A Grouping Genetic Algorithm for Joint Stratification and Sample Allocation Designs

Mervyn O'Luing,[1] Steven Prestwich,[1] S. Armagan Tarim[2]

Abstract

Finding the optimal stratification and sample size in univariate and multivariate sample design is hard when the population frame is large. There are alternative ways of modelling and solving this problem, and one of the most natural uses genetic algorithms (GA) combined with the Bethel-Chromy evaluation algorithm. The GA iteratively searches for the minimum sample size necessary to meet precision constraints in partitionings of atomic strata created by the Cartesian product of auxiliary variables. We point out a drawback with classical GAs when applied to the grouping problem, and propose a new GA approach using "grouping" genetic operators instead of traditional operators. Experiments show a significant improvement in solution quality for similar computational effort.

Keywords: Grouping genetic algorithm; Optimal stratification; Sample allocation; R software.

[1] *Insight Centre for Data Analytics, Department of Computer Science, University College Cork, Ireland.* Email: mervyn.oluing@insight-centre.org, steven.prestwich@insight-centre.org

[2] *Cork University Business School, University College Cork, Ireland.* Email: armagan.tarim@ucc.ie

1 Introduction

In this paper we address the optimization problem of jointly determining stratification and sample allocation for univariate and mulitivariate scenarios. To serve this purpose, we refer to (Ballin and Barcaroli, 2013). 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. The Bethel-Chromy algorithm combines similar algorithms from (Bethel, 1985, 1989) and (Chromy, 1987) and is suitable for univariate and mulitivariate cases. It uses lagrangian multipliers to find the minimum sample size that meets precision constraints for a given stratification. (Ballin and Barcaroli, 2013) combine a GA with this 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.

In this paper we propose to 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 has been updated following this work (Barcaroli, 2019). Section 2 motivates the work and introduces GGAs. Section 2.3 describes our GGA for the problem. Section 3 compares the original GA with our GGA on publicly-available test data. Section 4 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 5 concludes the paper.

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.

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 $\{4\}$.

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. We shall illustrate these differences on the above examples.

GGAs represent a grouping as an ordered list of subsets, omitting empty sets. The parents in the second example of Section 2.1 might 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 $\{3,6\}$. In the first example of Section 2.1 it is easily verified that both offspring represent the same grouping as the parents, as one would expect. 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 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 an additional genetic operator called *inversion* 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.

Injection, mutation and inversion are the common operators used in GGAs, but 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. In Section 2.3 we design a GGA for our problem.

2.2.1 Note on implementation

For the sake of clarity the descriptions in Section 2.2 omit implementation details, for example the fact that GGA chromosomes are usually implemented in two parts (or sometimes more). The first part uses a classical representation as above, while the second part lists the nonempty groups as a permutation. Injection occurs on the second parts of parent chromosomes and some renaming of groups is necessary.

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, 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 datasets, or with the existing dataset 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 certain 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 processing 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). We apply 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. Thus we compare 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.

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, because this is enough to maintain diversity.

The size of the chromosome population can be decided by trial and error. 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 datasets we used in this paper (we discuss this further in Section 4).

For further details on the implementation of GGAs (e.g. elitism rate) we refer the reader to papers such as (Falkenauer, 1998).

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 4) 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. The GGA is summarised in Figure 2.1.

Grouping Genetic Algorithm (GGA)

Step 1: Initialization
(a) Randomly generate a chromosome population of size N_P

Step 2: Selection part 1
(a) Rank chromosomes based on sample size
(b) Save best E chromosomes for the next generation

Step 3: Inversion
With probability 0.01 invert groups in the N_P chromosomes

Step 4: Selection part 2
For each of the remaining $N_P - E$ 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 2.2
(c) Remove empty groups
(d) Renumber groups

Step 5: Mutation
Mutate integers in $N_P - E$ chromosomes at a selected probability

Step 6: if #iterations<maximum
(optional: and sample size > desired value) go to step 2

Figure 2.1: Pseudocode for our GGA

Following the problem statement in (Ballin and Barcaroli, 2013) we summarise the cost

function as follows:

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

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 C_0 is set to 0, and C_h is set to 1. The expectation of the estimator of the "g-th" population total is:

$$E\left(\hat{T}_g\right) = \sum_{h=1}^{H} N_h \bar{Y}_{h,g} \quad (g = 1, \ldots, G)$$

where $\bar{Y}_{h,g}$ is the mean of the G different target variables Y in each stratum h. The variance of the estimator is given by:

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

The upper limit of variance or precision U_g is expressed as a coefficient of variation CV for each $\hat{T}_g$:

$$CV(\hat{T}_g) = \frac{\sqrt{VAR(\hat{T}_g)}}{E(\hat{T}_g)} \leq U_g \tag{2}$$

The problem can be summarised as follows:

$$\begin{aligned} &\min n = \textstyle\sum_{h=1}^{H} n_h \\ &CV\left(\hat{T}_g\right) \leq U_g \end{aligned}$$

3 Comparing the Genetic Algorithms

We now run a number of comparisons between the original GA and our GGA using publicly available datasets. Unless otherwise stated, for all the cases presented below, we adopt the following parameter setting for both genetic algorithms, where $N_{\mathrm{P}} = 20$, $U_{\mathrm{g}} \equiv 0.05$, the elitism rate is 0.2, and the mutation probability is 0.05.

3.1 A comparison for the iris dataset

(Ballin and Barcaroli, 2013) use the iris dataset (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 dataset 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 cross product of the categorical version of Sepal Length with Species creates 9 atomic strata. However, one atomic 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 3.1: Reproduction of table of atomic strata for estimating the minimum sample size for the target variables of iris dataset 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 3.1 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 deviations. 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 $N_p = 10$ in this case. For this test the *bethel.r* function will search for the minimum sample size, in integers rather than real numbers. 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. 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 dataset experiment.

Table 3.2: Iris dataset experiment results for GA and GGA

(a) GA			(b) GGA		
Number of			Number of		
Experiment	**Iterations**	**Chromosomes**	**Experiment**	**Iterations**	**Chromosomes**
1	14	228	1	11	180
2	8	132	2	7	116
3	17	276	3	6	100
4	40	644	4	22	356
5	31	500	5	9	148
6	13	212	6	11	180
7	15	244	7	8	132
8	9	148	8	7	116
9	15	244	9	9	148
10	15	244	10	11	180
11	14	228	11	3	52
12	8	132	12	9	148
13	17	276	13	27	436
14	40	644	14	12	196
15	31	500	15	16	260
16	13	212	16	6	100
17	15	244	17	20	324
18	9	148	18	6	100
19	15	244	19	7	116
20	15	244	20	6	100
21	16	260	21	11	180
22	67	1076	22	7	116
23	19	308	23	8	132
24	9	148	24	5	84
25	11	180	25	7	116
26	20	324	26	5	84
27	32	516	27	6	100
28	10	164	28	6	100
29	37	596	29	9	148
30	9	148	30	6	100

Table 3.2 provides the number of iterations (and chromosomes generated) taken to find n=11 over 30 experiments for both GAs.

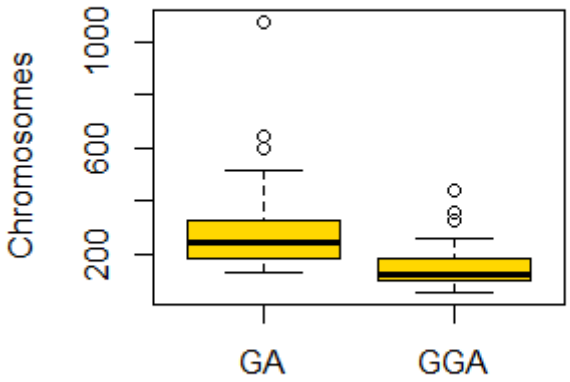


Figure 3.1: *Boxplot distribution of number of Chromomomes generated to find n=11 for GA and GGA after 30 experiments*

Figure 3.1 provides the distribution of the number of chromosomes generated to find the optimal solution for the GA and the GGA. The boxplots indicate that the GGA typically needs to generate fewer chromosomes to find the optimum solution.

Table 3.3: Example stratifications for the GA and GGA on the iris dataset 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 3.3 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. It is seen that the GGA results in fewer, less fragmented design strata. The same tendency can be observed in the latter cases.

3.2 Swiss municipality dataset

The swissminucipalities dataset 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 men and women aged between 0 and 19,

Y2: number of men and women aged between 20 and 39,

Y3: number of men and women aged between 40 and 64,

Y4: number of men and women aged 65 and over.

We consider 6 auxiliary variables, formed using the same k- means clustering method as the iris dataset 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 3.2 below.

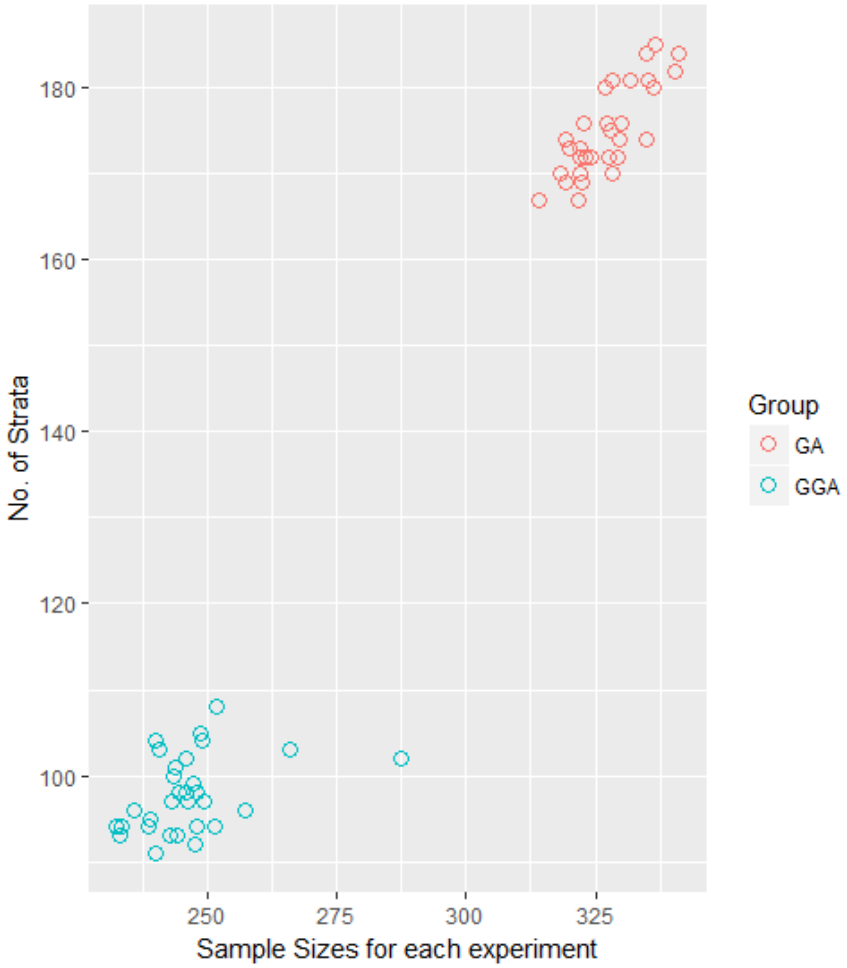


Figure 3.2: *Scatterplot of Results for Strata v Sample size for GA and GGA after 30 experiments*

Figure 3.2 clearly shows that the GGA returns a smaller sample size to the GA for these settings. The median for the GGA, 246, is 25% lower than that for the GA, 328.

3.3 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 2015 ACS Public Use Microdata Sample (PUMS) file (US Census Bureau, 2016) is a sample of actual responses to the ACS representing 1% of the US population. The PUMS file contains 1,496,678 records each of which represents a unique housing unit or group quarters. There are 235 variables. The full data dictionary is available in (US Census Bureau, 2016). 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).

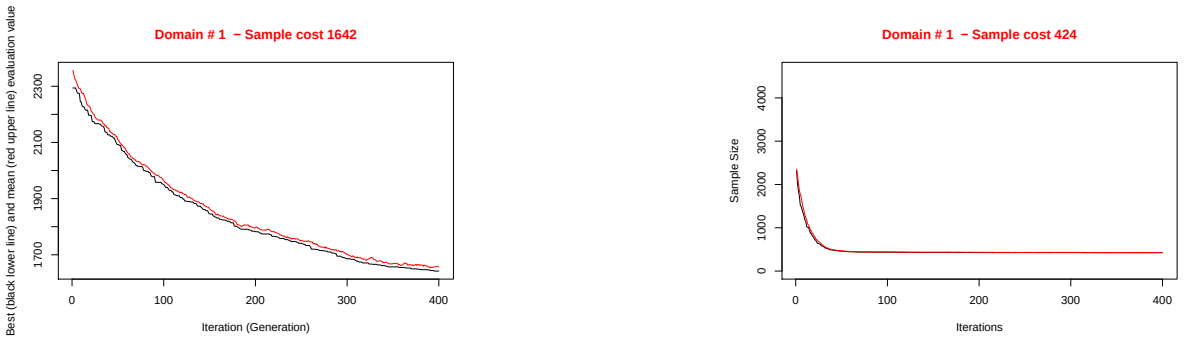
and the following auxiliary variables:

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.

The PUMS data for which all the values are present contains 619,747 records. We use the 51 states (based on census definitions) as domains.

In the convergence plots of Figure 3.3, 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 3.3: 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.

3.4 Kaggle Data Science for Good challenge Kiva Loans data

The online crowdfunding platform kiva.org provided a dataset 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 dataset 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:

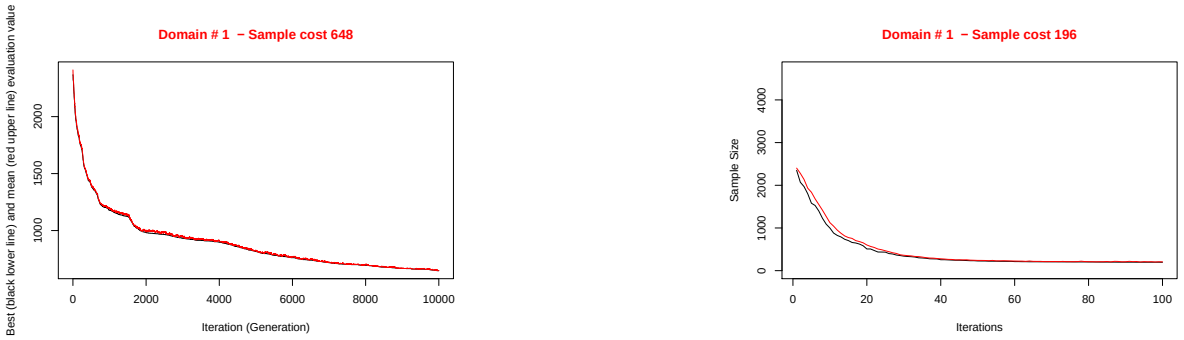
1. sector
2. currency
3. activity
4. region
5. partner id

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. The variable country-code defines the 73 design domains in this experiment.

Table 3.4: 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 3.4 shows an 84.67% reduction in sample size and a 95.83% reduction in the number of strata after 100 iterations. Figure 3.4 shows that for the same starting chromosome population size for Domain 1 of the Kiva Loans dataset, 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 3.4: Convergence plots for Sample Size for the 1st Domain for GA (10,000 iterations) and GGA (100 iterations) in the Kiva Loans dataset experiment. Note the different scales on the vertical and horizontal axes

3.5 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 3.5: 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%

3.6 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 variable combinations. 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 PUMS dataset we have taken a subset for which all values are present. This has resulted in a subset with 627,611 records. The domain

for this experiment was Census region and division.

Table 3.6: Sample size and strata for the 2000 US census data by Census 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 for the GGA and a sample size of 477,493 for the GA after 100 iterations.

4 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 dataset the experiment took approximately 30 days for either GA or GGA with 100 iterations.

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 dataset of any functional size (which we have the computation power to process) for any number of iterations. Bigger datasets 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 4.7: Performance comparison for the above datasets using the** R **and** Rcpp **versions of the Bethel-Chromy algorithm

Dataset	**Records**	**Domains**	**Atomic Strata**	**Bethel** μ**s**	**BethelRcpp** μ**s**	**Speed-up Factor**
iris	150	1	8	2684.77	143.13	18.76
swissmunicipalities	2,896	7	641	99,916	10,749.51	9.29
American Community Survey 2015	619,747	51	123,007	565,278,500	47,858,200	11.81
Kiva Loans Data	614,361	73	84,897	826,297,710	82,894,480	9.97
UN Commodity Trade Data 2011	351,057	171	350,895	139,749,810	87,555,870	1.6
US Census Data 2000	627,611	9	517,632	2,686,771	1,303,667	2.06

Table 4.7 shows the median time taken to run the Bethel algorithm one hundred times for the datasets 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*. However, 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. 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.

5 Conclusion and further work

We created a GGA as an alternative to the existing *SamplingStrata* GA in R. We then compared the two algorithms using a number of datasets. The GGA compares favourably with the GA at finding the correct solution and meeting constraints on smaller datasets, but significantly outperforms the GA on larger datasets where the number of iterations was restricted. This is useful for datasets 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.

This work 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). For example instead of searching for a stratified simple random sample to meet precision constraints based on population totals or means, 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).

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.

Acknowledgement

We 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. We would also like to thank Giulio Barcaroli and Marco Ballin, the co-authors of (Ballin and Barcaroli, 2013), for independently testing our GGA. Last but not least we are 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.

References

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.

Anderson, E. (1935). The irises of the gaspe peninsula. *Bulletin of the American Iris society 59*, 2–5.

Ballin, M. and G. Barcaroli (2013). Joint determination of optimal stratification and sample allocation using genetic algorithm. *Survey Methodology 39*(2), 369–393.

Barcaroli, G. (2014). SamplingStrata: An R package for the optimization of stratified sampling. *Journal of Statistical Software 61*(4), 1–24.

Barcaroli, G. (2019 (accessed April 29, 2019)). Optimization of sampling strata with the SamplingStrata package. https://cran.r-project.org/web/packages/SamplingStrata/vignettes/SamplingStrata.html.

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.

E.C. Brown and M. Vroblefski. A grouping genetic algorithm for the microcell sectorization problem. *Engineering Applications of Artificial Intelligence 17*(6), 589–598.

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 (2016). *2015 ACS Public Use Microdata Sample (PUMS)*. Washington, D.C. https : / / factfinder.census.gov / faces / nav / jsf / pages / searchresults.xhtml?refresh=t#.

Chromy, J. R. (1987). Design optimization with multiple objectives. *Proceedings of the Survey Research Methods Section*.

De Lit, P., Falkenauer, E., Delchambre A. Grouping genetic algorithms: an efficient method to solve the cell formation problem.

Eddelbuettel, E. (2013). Seamless R and C++ Integration with Rcpp *ISBN 978-1-4614-6867-7* 10.1007/978-1-4614-6868-4

Falkenauer, E. (1998). *Genetic algorithms and grouping problems.* John Wiley & Sons, Inc.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics 7*(2), 179–188.

Galinier, P. , Hao, J. K.. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization* 3(4):379–397, 1999.

Hartigan, J. A. and M. A. Wong (1979). Hybrid Evolutionary Algorithms for Graph Coloring.Algorithm as 136: A k-means clustering algorithm *Journal of the Royal Statistical Society.* Series C (Applied Statistics) 28(1), 100–108.

Hung, C. , Sumichrast, R. T., Brown, E.C. CPGEA: a grouping genetic algorithm for material cutting plan generation. *Computers & Industrial Engineering* 44(4), 651-672.

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.

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.

Pelikan, M. and D. E. Goldberg (2000), Genetic Algorithms, Clustering, and the Breaking of Symmetry. *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature,.*

Prügel-Bennett, A. (2004). Symmetry Breaking in Population-Based Optimization. *IEEE Transactions on Evolutionary Computation 8*(1), 63-79.

Stokes, L. and J. Plummer (2004). Using spreadsheet solvers in sample design. *Computational statistics & data analysis 44*(3), 527–546.

R Core Team (2015). *R A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Ruggles, S., K. Genadek, R. Goeken, J. Grover, and M. Sobek (2017). Integrated public use microdata series: Version 7.0 [dataset]. minneapolis: University of minnesota, 2017.

Willighagen, E. (2005). Genalg: R based genetic algorithm. *R package version 1*.