	3,928.33	13,255.75
4	50	1,000	0.000000493	0.5735955	0.0002071	0.0000002	5,493.81	3,087.65
5	30	2,000	0.000000139	0.6155435	0.0001774	0.0000007	4,665.47	6,484.61
6	30	1,000	0.000000013	0.5065807	0.0001016	0.0000006	6,120.55	1,818.47
7	30	1,000	0.000000631	0.6871060	0.0001455	0.0000004	5,244.37	3,859.64
8	40	1,000	0.000000513	0.7662841	0.0001173	0.0000002	4,922.72	5,144.33
9	50	2,000	0.000000871	0.9550385	0.0002043	0.0000007	3,941.14	13,467.53
10	40	1,000	0.000000795	0.9139578	0.0002369	0.0000010	4,922.65	5,166.95
11	**50**	**2,000**	**0.000000045**	**0.9528952**	**0.0001021**	**0.0000008**	**3,915.48**	**13,351.19**
12	50	2,000	0.000000028	0.5371501	0.0001007	0.0000008	5,097.75	4,378.83
13	50	2,000	0.000000008	0.9666875	0.0001015	0.0000008	3,930.67	13,506.55
14	50	2,000	0.000000014	0.9563341	0.0001018	0.0000006	3,929.90	13,349.35
15	50	2,000	0.000000012	0.9117070	0.0001017	0.0000006	3,927.90	13,400.80
16	50	1,000	0.000000026	0.9743898	0.0001044	0.0000005	4,663.33	6,576.00
17	50	2,000	0.000000014	0.5169004	0.0001016	0.0000007	5,263.99	3,941.12
18	50	2,000	0.000000013	0.9084592	0.0001001	0.0000006	3,927.58	13,565.64
19	30	1,000	0.000000007	0.8707087	0.0001001	0.0000006	5,236.27	3,928.38
20	50	2,000	0.000000005	0.8692340	0.0001016	0.0000006	3,920.20	13,562.64
						Minimum	**3,915.48**	
							Total	**169,115.92**

US Census, 2000								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	1	2,000	0.000957473	0.8673938	0.0137552	0.0447840	181.97	48.88
2	1	1,000	0.000470984	0.9971055	0.0226560	0.0188363	181.66	26.56
3	2	1,000	0.000650982	0.6336578	0.0055086	0.0969599	188.15	48.17
4	1	2,000	0.000263105	0.7842547	0.0100731	0.0034531	184.53	48.00
5	2	1,000	0.000792690	0.6855781	0.0027820	0.0731402	190.14	49.42
6	1	2,000	0.000167589	0.8087646	0.0155608	0.0206425	182.01	48.75
7	1	1,000	0.000027559	0.5403833	0.0024228	0.0593191	189.72	24.65

8	2	2,000	0.000583566	0.9403833	0.0100076	0.0822596	184.07	95.25
9	2	1,000	0.000897649	0.7161330	0.0182939	0.0356279	183.19	48.18
10	2	2,000	0.000344408	0.5924741	0.0211623	0.0665908	181.94	96.47
11	1	1,000	0.000016004	0.5298720	0.0231786	0.0195540	181.42	25.26
12	1	2,000	0.000018897	0.9695404	0.0026302	0.0152296	191.79	48.53
13	2	2,000	0.000006799	0.9707322	0.0219573	0.0133887	181.12	95.10
14	**1**	**2,000**	**0.000019997**	**0.9665631**	**0.0221147**	**0.0160408**	**179.89**	**51.94**
15	1	2,000	0.000019486	0.9169971	0.0037733	0.0170926	189.25	48.86
16	2	2,000	0.000019057	0.9685325	0.0209424	0.0152331	182.79	97.46
17	2	2,000	0.000010810	0.9619129	0.0069297	0.0160281	188.36	95.26
18	1	2,000	0.000001123	0.9858940	0.0229735	0.0738177	183.07	48.45
19	1	2,000	0.000018872	0.5186935	0.0225324	0.0149033	181.17	53.06
20	1	2,000	0.000015764	0.9706810	0.0081846	0.0123067	187.08	49.11
						Minimum	**179.89**	
							Total	**1,147.36**

Kiva Loans								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	1	1,000	0.000981906	0.9457190	0.0174738	0.0393213	3,599.90	83.48
2	1	1,000	0.000655920	0.9587443	0.0128550	0.0853692	3,578.24	84.64
3	**2**	**2,000**	**0.000538391**	**0.8660943**	**0.0014281**	**0.0216320**	**3,017.79**	**300.16**
4	2	2,000	0.000801771	0.7408514	0.0214577	0.0489079	3,125.42	298.78
5	2	2,000	0.000187353	0.8001870	0.0105165	0.0058942	3,105.84	305.48
6	2	2,000	0.000701907	0.6093193	0.0047224	0.0746280	3,067.50	301.55
7	1	1,000	0.000057746	0.5165655	0.0199479	0.0681241	3,620.04	81.91
8	1	2,000	0.000328496	0.7848433	0.0085076	0.0593833	3,472.99	156.89
9	1	1,000	0.000470193	0.5794217	0.0242193	0.0163568	3,626.58	86.09
10	2	1,000	0.000298510	0.6959190	0.0051002	0.0940684	3,275.66	157.39
11	2	2,000	0.000588620	0.5336602	0.0234903	0.0147222	3,115.71	301.97
12	1	2,000	0.000562786	0.8975400	0.0024679	0.0188372	3,334.02	158.85
13	2	1,000	0.000521897	0.5749066	0.0202772	0.0290174	3,306.66	156.59
14	2	1,000	0.000973641	0.9305538	0.0008023	0.0017096	3,218.07	156.73
15	2	2,000	0.000669995	0.9691405	0.0240978	0.0840468	3,101.27	301.38
16	2	2,000	0.000039912	0.9627319	0.0248438	0.0808439	3,107.35	303.72
17	2	2,000	0.000995324	0.5169494	0.0030456	0.0861112	3,064.74	300.69
18	1	2,000	0.000989806	0.5251522	0.0248557	0.0772447	3,557.47	154.07
19	2	2,000	0.000990423	0.5140439	0.0019326	0.0653920	3,024.65	303.72
20	2	1,000	0.000992703	0.5114841	0.0249080	0.0756890	3,319.09	154.97
						Minimum	**3,017.79**	
							Total	**4,149.06**

UN Commodity Trade Statistics								
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**Temperature, T**	**Decrement constant, DC**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H+ 1)**	**Sample Size**	**Execution Time**
1	2	250	0.000240473	0.5030943	0.0116457	0.0394269	3,289.63	78.32
2	2	250	0.000900782	0.8510434	0.0050178	0.0787035	3,324.06	74.89
3	1	250	0.000496287	0.9851146	0.0237895	0.0519271	3,303.16	43.17
4	2	250	0.000721745	0.7854319	0.0177733	0.0266660	3,277.64	74.34
5	1	250	0.000543522	0.8264456	0.0089342	0.0969365	3,343.34	42.22
6	1	250	0.000632438	0.7022129	0.0067530	0.0157643	3,366.27	41.77
7	1	250	0.000388473	0.6603318	0.0146680	0.0027671	3,325.35	41.78
8	2	250	0.000156551	0.9340585	0.0015163	0.0659874	3,358.09	73.59
9	2	250	0.000857761	0.5508424	0.0205828	0.0495423	3,266.40	73.78
10	1	250	0.000037784	0.6280942	0.0158217	0.0860740	3,313.66	42.36
11	2	250	0.000754307	0.5490271	0.0086306	0.0193790	3,312.51	73.30
12	2	250	0.000810211	0.8981822	0.0191303	0.0174392	3,267.66	73.71
13	2	250	0.000839566	0.9249103	0.0195673	0.0132012	3,267.29	73.06
14	1	250	0.000988895	0.7373825	0.0193662	0.0085304	3,317.97	41.98
15	**2**	**250**	**0.000674811**	**0.9309940**	**0.0203113**	**0.0149499**	**3,258.52**	**73.18**
16	2	250	0.000692824	0.9306800	0.0019885	0.0154043	3,357.00	72.93
17	2	250	0.000653517	0.9500990	0.0206834	0.0155980	3,265.87	74.09
18	2	250	0.000082588	0.9401383	0.0208771	0.0151883	3,264.79	73.07
19	2	250	0.000672200	0.9647793	0.0012812	0.0141657	3,366.76	72.92
20	2	250	0.000667377	0.9575300	0.0201864	0.0143703	3,287.52	72.92
						Minimum	**3,258.52**	
							Total	**1,287.38**

Appendix B

B.1 Descriptions of target and auxiliary variables for the HEDA experiment data sets

Table B.1.1: *Breakdown by data set of descriptions for target and auxiliary variables*

Data set	Variable	Description
SwissMunicipalities	Surfacebois	wood area.
SwissMunicipalities	Airbat	area with buildings.
SwissMunicipalities	POPTOT	total population
SwissMunicipalities	Hapoly	municipality area.
American Community Survey, 2015	HINCP	Household income (past 12 months)
American Community Survey, 2015	VALP	Property value
American Community Survey, 2015	SMOCP	Selected monthly owner costs
American Community Survey, 2015	INSP	Fire/hazard/flood insurance (yearly amount)
American Community Survey, 2015	BLD	Units in structure
American Community Survey, 2015	TEN	TEN
American Community Survey, 2015	WKEXREL	Work experience of householder and spouse
American Community Survey, 2015	WORKSTAT	Work status of householder or spouse in family households
American Community Survey, 2015	HFL	House heating fuel
American Community Survey, 2015	YBL	When structure first built
Kiva Loans	term_in_months	The duration for which the loan was disbursed in months
Kiva Loans	lender_count	The total number of lenders that contributed to this loan
Kiva Loans	loan_amount	The amount disbursed by the field agent to the borrower (USD)
Kiva Loans	sector	High level category, e.g. Food
Kiva Loans	currency	The currency in which the loan was disbursed
Kiva Loans	activity	More granular category, e.g. Fruits & Vegetables
Kiva Loans	region	Full region name within the country
Kiva Loans	partner_id	ID of partner organization

B.2 Fine-tuning the hyperparameters for the hybrid Estimation of Distribution Algorithm - Atomic Strata

Table B.2.2: *Hyperparameters for the HEDA in the case of Atomic Strata*

Swiss Municipalities							
Fine-tuning iteration No.	**Iterations, I**	**SAA Frequency**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Temperature, T**	**Decrement Constant, DC**
1	5,000	100	0.0004011	0.15	0.0000008	0.00000002	0.6933468
2	3,000	100	0.0009943	0.15	0.0000002	0.00000005	0.5736595
3	5,000	100	0.0002985	0.2	0.0000005	0.00000009	0.9229554
4	5,000	50	0.0004681	0.1	0.0000008	0.00000003	0.7242638
5	3,000	100	0.0001686	0.1	0.0000005	0.00000007	0.8150428
6	3,000	50	0.0000172	0.1	0.0000000	0.00000006	0.6266293
7	4,000	50	0.0006155	0.1	0.0000004	0.00000004	0.8724350
8	4,000	50	0.0005785	0.15	0.0000002	0.00000003	0.9842787
9	5,000	100	0.0008674	0.2	0.0000006	0.00000008	0.7707891
10	4,000	50	0.0007308	0.2	0.0000010	0.00000010	0.5456765
11	3,000	100	0.0007510	0.1	0.0000006	0.00000007	0.8459806
12	4,000	50	0.0007705	0.1	0.0000004	0.00000006	0.8436530
13	3,000	100	0.0001289	0.15	0.0000006	0.00000007	0.8117457
14	3,000	100	0.0001827	0.2	0.0000003	0.00000007	0.8184057
15	3,000	100	0.0001847	0.2	0.0000003	0.00000003	0.8292686
16	3,000	50	0.0001903	0.2	0.0000003	0.00000004	0.8295554
17	3,000	100	0.0001852	0.2	0.0000001	0.00000003	0.9414117
18	4,000	100	0.0001856	0.2	0.0000001	0.00000001	0.8659521
19	3,000	100	0.0001844	0.2	0.0000008	0.00000002	0.8288662
20	3,000	100	0.0001343	0.2	0.0000002	0.00000003	0.9526036
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**	**Sample Size**	**Execution Time**
1	1	10,000	0.0801744	0.0007444	20	128.98	840.66
2	2	30,000	0.0546600	0.0096341	20	131.21	3,752.61
3	1	20,000	0.0458527	0.0073244	20	128.19	1,314.53
4	1	30,000	0.0287394	0.0048503	20	129.06	5,527.23
5	2	20,000	0.0120236	0.0060850	20	133.58	1,353.60
6	2	10,000	0.0670922	0.0037592	20	126.99	886.40
7	2	20,000	0.0821999	0.0028418	20	125.84	3,697.88
8	2	30,000	0.0625123	0.0057328	20	127.06	7,791.48
9	1	20,000	0.0946376	0.0082044	20	126.42	1,367.88
10	1	10,000	0.0233130	0.0018504	20	130.34	659.67
11	2	20,000	0.0954986	0.0064802	20	125.10	1,304.16
12	2	20,000	0.0135087	0.0033416	20	130.00	3,382.07
13	2	20,000	0.0880779	0.0050088	20	126.38	1,379.07
14	2	20,000	0.0958774	0.0076491	20	122.92	1,316.11
15	2	20,000	0.0101602	0.0088849	20	138.17	1,240.32
16	2	30,000	0.0988187	0.0085451	20	125.56	5,776.53
17	2	20,000	0.0990880	0.0091836	20	122.39	1,337.31
18	2	20,000	0.0105801	0.0090396	20	133.49	1,587.33
19	2	20,000	0.0992476	0.0093288	20	128.43	1,319.65
20	2	20,000	0.0102580	0.0082125	20	133.65	1,211.38
					Minimum	122.39	
						Total	47,045.87
American Community Survey, 2015							
Fine-tuning iteration No.	**Iterations, I**	**SAA Frequency**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Temperature, T**	**Decrement Constant, DC**
1	300	10	0.0000000	0.4	0.0000001	0.00009222	0.6933468
2	200	10	0.0000001	0.4	0.0000000	0.00042573	0.5736595
3	300	10	0.0000000	0.5	0.0000000	0.00085991	0.9229554
4	300	5	0.0000000	0.3	0.0000001	0.00019888	0.7242638
5	200	10	0.0000000	0.3	0.0000001	0.00065586	0.8150428
6	200	5	0.0000000	0.3	0.0000000	0.00057615	0.6266293
7	200	5	0.0000001	0.3	0.0000000	0.00033515	0.8724350
8	300	5	0.0000001	0.4	0.0000000	0.00021361	0.9842787
9	300	10	0.0000001	0.5	0.0000001	0.00077701	0.7707891
10	200	5	0.0000001	0.5	0.0000001	0.00095672	0.5456765
11	200	10	0.0000000	0.3	0.0000001	0.00013141	0.8217492
12	300	10	0.0000000	0.5	0.0000001	0.00004408	0.7949279
13	300	10	0.0000001	0.3	0.0000001	0.00011081	0.7006825
14	300	10	0.0000000	0.4	0.0000001	0.00001201	0.8424332
15	300	10	0.0000000	0.5	0.0000001	0.00015274	0.8056020

B.

16	300	10	0.0000001	0.3	0.0000001	0.00009295	0.6690758
17	200	10	0.0000000	0.4	0.0000001	0.00007020	0.7462928
18	300	10	0.0000000	0.3	0.0000001	0.00076447	0.8238593
19	300	10	0.0000001	0.5	0.0000001	0.00009108	0.6111786
20	200	10	0.0000001	0.4	0.0000001	0.00043680	0.7087167
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**	**Sample Size**	**Execution Time**
1	10	2,500	0.0216957	0.0000001	20	8,849.13	8,542.77
2	20	5,000	0.0174433	0.0000010	20	8,829.25	28,978.64
3	10	2,500	0.0159755	0.0000007	20	8,834.01	6,984.81
4	10	5,000	0.0131232	0.0000005	20	8,805.62	36,489.00
5	20	5,000	0.0103373	0.0000006	20	8,800.53	28,857.52
6	20	2,500	0.0195154	0.0000004	20	8,824.74	20,805.10
7	20	2,500	0.0220333	0.0000003	20	8,816.72	20,756.65
8	20	5,000	0.0187521	0.0000006	20	8,811.21	89,838.25
9	10	5,000	0.0241063	0.0000008	20	8,838.34	17,576.73
10	10	2,500	0.0122188	0.0000002	20	8,809.30	9,145.87
11	20	5,000	0.0110847	0.0000001	20	8,801.18	28,832.86
12	20	5,000	0.0100309	0.0000001	20	8,810.14	44,465.19
13	20	5,000	0.0143632	0.0000001	20	8,807.34	44,395.63
14	10	2,500	0.0109556	0.0000001	20	8,822.30	7,088.03
15	20	5,000	0.0217767	0.0000000	20	8,819.96	44,605.16
16	10	5,000	0.0154209	0.0000006	20	8,824.89	17,641.07
17	20	5,000	0.0153991	0.0000007	20	8,815.61	28,870.49
18	10	5,000	0.0111377	0.0000007	20	8,811.88	17,556.67
19	20	5,000	0.0105166	0.0000000	20	8,804.58	44,660.20
20	10	2,500	0.0103235	0.0000004	20	8,850.46	4,585.94
					Minimum	8,800.53	
						Total	550,676.58
Kiva Loans							
Fine-tuning iteration No.	**Iterations, I**	**SAA Frequency**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Temperature, T**	**Decrement Constant, DC**
1	600	300	0.0000396	0.2	0.0000008	0.00100396	0.6933468
2	400	300	0.0000994	0.3	0.0000002	0.00430903	0.5736595
3	600	300	0.0000292	0.3	0.0000005	0.00861177	0.9229554
4	600	100	0.0000463	0.2	0.0000008	0.00206098	0.7242638
5	400	300	0.0000161	0.2	0.0000005	0.00658958	0.8150428
6	400	200	0.0000008	0.2	0.0000000	0.00579972	0.6266293
7	500	100	0.0000612	0.2	0.0000003	0.00341138	0.8724350
8	500	100	0.0000575	0.3	0.0000002	0.00220699	0.9842787
9	600	200	0.0000866	0.3	0.0000006	0.00779018	0.7707891
10	500	200	0.0000728	0.3	0.0000010	0.00957111	0.5456765
11	600	100	0.0000388	0.3	0.0000007	0.00140096	0.9542183
12	600	100	0.0000058	0.3	0.0000007	0.00025835	0.9846227
13	600	100	0.0000402	0.3	0.0000008	0.00950009	0.9444742
14	600	300	0.0000919	0.3	0.0000005	0.00164402	0.9690933
15	600	100	0.0000229	0.2	0.0000010	0.00040293	0.6803285
16	600	300	0.0000342	0.3	0.0000009	0.00065333	0.9685014
17	600	100	0.0000345	0.3	0.0000008	0.00030204	0.9093287
18	500	100	0.0000228	0.3	0.0000004	0.00848591	0.9809824
19	600	100	0.0000375	0.3	0.0000009	0.00109957	0.6936422
20	600	100	0.0000281	0.3	0.0000009	0.00166877	0.6873736
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**	**Sample Size**	**Execution Time**
1	1	5,000	0.0216957	0.0000075	20	6,536.39	7,325.52
2	3	10,000	0.0174433	0.0000963	20	6,338.61	6,190.59
3	2	5,000	0.0159755	0.0000733	20	6,559.13	3,720.53
4	1	10,000	0.0131232	0.0000486	20	6,285.78	2,837.05
5	3	10,000	0.0103373	0.0000609	20	6,374.74	1,422.56
6	3	5,000	0.0195154	0.0000377	20	6,441.66	1,111.02
7	2	5,000	0.0220333	0.0000285	20	6,321.22	1,768.59
8	3	10,000	0.0187521	0.0000574	20	6,265.44	3,174.16
9	2	10,000	0.0241063	0.0000821	20	6,349.25	2,045.19
10	1	5,000	0.0122188	0.0000186	20	6,447.99	1,280.92

11	2	10,000	0.0151998	0.0000524	20	6,246.74	3,292.97
12	2	5,000	0.0162442	0.0000388	20	6,287.11	2,154.07
13	2	5,000	0.0152621	0.0000566	20	6,439.26	2,109.17
14	2	5,000	0.0150536	0.0000973	20	6,466.34	1,400.07
15	1	10,000	0.0187736	0.0000071	20	6,250.33	2,703.06
16	2	10,000	0.0195352	0.0000050	20	6,365.74	1,649.00
17	1	5,000	0.0162050	0.0000009	20	6,313.38	1,888.96
18	1	10,000	0.0141372	0.0000678	20	6,428.67	2,138.39
19	1	10,000	0.0159892	0.0000338	20	6,250.00	2,677.30
20	2	5,000	0.0160194	0.0000343	20	6,274.33	2,117.25
					Minimum	6,246.74	
						Total	53,006.37

B.3 Fine-tuning the hyperparameters for the hybrid Estimation of Distribution Algorithm - Continuous Strata

Table B.3.3: *Hyperparameters for the HEDA in the case of Continuous Strata*

Swiss Municipalities							
Fine-tuning iteration No.	**Iterations, I**	**SAA Frequency**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Temperature, T**	**Decrement Constant, DC**
1	3,000	200	0.0000396	0.2	0.00000082	0.0000010	0.6933468
2	1,000	200	0.0000994	0.3	0.00000023	0.0000043	0.5736595
3	3,000	200	0.0000292	0.3	0.00000049	0.0000086	0.9229554
4	3,000	100	0.0000463	0.2	0.00000076	0.0000021	0.7242638
5	1,000	200	0.0000161	0.2	0.00000054	0.0000066	0.8150428
6	1,000	100	0.0000008	0.2	0.00000003	0.0000058	0.6266293
7	2,000	100	0.0000612	0.2	0.00000035	0.0000034	0.8724350
8	2,000	100	0.0000575	0.3	0.00000020	0.0000022	0.9842787
9	3,000	200	0.0000866	0.3	0.00000063	0.0000078	0.7707891
10	2,000	100	0.0000728	0.3	0.00000098	0.0000096	0.5456765
11	2,000	100	0.0000041	0.3	0.00000006	0.0000098	0.6376087
12	3,000	200	0.0000018	0.3	0.00000002	0.0000082	0.5233369
13	3,000	100	0.0000002	0.3	0.00000004	0.0000069	0.8926658
14	1,000	100	0.0000011	0.2	0.00000003	0.0000076	0.8248724
15	2,000	200	0.0000008	0.3	0.00000002	0.0000013	0.6029297
16	1,000	100	0.0000007	0.2	0.00000003	0.0000056	0.5545023
17	1,000	200	0.0000002	0.3	0.00000003	0.0000088	0.6750674
18	1,000	100	0.0000008	0.2	0.00000040	0.0000008	0.5495252
19	**1,000**	**100**	**0.0000677**	**0.2**	**0.00000002**	**0.0000057**	**0.5490809**
20	**1,000**	**200**	**0.0000008**	**0.2**	**0.00000008**	**0.0000085**	**0.6441513**
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**	**Sample Size**	**Execution Time**
1	5	3,000	0.0216957	0.0000075	20	107.80	458.67
2	10	5,000	0.0174433	0.0000963	20	107.94	218.91
3	5	4,000	0.0159755	0.0000733	20	108.06	364.22
4	5	5,000	0.0131232	0.0000486	20	107.91	629.75
5	10	4,000	0.0103373	0.0000609	20	107.82	161.03
6	10	3,000	0.0195154	0.0000377	20	104.93	207.11
7	10	4,000	0.0220333	0.0000285	20	107.74	571.95
8	10	5,000	0.0187521	0.0000574	20	107.48	728.80
9	5	4,000	0.0241063	0.0000821	20	108.03	359.16
10	5	3,000	0.0122188	0.0000186	20	108.32	296.15
11	5	3,000	0.0156453	0.0000395	20	107.76	296.91
12	5	4,000	0.0179254	0.0000770	20	107.89	371.97
13	10	3,000	0.0155686	0.0000116	20	106.97	674.13
14	5	3,000	0.0213493	0.0000442	20	106.65	139.98

B.

15	10	3,000	0.0191401	0.0000816	20	107.40	280.55
16	10	4,000	0.0119913	0.0000363	20	107.00	277.91
17	10	5,000	0.0195217	0.0000438	20	106.90	201.23
18	10	3,000	0.0188604	0.0000191	20	107.65	213.88
19	5	4,000	0.0235152	0.0000413	**20**	107.03	174.75
20	10	4,000	0.0218042	0.0000257	**20**	**105.95**	165.34
					Minimum	**104.93**	
						Total	6,792.40
American Community Survey, 2015							
Fine-tuning iteration No.	**Iterations, I**	**SAA Frequency**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Temperature, T**	**Decrement Constant, DC**
1	30	15	0.0000004	0.2	0.0000008	0.0000001	0.6933468
2	20	15	0.0000010	0.3	0.0000002	0.0000004	0.5736595
3	30	15	0.0000003	0.3	0.0000005	0.0000009	0.9229554
4	30	10	0.0000005	0.2	0.0000008	0.0000002	0.7242638
5	20	15	0.0000002	0.2	0.0000005	0.0000007	0.8150428
6	20	10	0.0000000	0.2	0.0000000	0.0000006	0.6266293
7	20	10	0.0000006	0.2	0.0000003	0.0000003	0.8724350
8	30	10	0.0000006	0.3	0.0000002	0.0000002	0.9842787
9	30	15	0.0000009	0.3	0.0000006	0.0000008	0.7707891
10	20	10	0.0000007	0.3	0.0000010	0.0000010	0.5456765
11	30	10	0.0000003	0.3	0.0000002	0.0000009	0.9659689
12	30	10	0.0000005	0.3	0.0000002	0.0000009	0.9542911
13	30	15	0.0000002	0.3	0.0000002	0.0000009	0.9813440
14	30	10	0.0000002	0.3	0.0000001	0.0000001	0.9533995
15	30	10	0.0000003	0.3	0.0000002	0.0000009	0.9863629
16	30	10	0.0000003	0.3	0.0000002	0.0000010	0.9702858
17	30	10	0.0000003	0.3	0.0000010	0.0000010	0.9849021
18	20	15	0.0000004	0.3	0.0000010	0.0000008	0.9729872
19	**30**	**10**	**0.0000000**	**0.3**	**0.0000002**	**0.0000010**	**0.5353839**
20	**30**	**10**	**0.0000003**	**0.3**	**0.0000008**	**0.0000010**	**0.5662247**
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**	**Sample Size**	**Execution Time**
1	1	25,000	0.0216957	0.00000008	20	3,026.35	4,003.11
2	2	50,000	0.0174433	0.00000096	20	2,762.46	11,470.11
3	1	25,000	0.0159755	0.00000073	20	3,029.18	3,987.64
4	1	50,000	0.0131232	0.00000049	20	2,703.32	15,910.83
5	2	50,000	0.0103373	0.00000061	20	2,766.39	11,459.90
6	2	25,000	0.0195154	0.00000038	20	2,932.12	5,156.96
7	2	25,000	0.0220333	0.00000028	20	2,928.78	5,177.00
8	2	50,000	0.0187521	0.00000057	20	2,620.40	22,859.85
9	1	50,000	0.0241063	0.00000082	20	2,861.24	8,281.68
10	1	25,000	0.0122188	0.00000019	20	3,027.27	3,872.97
11	2	50,000	0.0157505	0.00000043	20	2,621.62	22,758.15
12	1	50,000	0.0165162	0.00000033	20	2,704.42	15,941.84
13	1	25,000	0.0155316	0.00000041	20	3,015.87	4,023.10
14	2	25,000	0.0145631	0.00000047	20	2,759.56	10,261.55
15	1	25,000	0.0177011	0.00000055	20	2,856.91	7,632.94
16	2	50,000	0.0158378	0.00000001	20	2,623.53	22,497.67
17	2	50,000	0.0113557	0.00000013	20	2,619.06	22,663.52
18	2	50,000	0.0166339	0.00000014	20	2,763.13	11,136.54
19	2	50,000	0.0126639	0.00000001	**20**	2,619.99	22,400.28
20	1	50,000	0.0121512	0.00000007	**20**	**2,701.10**	15,818.99
					Minimum	**2,619.06**	
						Total	247,314.63
Kiva Loans							
Fine-tuning iteration No.	**Iterations, I**	**SAA Frequency**	**Mutation Chance**	**Elitism Rate**	**Add Strata Factor**	**Temperature, T**	**Decrement Constant, DC**
1	90	20	0.0000004	0.2	0.0000008	0.0000001	0.6933468
2	50	20	0.0000010	0.3	0.0000002	0.0000004	0.5736595
3	90	20	0.0000003	0.3	0.0000005	0.0000009	0.9229554
4	100	10	0.0000005	0.2	0.0000008	0.0000002	0.7242638
5	60	20	0.0000002	0.2	0.0000005	0.0000007	0.8150428
6	50	10	0.0000000	0.2	0.0000000	0.0000006	0.6266293
7	70	10	0.0000006	0.2	0.0000003	0.0000003	0.8724350
8	80	10	0.0000006	0.3	0.0000002	0.0000002	0.9842787
9	100	20	0.0000009	0.3	0.0000006	0.0000008	0.7707891

10	70	10	0.0000007	0.3	0.0000010	0.0000010	0.5456765
11	80	10	0.0000004	0.3	0.0000008	0.0000001	0.9727142
12	80	20	0.0000003	0.3	0.0000008	0.0000000	0.9796571
13	90	10	0.0000004	0.3	0.0000008	0.0000002	0.9820326
14	90	20	0.0000004	0.3	0.0000004	0.0000002	0.9887106
15	90	10	0.0000004	0.3	0.0000001	0.0000001	0.9844605
16	90	20	0.0000003	0.3	0.0000000	0.0000001	0.9838737
17	90	10	0.0000004	0.3	0.0000007	0.0000001	0.9778088
18	90	10	0.0000001	0.3	0.0000005	0.0000001	0.9318817
19	**90**	**10**	**0.0000004**	**0.3**	**0.0000004**	**0.0000001**	**0.9233493**
20	**90**	**10**	**0.0000004**	**0.3**	**0.0000008**	**0.0000002**	**0.7065010**
Fine-tuning iteration No.	**Max number of sequences, maxit**	**Length of sequence, J**	**% of L for maximum q value, Lmax%**	**Probability of New Stratum, P(H + 1)**	**Population Size, N_p**	**Sample Size**	**Execution Time**
1	1	10,000	0.0216957	0.0000001	20	1,806.82	1,150.00
2	2	20,000	0.0174433	0.0000010	20	1,747.71	2,157.59
3	1	10,000	0.0159755	0.0000007	20	1,797.86	1,120.60
4	1	20,000	0.0131232	0.0000005	20	1,613.25	6,316.39
5	2	20,000	0.0103373	0.0000006	20	1,719.17	2,172.81
6	2	10,000	0.0195154	0.0000004	20	1,753.86	1,463.30
7	2	10,000	0.0220333	0.0000003	20	1,681.44	2,239.25
8	2	20,000	0.0187521	0.0000006	20	1,597.02	7,702.03
9	1	20,000	0.0241063	0.0000008	20	1,647.24	2,869.39
10	1	10,000	0.0122188	0.0000002	20	1,683.68	1,517.16
11	1	20,000	0.0183654	0.0000007	20	1,606.42	4,810.26
12	1	20,000	0.0187089	0.0000006	20	1,739.74	2,112.80
13	2	20,000	0.0137673	0.0000001	20	1,592.34	8,885.01
14	2	20,000	0.0138855	0.0000001	20	1,629.67	4,429.31
15	2	20,000	0.0156086	0.0000008	20	1,576.80	8,791.12
16	2	20,000	0.0152018	0.0000008	20	1,622.36	4,377.68
17	1	20,000	0.0197092	0.0000001	20	1,600.29	5,549.41
18	2	20,000	0.0156407	0.0000009	20	1,607.65	9,208.90
19	2	10,000	0.0166562	0.0000008	**20**	1,651.75	3,107.61
20	2	20,000	0.0150126	0.0000001	**20**	**1,590.36**	8,971.35
					Minimum	**1,576.80**	
						Total	88,951.97

References

Aaronson, S. (2014). My conversation with 'eugene goostman,'the chatbot that's all over the news for allegedly passing the turing test. *The Blog of Scott Aaronson*.

Administration, E. . S. (2015 (accessed February 15, 2017)). *The Value of the American Community Survey: Smart Government, Competitive Businesses, and Informed Citizens*. `http://www.esa.gov/sites/default/files/the-value-of-the-acs.pdf`.

Ahmed, I. and A. Aziz (2010). Dynamic approach for data scrubbing process. *International Journal on Computer Science and Engineering 2*(02), 416–423.

Agustín-Blas, L.E. , Salcedo-Sanz, S. , Vidales, P. , Urueta, G. , Portilla-Figueras, J.A. (2011). Near optimal citywide WiFi network deployment using a hybrid grouping genetic algorithm. *Expert Systems with Applications 38*(8) 9543–9556.

Allen, M. (2017). *The SAGE encyclopedia of communication research methods*. SAGE publications.

Anderson, E. (1935). The irises of the gaspe peninsula. *Bulletin of the American Iris society 59*, 2–5.

Arthur, D. and S. Vassilvitskii. k-means++: The advantages of careful seeding,(2006). In *SODA'07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics*, pp. 1027–1035.

Asan, U. and S. Ercan (2012). An introduction to self-organizing maps. In *Computational Intelligence Systems in Industrial Engineering*, pp. 295–315. Springer.

ACS (2022). Artificial intelligence ethics committee. Australian Computer Society Accessed 21 July 2022. `https://www.acs.org.au/governance/ai-ethics-committee.html`.

Äyrämö, S. and T. Kärkkäinen (2006). Introduction to partitioning-based clustering methods with a robust example. *Reports of the Department of Mathematical Information Technology. Series C, Software engineering and computational intelligence* (1/2006).

Bação, F., V. Lobo, and M. Painho (2005). Self-organizing maps as substitutes for k-means clustering. In *International Conference on Computational Science*, pp. 476–483. Springer.

Baillargeon, S. and L.-P. Rivest (2009). A general algorithm for univariate stratification. *International Statistical Review 77*(3), 331–344.

Baillargeon, S. and L.-P. Rivest (2011). The construction of stratified designs in r with the package stratification. *Survey Methodology 37*(1), 53–65.

Ballin, M. and G. Barcaroli (2013). Joint determination of optimal stratification and sample allocation using genetic algorithm. *Survey Methodology 39*(2), 369–393.

Ballin, M., G. Barcaroli, E. Catanese, M. D'Orazio, et al. (2016). Stratification in business and agriculture surveys with r. *Romanian Statistical Review 64*(2), 43–58.

Ballin, M., G. Barcaroli, M. Masselli, and M. Scarnò (2018). Redesign sample for land use/cover area frame survey (lucas) 2018. *Eurostat: statistical working papers. doi 10*, 132365.

Ballin, M. and G. Barcaroli (2020). R package samplingstrata: new developments and extension to spatial sampling. *arXiv preprint arXiv:2004.09366*.

Ballin, M. and G. Barcaroli (2020). Optimization of sampling strata with the SamplingStrata package. Accessed 3 May 2021. https://barcaroli.github.io/SamplingStrata/articles/SamplingStrata.html

Baluja, S. (1994). Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science.

Bankier, M. D. (1988). Power allocations: determining sample sizes for subnational areas. *The American Statistician 42*(3), 174–177.

Barcaroli, G. (2014). SamplingStrata: An R package for the optimization of stratified sampling. *Journal of Statistical Software 61*(4), 1–24.

Barcaroli, G. (2018). Optimization of sampling strata with the samplingstrata package.

Barcaroli, G., G. Ilardi, A. Neri, and T. Tuoto. (2021) Optimal sampling design for household finance surveys using administrative income data.

Benedetti, R., G. Espa, and G. Lafratta (2008). A tree-based approach to forming strata in multipurpose business surveys. *Survey Methodology 34*(2), 195–203.

Bergsma, W. (2013). A bias-correction for cramér's v and tschuprow's t. *Journal of the Korean Statistical Society 42*(3), 323–328.

Bethel, J. W. (1985). *An optimum allocation algorithm for multivariate surveys*. American Statistical Proceedings of the Survey Research Methods, Section, 209-212.

Bethel, J. (1989). Sample allocation in multivariate surveys. *Survey methodology 15*(1), 47-57.

Bethlehem, J. G. and W. J. Keller (1987). Linear weighting of sample survey data. *Journal of official Statistics 3*(2), 141–153.

Bethlehem, J. (2009). *Applied survey methods: A statistical perspective*, Volume 558. John Wiley & Sons.

Bezdek, J. C. (1981). Objective function clustering. In *Pattern recognition with fuzzy objective function algorithms*, pp. 43–93. Springer.

Bezdek, J. C., R. Ehrlich, and W. Full (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences 10*(2-3), 191–203.

Bischla, B., J. Richterb, J. Bossekc, D. Hornb, J. Thomasa, and M. Langb (2017). mlrmbo: A modular framework for model-based optimization of expensive black-box functions. *Gradient Boosting in Automatic Machine Learning: Feature Selection and Hyperparameter Optimization*, 36.

Bonomi, E. and J.-L. Lutton (1984). The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. *SIAM review 26*(4), 551–568.

Bonomi, E. and J.-L. Lutton (1986). The asymptotic behaviour of quadratic sum assignment problems: A statistical mechanics approach. *European Journal of Operational Research 26*(2), 295–300.

Bostrom, N. (2009). Superintelligence. answer to the 2009 edge question:"what will change everything?".

Bowley, A. L. (1906). Address to the economic science and statistics section of the british association for the advancement of science, york, 1906. *Journal of the Royal Statistical Society 69*(3), 540–558.

Boyd, S., S. P. Boyd, and L. Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Briggs, J., V. Duoba, and S. N. Zealand (2000). *STRAT2D: Optimal bi-variate stratification system*. Statistics New Zealand.

Brochu, E., V. M. Cora, and N. De Freitas (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Brown, E.C. and M. Vroblefski. A grouping genetic algorithm for the microcell sectorization problem. *Engineering Applications of Artificial Intelligence 17*(6), 589–598.

Bühler, W. and T. Deutler (1975). Optimal stratification and grouping by dynamic programming. *Metrika 22*(1), 161–175.

Bureau, US Census (2013 (accessed February 15, 2017)). *American Community Survey Information Guide*. `http://www.census.gov/content/`

dam/Census/programs-surveys/acs/about/ACS_Information_Guide.pdf.

Bureau, US Census (2016 (accessed February 15, 2017)). *2015 ACS PUMS DATA DICTIONARY*. http://www2.census.gov/programs-surveys/acs/tech_docs/pums/data_dict/PUMSDataDict15.pdf.

Bureau,US Census (2014 (accessed February 15, 2017)b). *American Community Survey Design and Methodology*. http://www2.census.gov/programs-surveys/acs/methodology/design_and_methodology/acs_design_methodology_report_2014.pdf.

Caiola, G. and J. P. Reiter (2010). Random forests for generating partially synthetic, categorical data. *Trans. Data Priv. 3*(1), 27–42.

Canul-Reich, J., J. Hernández-Torruco, J. Frausto-Solís, and J. J. Méndez Castillo (2015). Finding relevant features for identifying subtypes of guillain-barré syndrome using quenching simulated annealing and partitions around medoids. *International Journal of Combinatorial Optimization Problems and Informatics 6*(2), 11–27.

Černỳ, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications 45*(1), 41–51.

Chadha, A. (2020). Expectation maximization algorithms. *Distilled Notes for Stanford CS229: Machine Learning*. https://aman.ai.

Chromy, J. R. (1987). Design optimization with multiple objectives. *Proceedings of the Survey Research Methods Section*.

Chu, X., I. F. Ilyas, S. Krishnan, and J. Wang (2016). Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pp. 2201–2206.

Cochran, W. G. (1953). Sampling techniques. *New York: Wiley*.

Cochran, W. G. (1977). Sampling techniques 3. pp. 89.

Collins, N., R. Eglese, and B. Golden (1988). Simulated annealing–an annotated bibliography. *American Journal of Mathematical and Management Sciences 8*(3-4), 209–307.

Corporation, M. and S. Weston (2020). *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*. R package version 1.0.16.

Cortez, P. (2014). *Modern optimization with R*. Springer.

Cramér, H. (1999). *Mathematical methods of statistics*, Volume 43. Princeton university press.

CSO (2021 (accessed July 16, 2021)). *Quarterly Survey In Construction Quality Report 2021*. `https://www.cso.ie/en/media/csoie/methods/productioninbuildingandconstructionindex/QSC_Quality_Report_2021.pdf`

Dalenius, T. (1950). The problem of optimum stratification. *Scandinavian Actuarial Journal 1950*(3-4), 203–213.

Dalenius, T. and J. L. Hodges Jr (1959). Minimum variance stratification. *Journal of the American Statistical Association 54*(285), 88–101.

Dang, D.-C., P. K. Lehre, and P. T. H. Nguyen (2019). Level-based analysis of the univariate marginal distribution algorithm. *Algorithmica 81*(2), 668–702.

Danish, F., S. Rizvi, M. K. Sharma, and M. I. Jeelani (2017). Optimum stratification using mathematical programming approach: A review. *Journal of Statistics Applications & Probability Letters 4*(3), 123–129.

Daszykowski, M., B. Walczak, and D. L. Massart (2002). On the optimal partitioning of data with k-means, growing k-means, neural gas, and growing neural gas. *Journal of chemical information and computer sciences 42*(6), 1378–138

Day, C. (2006). Application of an evolutionary algorithm to multivariate optimal allocation in stratified sample designs. In *2006 Proceedings of the American Statistical Association*. Citeseer.

Day, C. D. (2010). A multi-objective evolutionary algorithm for multivariate optimal allocation. *Section on Survey Research Methods-JSM 2010*, 3351–3358.

De Bonet, J. S., C. L. Isbell, P. Viola, et al. (1997). Mimic: Finding optima by estimating probability densities. *Advances in neural information processing systems*, 424–430.

De Lit, P. , Falkenauer, E. , Delchambre A. Grouping genetic algorithms: an efficient method to solve the cell formation problem.

De Meo, M. and M. M. De Meo (2009). Package 'bethel'.

de Moura Brito, J. A., G. S. Semaan, P. L. do Nascimento Silva, and N. Maculan (2013). An integer programming formulation applied to optimum allocation in multivariate stratified sampling. *arXiv e-prints*, arXiv–1309.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological) 39*(1), 1–22.

Department of Enterprise, T. and Employment (2021). *AI – HERE FOR GOOD A National Artificial Intelligence Strategy for Ireland*. Dublin: Rialtas na hÉireann.

Durmus, M. (2022). *Taxonomy of the most commonly used Machine Learning Algorithms*. Germany: Kindle.

Dias, M. L. D. and A. R. R. Neto (2017). Training soft margin support vector machines by simulated annealing: A dual approach. *Expert Systems with Applications 87*, 157–169.

Díaz-García, J. A. and L. U. Cortez (2008). Multi-objective optimisation for optimum allocation in multivariate stratified sampling. *Survey Methodology 34*(2), 215–222.

Dimitriadou, E., K. Hornik, and M. K. Hornik (2020). Package 'cclust'.

Dirgová Luptáková, I., M. Šimon, L. Huraj, and J. Pospíchal (2016). Neural gas clustering adapted for given size of clusters. *Mathematical Problems in Engineering 2016*.

Division, S. C. S. S. M. (2003). *Survey methods and practices*. Statistics Canada.

Doerr, B. and M. S. Krejca (2020). The univariate marginal distribution algorithm copes well with deception and epistasis. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pp. 17–18.

Doshi-Velez, F. and B. Kim (2017). Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Drechsler, J. and J. P. Reiter (2011). An empirical evaluation of easily implemented, nonparametric methods for generating synthetic datasets. *Computational Statistics & Data Analysis 55*(12), 3232–3243.

Drineas, P., A. Frieze, R. Kannan, S. Vempala, and V. Vinay (2004). Clustering large graphs via the singular value decomposition. *Machine learning 56*(1), 9–33.

Eddelbuettel,E. (2013). Seamless R and C++ Integration with Rcpp *ISBN 978-1-4614-6867-7* 10.1007/978-1-4614-6868-4

Edgeworth, F. Y. (1883). Xlii. the law of error. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 16*(100), 300–309.

Engel, M., G. Marsden, and S. W. Pollock (1971). Child work and social class. *Psychiatry 34*(2), 140–155.

Etxeberria, R. (1999). Global optimization using bayesian networks. In *Proc. 2nd Symposium on Artificial Intelligence (CIMAF-99)*.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc.

Feurer, M. and F. Hutter (2019). Hyperparameter optimization. In *Automated machine learning*, pp. 3–33. Springer, Cham.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics 7*(2), 179–188.

Forgy, E. W. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *biometrics 21*, 768–769.

Foundation, O. K. (2015). Open definition. Accessed 14 July 2022.

Franklin, S. and C. Walker (2003). Survey methods and practices. statistics canada. *Social Survey Methods Division, Ottawa*.

Friedman, J., T. Hastie, R. Tibshirani, et al. (2001). *The elements of statistical learning*, Volume 1. Springer series in statistics New York.

Galinier, P. , Hao, J. K.. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization* 3(4):379–397, 1999.

Gao, Z., Y. Feng, and K. Xing (2020). A hybrid estimation-of-distribution algorithm for scheduling flexible job shop with limited buffers based on petri nets. *IEEE Access 8*, 165396–165408.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*, Volume 1. Springer.

Gonzalez, J. M. and J. L. Eltinge (2010). Optimal survey design: A review. *Section on Survey*

Gonzalez-Fernandez, Y. and M. Soto (2012). copulaedas: An r package for estimation of distribution algorithms based on copulas. arXiv preprint arXiv:1209.5429. *Research Methods–JSM*.

Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical computer science 38*, 293–306.

Graham, R. L., D. E. Knuth, O. Patashnik, and S. Liu (1989). Concrete mathematics: a foundation for computer science. *Computers in Physics 3*(5), 373,566.

Grover, L. K. (1987). Standard cell placement using simulated sintering. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 56–59.

Groves, R. M., F. J. Fowler Jr, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau (2011). *Survey methodology*, Volume 561. John Wiley & Sons.

Gunning, P. and J. M. Horgan (2004). A new algorithm for the construction of stratum boundaries in skewed populations. *Survey Methodology 30*(2), 159–166.

Han, J., J. Pei, and M. Kamber (2011). *Data mining: concepts and techniques*. Elsevier.

Han, M. (2013). *Heuristic optimization of the p-median problem and population re-distribution*. Ph. D. thesis, Dalarna University.

Handley, E. and P. e. y. n. here (2022, Jun). The synthetic data approach: The new unlimited data plan for ai models.

Hao, J.-K. and C. Solnon (2020). Meta-heuristics and artificial intelligence. In *A Guided Tour of Artificial Intelligence Research*, pp. 27–52. Springer.

Harik, G. R., F. G. Lobo, and D. E. Goldberg (1999). The compact genetic algorithm. *IEEE transactions on evolutionary computation 3*(4), 287–297.

Harik, G., E. Cantú-Paz, D. E. Goldberg, and B. L. Miller (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary computation 7*(3), 231–253.

Harnad, S. (1991). Other bodies, other minds: A machine incarnation of an old philosophical problem. *Minds and Machines 1*(1), 43–54.

Hartigan, J. A. and M. A. Wong (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*(1), 100–108.

Kashan, A. H., M. H. Kashan, and S. Karimiyan (2013). A particle swarm optimizer for grouping problems. *Information Sciences 252*, 81–95.

Hauschild, M. and M. Pelikan (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and evolutionary computation 1*(3), 111–128.

Hidiroglou, M. A. (1986). The construction of a self-representing stratum of large units in survey design. *The American Statistician 40*(1), 27–31.

Honkela, T. (1997). *Self-organizing maps in natural language processing*. Ph. D. thesis, Citeseer.

Hooke, R. and T. A. Jeeves (1961). "direct search"solution of numerical and statistical problems. *Journal of the ACM (JACM) 8*(2), 212–229.

Hoos, H. H. and T. Stützle (2004). *Stochastic local search: Foundations and applications*. Elsevier.

Huddleston, H., P. Claypool, and R. Hocking (1970). Optimal sample allocation to strata using convex programming. *Applied Statistics*, 273–278.

Hung, C. , Sumichrast, R. T., Brown, E.C. CPGEA: a grouping genetic algorithm for material cutting plan generation. *Computers & Industrial Engineering 44*(4)m 651–672.

Hutter, F., H. H. Hoos, and K. Leyton-Brown (2010). Sequential model-based optimization for general algorithm configuration (extended version). *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*.

Isaki, C. T. and W. A. Fuller (1982). Survey design under the regression superpopulation model. *Journal of the American Statistical Association 77*(377), 89–96.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters 31*(8).

James, T., E. Brown, and C. T. Ragsdale (2010). Grouping genetic algorithm for the blockmodel problem. *IEEE Transactions on Evolutionary Computation 14*(1), 103–111.

John, H. (1975). Holland, adaptation in natural and artificial systems, 1992. *Ann Arbor, MI: University of Michigan Press*.

Johnson, D. S., C. R. Aragon, L. A. McGeoch, and C. Schevon (1989). Optimization

by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations research 37*(6), 865–892.

Jones, D. R., M. Schonlau, and W. J. Welch (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization 13*(4), 455–492.

Kaufman, L. and P. J. Rousseeuw (2008). Clustering large applications (program clara). *Finding groups in data: an introduction to cluster analysis*, 126–163.

Kaufman, L. and P. J. Rousseeuw (2009). *Finding groups in data: an introduction to cluster analysis*, Volume 344. John Wiley & Sons.

Keskintürk, T. and Ş. Er (2007). A genetic algorithm approach to determine stratum boundaries and sample sizes of each stratum in stratified sampling. *Computational Statistics & Data Analysis 52*(1), 53–67.

Khan, M. G. M., N. Nand, and N. Ahmad (2008). Determining the optimum strata boundary points using dynamic programming. *Survey methodology 34*(2), 205–214.

Khan, M. G., T. Maiti, and M. Ahsan (2010). An optimal multivariate stratified sampling design using auxiliary information: an integer solution using goal programming approach. *Journal of official Statistics 26*(4), 695.

Kiaër, A. (1899). Sur les méthodes représentatives ou typologiques appliquées à la statistique. *Bulletin de l'Institut International de Statistique 11*(1), 180–185.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *science 220*(4598), 671–680.

Kish, L. et al. (1965). *Survey sampling*. Chichester.

Kish, L. (1976). Optima and proxima in linear sample designs. *Journal of the Royal Statistical Society. Series A (General)*, 80–95.

Kish, L. and D. W. Anderson (1978). Multivariate and multipurpose stratification. *Journal of the American statistical Association 73*(361), 24–34.

Kiva (2018, Mar). Data science for good: Kiva crowdfunding.

Klawonn, F. (2004). Fuzzy clustering: Insights and a new approach. *Mathware & soft computing. 2004 Vol. 11 Núm. 3*.

K-Means and SOM: Introduction to Popular Clustering Algorithms - DZone AI. https://dzone.com/articles/k-means-and-som-gentle-introduction-to-worlds-most. Accessed 25th May 2021.

Kodinariya, T. M. and P. R. Makwana (2013). Review on determining number of cluster in k-means clustering. *International Journal 1*(6), 90–95.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE 78*(9), 1464–1480.

Kollat, J. B., P. M. Reed, and J. R. Kasprzyk (2008). A new epsilon-dominance hierarchical bayesian optimization algorithm for large multiobjective monitoring network design problems. *Advances in Water Resources 31*(5), 828–845.

Korf, Richard E (1999). *Artificial intelligence search algorithms*. Citeseer.

Kozak, M. (2004, 05). Optimal stratification using random search method in agricultural surveys. *Statistics in Transition 6*, 797–806.

Kozak, M. and S. Singh (2007). Application of evolutionary algorithms in multivariate stratification.

Kozak, M., M. R. Verma, and A. Zielinski (2007). Modern approach to optimum stratification: Review and perspectives. *Statistics in Transition-new series 8*(2), 223–250.

Kozak, M., A. Zieliński, and S. Singh (2008). Stratified two-stage sampling in domains: Sample allocation between domains, strata, and sampling stages. *Statistics & Probability Letters 78*(8), 970–974.

Kozak, M. and H. Y. Wang (2010). On stochastic optimization in sample allocation among strata. *Metron 68*(1), 95–103.

Larrañaga, P. and J. A. Lozano (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation*, Volume 2. Springer Science & Business Media.

Lavallée, P. and M. A. Hidiroglou (1988). On the stratification of skewed populations. *Survey methodology 14*(1), 33–43.

Lednicki, B. and R. Wieczorkowski (2003). Optimal stratification and sample allocation between subpopulations and strata. *Statistics in Transition 6*(2), 287–305.

Li, H. (2014). Smile. `https://haifengl.github.io`. Accessed 08th June 2021.

Lima, C. F., M. Pelikan, F. G. Lobo, and D. E. Goldberg (2009). Loopy substructural local search for the bayesian optimization algorithm. In *International Workshop on Engineering Stochastic Local Search Algorithms*, pp. 61–75. Springer.

Lima, C. F., F. G. Lobo, M. Pelikan, and D. E. Goldberg (2011). Model accuracy in the bayesian optimization algorithm. *Soft Computing 15*(7), 1351–1371.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal 44*(10), 2245–2269.

Lin, S. and B. W. Kernighan (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research 21*(2), 498–516.

Lisic, J., H. Sang, Z. Zhu, and S. Zimmer (2018). Optimal stratification and allocation for the june agricultural survey. *Journal of Official Statistics 34*(1), 121–148.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory 28*(2), 129–137.

Lohr, S. L. (2019). *Sampling: design and analysis*. Chapman and Hall/CRC.

Lu, W. and R. R. Sitter (2002). Multi-way stratification by linear programming made practical. *Survey Methodology 28*(2), 199–208.

Luke, S. (2013). *Essentials of Metaheuristics* (second ed.). Lulu. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

Lutton, J.-L. and E. Bonomi (1986). Simulated annealing algorithm for the minimum weighted perfect euclidean matching problem. *RAIRO-Operations Research-Recherche Opérationnelle 20*(3), 177–197.

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

Marcinkevičs, R. and J. E. Vogt (2020). Interpretability and explainability: A machine learning zoo mini-tour. *arXiv preprint arXiv:2012.01805*.

Martinetz, T., K. Schulten, et al. (1991). A "neural-gas" network learns topologies.

Martinetz, T. M., S. G. Berkovich, and K. J. Schulten (1993). 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE transactions on neural networks 4*(4), 558–569.

McKay, M. D., R. J. Beckman, and W. J. Conover (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics 42*(1), 55–61.

Meilă, M. (2006). The uniqueness of a good optimum for k-means. In *Proceedings of the 23rd international conference on Machine learning*, pp. 625–632.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics 21*(6), 1087–1092.

Meyer, D., E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, C.-C. Chang, C.-C. Lin, and M. D. Meyer (2019). Package 'e1071'. *The R Journal*.

Michalewicz, Z. and D. B. Fogel (2013). *How to solve it: modern heuristics*. Springer Science & Business Media.

Michie, D. (1968). "memo" functions and machine learning. *Nature 218*(5136), 19–22.

Microsoft and S. Weston (2020). *foreach: Provides Foreach Looping Construct*. R package version 1.5.1.

Morissette, L. and S. Chartier (2013). The k-means clustering technique: General considerations and implementation in mathematica. *Tutorials in Quantitative Methods for Psychology 9*(1), 15–24.

Mühlenbein, H. and G. Paaß (1996). From recombination of genes to the estimation of distributions i. binary parameters. In *International conference on parallel problem solving from nature*, pp. 178–187. Springer.

Mühlenbein, H. and T. Mahnig (1999). Fda-a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary computation 7*(4), 353–376.

Mühlenbein, H. and T. Mahnig (2001). Evolutionary algorithms: From recombination to search distributions. In *Theoretical Aspects of Evolutionary Computing*, pp. 135–173. Springer.

Nahar, S., S. Sahni, and E. Shragowitz (1985). Experiments with simulated annealing. In *22nd ACM/IEEE Design Automation Conference*, pp. 748–752. IEEE.

Nayak, J., B. Naik, and H. Behera (2015). Fuzzy c-means (fcm) clustering algorithm: a decade review from 2000 to 2014. *Computational intelligence in data mining-volume 2*, 133–149.

Nelder, J. A. and R. Mead (1965). A simplex method for function minimization. *The computer journal 7*(4), 308–313.

Neyman, J. (1934). On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society 97*(4), 558–625.

Nikolaev, A. G. and S. H. Jacobson (2010). Simulated annealing. In *Handbook of metaheuristics*, pp. 1–39. Springer.

Nowok, B., G. M. Raab, and C. Dibben (2016). synthpop: Bespoke creation of synthetic data in r. *Journal of statistical software 74*, 1–26.

Paul, T. K. and H. Iba (2002). Linear and combinatorial optimizations by estimation of distribution algorithms. In *9th MPS symposium on Evolutionary Computation, IPSJ, Japan*, Volume 12.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pearson, K. (1900). X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 50*(302), 157–175.

Pearson, K. (1907). *On further methods of determining correlation*, Volume 16. Dulau and Company.

Pelikan, M. and Goldberg, D. E. Genetic Algorithms, Clustering, and the Breaking of Symmetry. *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature,* 2000.

Pelikan, M., D. E. Goldberg, E. Cantú-Paz, et al. (1999). Boa: The bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, Volume 1, pp. 525–532. Citeseer.

Pelikan, M., D. E. Goldberg, and S. Tsutsui (2003). Hierarchical bayesian optimization algorithm: toward a new generation of evolutionary algorithms. In *SICE 2003 Annual Conference (IEEE Cat. No. 03TH8734)*, Volume 3, pp. 2738–2743. IEEE.

Pena, M. and C. Fyfe (2006). The topographic neural gas. In *Intelligent Data Engineering and Automated Learning–IDEAL 2006: 7th International*

Conference, Burgos, Spain, September 20-23, 2006. Proceedings 7, pp. 241–248. Springer.

Pérez-Ortega, J., N. N. Almanza-Ortega, A. Vega-Villalobos, R. Pazos-Rangel, C. Zavala-Díaz, and A. Martínez-Rebollar (2019). The k-means algorithm evolution. In *Introduction to Data Science and Machine Learning*. IntechOpen.

Petrovski, A., S. Shakya, and J. McCall (2006). Optimising cancer chemotherapy using an estimation of distribution algorithm and genetic algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 413–418.

Pham, D. T., S. S. Dimov, and C. D. Nguyen (2005). Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 219*(1), 103–119.

Prügel-Bennett, A. (2004). Symmetry Breaking in Population-Based Optimization. *IEEE Transactions on Evolutionary Computation 8*(1):63–79.

R Core Team (2015). *R* A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. `https://www.R-project.org/`.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

Ramos-Figueroa, O., M. Quiroz-Castellanos, E. Mezura-Montes, and O. Schütze (2020). Metaheuristics to solve grouping problems: A review and a case study. *Swarm and Evolutionary Computation 53*, 100643.

Rao, T. (1977). Optimum allocation of sample size and prior distributions: a review. *International Statistical Review/Revue Internationale de Statistique*, 173–179.

Reddy, K. G., M. G. Khan, and D. Rao (2016). A procedure for computing optimal stratum boundaries and sample sizes for multivariate surveys. *JSW 11*(8), 816–832.

Rice, J. A. (2006). *Mathematical statistics and data analysis*. Cengage Learning.

Rizvi, S., J. Gupta, and R. Singh (2000). Approximately optimum stratification for two study variables using auxiliary information. *Jour. Ind. Soc. Ag. Statistics 53*(3), 287–298.

Robles, V., J. M. Peña, P. Larrañaga, M. S. Pérez, and V. Herves (2006). Ga-

eda: A new hybrid cooperative search evolutionary algorithm. In *Towards a New Evolutionary Computation*, pp. 187–219. Springer.

Rose, J. S., D. Blythe, W. Snelgrove, and Z. Vranesic (1986). *Fast, high quality VLSI placement on an MIMD multiprocessor*. Ph. D. thesis, University of Toronto, Computer Systems' Research Institute.

Rose, J. S., W. M. Snelgrove, and Z. G. Vranesic (1988). Parallel standard cell placement algorithms with quality equivalent to simulated annealing. *IEEE transactions on computer-aided design of integrated circuits and systems 7*(3), 387–396.

Ross, P., D. Corne, and H.-L. Fang (1994). Improving evolutionary timetabling with delta evaluation and directed mutation. In *International Conference on Parallel Problem Solving from Nature*, pp. 556–565. Springer.
Gazi university journal of science 24(2), 249–262.

Ruggles, S., K. Genadek, R. Goeken, J. Grover, and M. Sobek (2017). Integrated public use microdata series: Version 7.0 [data set]. minneapolis: University of minnesota, 2017.

Russell, S. J. and P. Norvig (2010). Artificial intelligence-a modern approach, third int. edition.

ŞAH'IN, S. T. and S. SAH'IN (2011). Determination of sample size selecting from strata under nonlinear cost constraint by using goal programming and kuhn-tucker methods. *Gazi university journal of science 24*(2), 249–262.

Salamon, P., P. Sibani, and R. Frost (2002). *Facts, conjectures, and improvements for simulated annealing*. SIAM.

Samuel, A. L. (1959). Machine learning. *The Technology Review 62*(1), 42–45.

Särndal, C.-E., B. Swensson, and J. Wretman (2003). *Model assisted survey sampling*. Springer Science & Business Media.

Schubert, E. and P. J. Rousseeuw (2019). Faster k-medoids clustering: improving the pam, clara, and clarans algorithms. In *International Conference on Similarity Search and Applications*, pp. 171–187. Springer.

Schneeberger, H. and J. Pöllot (1985). Optimum stratification with two variates. *Statistische Hefte 26*(1), 97.

Scrucca, L., M. Fop, T. B. Murphy, and A. E. Raftery (2016). mclust 5: clustering,

classification and density estimation using Gaussian finite mixture models. *The R Journal 8*(1), 289–317.

Searle, J. R. (1990). Is the brain's mind a computer program? *Scientific American 262*(1), 25–31.

Sechen, C. and A. Sangiovanni-Vincentelli (1985). The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits 20*(2), 510–522.

Sethi, V. (1963). A note on optimum stratification of populations for estimating the population means. *Australian Journal of Statistics 5*(1), 20–33.

Shah, R. and P. Reed (2011). Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems. *European Journal of Operational Research 211*(3), 466–479.

Shieber, S. M. (1994). Lessons from a restricted turing test. *arXiv preprint cmp-lg/9404002*.

Silviu, B. (2001). Fuzzy clustering. *Babes-Bolyai University*.

Singh, R. (1971). Approximately optimum stratification on the auxiliary variable. *Journal of the American Statistical Association 66*(336), 829–833.

Singh, K. (2021, May). Synthetic data − key benefits, types, generation methods, and challenges!

Skinner, C. J., D. Holmes, and D. Holt (1994). Multiple frame sampling for multivariate stratification. *International Statistical Review/Revue Internationale de Statistique*, 333–347.

Smith, T. (1976). The foundations of survey sampling: a review. *Journal of the Royal Statistical Society: Series A (General) 139*(2), 183–195.

Sörensen, K. and F. Glover (2013). Metaheuristics. *Encyclopedia of operations research and management science 62*, 960–970.

Späth, H. (1980). Cluster analysis algorithms for data reduction and classification of objects.

Spearman, C. (1904). Reprinted: The proof and measurement of association between two things (2010). *International Journal of Epidemiology 39*(5), 1137–1150.

Spearman, C. (1987). The proof and measurement of association between two things. *The American journal of psychology 100*(3/4), 441–471.

Stokes, L. and J. Plummer (2004). Using spreadsheet solvers in sample design. *Computational statistics & data analysis 44*(3), 527–546.

Strong, A. (2016). Applications of artificial intelligence & associated technologies. *Science [ETEBMS-2016] 5*(6).

Sweet, E. M. and R. S. Sigman (1995). Evaluation of model-assisted procedures for stratifying skewed populations using auxiliary data. In *Proceedings of the Section on Survey Research Methods*, Volume 1, pp. 491–496.

Takeang, C. and A. Aurasopon (2019). Multiple of hybrid lambda iteration and simulated annealing algorithm to solve economic dispatch problem with ramp rate limit and prohibited operating zones. *Journal of Electrical Engineering & Technology 14*(1), 111–120.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, Volume 74. John Wiley & Sons.

Tang, M. and R. Lau (2005). A hybrid estimation of distribution algorithm for the minimal switching graph problem. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Volume 1, pp. 708–713.

Techopedia (2022, Apr). What is weak artificial intelligence (weak ai)? - definition from techopedia.

Tepping, B. J., W. N. Hurwitz, and W. E. Deming (1943). On the efficiency of deep stratification in block sampling. *Journal of the American Statistical Association 38*(221), 93–100.

Theobald, O. (2017). *Machine learning for absolute beginners: a plain English introduction*, Volume 157. Scatterplot press.

Thomsen, I. (1977). On the effect of stratification when two stratifying variables are used. *Journal of the American Statistical Association 72*(357), 149–153. *Statistics in Transition 6*(2), 287–305.

Tom, M. (1997). Mitchell: Machine learning. *1997 Burr Ridge 45*(37), 870–877.

Tschuprow, A. A. (1923). *On the mathematical expectation of the moments of frequency distributions in the case of correlated observation*. Taddei.

Tschuprow, A. A. (1939). Principles of the mathematical theory of correlation. Technical report.

Tschuprow, A. A. and A. Tschuprow (1925). *Grundbegriffe und grundprobleme der korrelationstheorie*. BG Teubner.

Turing, A. M. (1950). Mind. *Mind 59*(236), 433–460.

Nations, U. (2017, Nov). Global commodity trade statistics.

Bureau, US Census (2016). *2015 ACS Public Use Microdata Sample (PUMS)*. Washington, D.C. `https://factfinder.census.gov/faces/nav/jsf/pages/searchresults.xhtml?refresh=t#`.

Valliant, R. and J. E. Gentle (1997). An application of mathematical programming to sample allocation. *Computational Statistics & Data Analysis 25*(3), 337–360.

Van Laarhoven, P. J. and E. H. Aarts (1987). Simulated annealing. In *Simulated annealing: Theory and applications*, Springer.

Varanelli, J. M. (1996). *On the acceleration of simulated annealing*. Citeseer.

Vesanto, J. and E. Alhoniemi (2000). Clustering of the self-organizing map. *IEEE Transactions on neural networks 11*(3), 586–600.

Vouros, A., S. Langdell, M. Croucher, and E. Vasilaki (2021). An empirical comparison between stochastic and deterministic centroid initialisation for k-means variations. *Machine Learning 110*(8), 1975–2003.

Wehrens, R., L. M. Buydens, et al. Flexible self-organizing maps in kohonen 3.0. *Journal of Statistical Software*.

Wickham, H., J. Hester, W. Chang, K. Müller, and D. Cook (2021). *memoise: 'Memoisation' of Functions*. R package version 2.0.1.

Willighagen, E. (2005). Genalg: R based genetic algorithm. *R package version 0.1 1*.

Winston, P. H. (1992). *Artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc.

Yeung, K. (2020). Recommendation of the council on artificial intelligence (oecd). *International Legal Materials 59*(1), 27–34.

Zlochin, M., M. Birattari, N. Meuleau, and M. Dorigo (2004). Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research 131*(1), 373–395.