Títol: Using Genetic Algorithms for Attribute Grouping in Multivariate Microaggregation

Volum: 1

Alumne: Jordi Balasch i Masoliver

Director/Ponent: Victor Muntés Mulero

Departament: Arquitectura de Computadors

Data: 29 de gener de 2010

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

PROJECTE FINAL DE CARRERA

Using Genetic Algorithms for Attribute Grouping in Multivariate Microaggregation

Alumne:
Jordi Balasch i Masoliver

Director:
Victor Muntés Mulero

Acknowledgements

Foremost, I would like to thank my daily supervisors Victor Muntés and Jordi Nin for their support and guidance during the development of this project. Without their help, this thesis would have not been possible. Also my gratitude to Josep Lluís Larriba for giving me the opportunity to develop the project at the DAMA-UPC research group.

I would also want to thank the people from the CRISES research group of the Universitat Rovira i Virgili for providing useful source code of several quality measures for protected microdata.

Very special thanks to Joan Guisado, Arnau Prat and David Domínguez for making my work enjoyable during this year.

Last but not least, to my parents, my brother and Laia for their love, support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, any public or private entity stores a vast amount of information in databases. When this information is recorded separately for every person is called microdata. Part of it might be confidential, such as medical information or salaries, and by law it must be kept and treated in a specific way to ensure individual's privacy.

Many situations require the need to transfer microdata to third parties, for example when official statistical institutes or researchers want to perform a data analysis. Then, if microdata contains sensitive information, it must not be released directly since the privacy of individuals could be breached. In these cases, it is necessary to anonymize this microdata.

The main objective of data anonymization is to protect individuals' privacy, *i.e.*, that any intruder cannot relate each database entry to the corresponding unique individual. Nevertheless, this process must keep the statistical properties of data as much as possible in order to maintain its usefulness. Thus, any anonymization method faces a trade-off between data privacy and its statistical utility.

Microaggregation [2, 6] is one of the most commonly used anonymization techniques. Broadly speaking, if we have a dataset R with m records of n attributes each, during the microaggregation process clusters of k records are created and each record is replaced by the value of the centroid of the cluster, generating a protected dataset R'. After this process, any record r is indistinguishable among other k - 1 records (this property is known as k-anonymity [17, 18]).

In the example shown in Table 1.1, records are clustered in the same order they appear in the dataset, but many other clustering strategies exist. Normally, microaggregation tries to cluster the closest records together, in such a way that the distances between the records and the corresponding centroids is as small as possible and thus, the protected dataset is similar to the original one. However, to find the optimal cluster configuration, *i.e.*, the best way to cluster the records to mantain the microdata statistical utility,

	Original Microdata R			Protected Microdata R'			
	a_1	...	a_n	a'_1	...	a'_n	
r_1	r_{11}		r_{1n}	$\bar{r}_{k1}$		$\bar{r}_{kn}$	r'_1
r_2	r_{21}		r_{2n}	$\bar{r}_{k1}$		$\bar{r}_{kn}$	r'_2
...		...			...		...
r_k	r_{k1}		r_{kn}	$\bar{r}_{k1}$		$\bar{r}_{kn}$	r'_k
$r_{(k+1)}$	$r_{(k+1)1}$		$r_{(k+1)n}$	$\bar{r}_{(2k)1}$		$\bar{r}_{(2k)n}$	$r'_{(k+1)}$
$r_{(k+2)}$	$r_{(k+2)1}$		$r_{(k+2)n}$	$\bar{r}_{(2k)1}$		$\bar{r}_{(2k)n}$	$r'_{(k+2)}$
...		...			...		...
r_{2k}	$r_{(2k)1}$		$r_{(2k)n}$	$\bar{r}_{(2k)1}$		$\bar{r}_{(2k)n}$	r'_{2k}
...		...			...		...

Table 1.1: Basic microaggregation process.

is a NP-hard problem [14]. This problem has been widely studied in the literature and a large variety of heuristic algorithms exist.

As the number of attributes increases, the microaggregation process must deal with more dimensions and, therefore, the distance between the centroid and the records also increases, making the protected dataset R' very different to the original dataset R. Hence, the statistical information that can be drawn from the protected dataset is more unreliable.

To avoid this drawback, multivariate microaggregation is used instead of the classical one: attributes are partitioned in different sets and microaggregation is applied to each one individually. This method helps to maintain microdata utility, but it can lead to a complete loss of the k-anonymity property because two records can be clustered together considering a set of attributes and separately when considering another set, so they are not undistinguishable anymore [12]. Then, individuals are easier to identify than before, *i.e.*, the risk of re-identification increases.

The way the attributes are grouped is crucial to achieve a protected dataset which preserves as much as possible the data utility and guarantees the privacy of individuals. Some studies in the literature [13] show that the attribute partitioning influences in the quality of the microaggregation as much as the record clustering strategy. The common practice in statistical agencies is to build hand-made groups, analyzing properties of attributes such as correlation. However, there have not been serious attempts to find the optimal attribute grouping in a general case.

1.1 Motivation

Some optimization problems are hard to solve, and the only known way to find the optimal solution is with the use of brute-force algorithms, *i.e.*, all possible solutions to the problem are tested and the best one is the optimal.

However, its cost is proportional to the number of possible solutions, which, in many practical problems, tends to increase exponentially as the size of the problem grows. That is the case of the attribute grouping problem in multivariate microaggregation.

Nevertheless, in real life, optimality is not mandatory and it is possible to simplify the problem to find a good solution close to the optimal one. This is where approximation algorithms play an important role. Among these algorithms, the evolutionary approaches have been proven efficient for several well-known problems. There are several aspects that make the use of this type of algorithms successful: the search space is complex and it is not well-known, it is possible to find a suitable encoding to represent the solutions of the problem to be optimized, and finally, it is possible to evaluate each possible solution using a fitness function. Moreover, evolutionary algorithms have shown to be also effective with grouping problems as well [8].

This project aims at bringing together two branches of computer science such as artificial intelligence and data privacy to try to solve the attribute grouping problem in multivariate microaggregation. We will propose, implement and analyze a novel approach to find the optimal, or near optimal, attribute partitioning, based on the use of genetic algorithms in order to explore the vast space of all possible attribute groupings.

1.2 Basic Concepts: Anonymization

Anonymizing a dataset consists in removing or modifying the identifying attributes it contains so that it can be released without revealing confidential information that can be linked to specific individuals. Formally, a dataset R can be seen as a collection of m records where each record r represents a point in a multidimensional space defined by the number of attributes. Depending on its capability to identify unique individuals, an attribute a can be classified in two disjoint categories:

- *Identifiers.* An identifier attribute a_{id} is able to identify unambigously a single individual; common examples are the passport number or the social security number.

- *Quasi-identifiers.* A quasi-identifier attribute cannot identify an individual when it is used alone; nevertheless, when some of these attributes are combined, they can uniquely identify an individual. Depending on whether they contain sensitive information about the individual or not, quasi-identifier attributes can be classified as non-confidential attributes (a_{nc}) and confidential attributes (a_c). Typical examples of non-confidential attributes are the postal code or the age, and an example of a confidential attribute is the salary.

Usually, when releasing a dataset, the scenario depicted in Figure 1.1 is followed. As identifiers are not needed for statistical or research purposes, they are either removed or encrypted. Normally, the most useful information is contained in the confidential attributes, so they are not modified. In order to preserve the privacy of the individuals, an anonymization method is applied to non-confidential quasi-indentifier attributes. This scenario allows third parties to have precise information on confidential data without revealing to whom the confidential data belongs to.

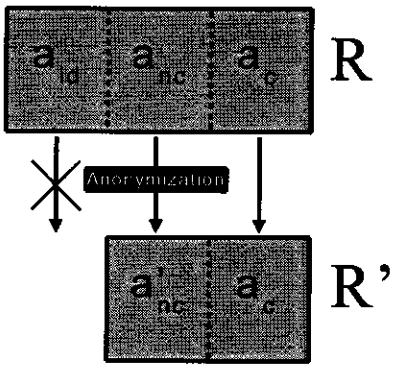

Figure 1.1: Microdata anonymization process.

The goal of any protection method is to achieve an acceptable level of privacy. Nevertheless, as third parties want to perform reliable statistical analysis, the protection method should guarantee that the protected dataset is as similar as possible to the original data. Therefore, the main objectives of any anonymization method must be:

- *Protecting the privacy of individuals.* Data has to be sufficiently modified to make identification difficult. The measure used to evaluate the risk of re-identification is called *Disclosure Risk.*
- *Preserving data utility.* The anonymization method should introduce as small noise as possible into the data to preserve its utility. The *Information Loss* measure is used in this case.

As it can be seen, these two goals are inversely related: as one of them increases, the other one decreases. One could think in two different extreme examples: if the original data is encrypted and released, the disclosure risk is (almost) zero, but the information loss is maximum. On the contrary, if the original data is released; the information loss is zero, but the disclosure risk is maximum.

There are several measures to evaluate the quality of a data protection method. One of the most accepted and used is the *score* [4], which is defined

as the average of the two values considered:

$$score = \frac{IL + DR}{2},$$

where IL stands for information loss measure and DR stands for disclosure risk measure.

On the one hand, the information loss measure takes into account five general parameters of the microdata distribution, which are compared to establish the silmilariry between the original and the protected microdata. These parameters are:

a. IL_1. Mean absolute error of the original microdata R with respect to the protected data R'.

b. IL_2. Mean variation of the attribute average vector.

c. IL_3. Mean variation of the attribute covariace matrix.

d. IL_4. Mean variation of the attribute variance vector.

e. IL_5. Mean variation of the attribute correlation matrix.

The overall IL is computed as follows:

$$IL = 100(0.2IL_1 + 0.2IL_2 + 0.2IL_3 + 0.2IL_4 + 0.2IL_5)$$

On the other hand, in order to compute the disclosure risk, usually two approaches are considered. The first one is the Interval Disclosure Risk ID which is the average percentage of protected values falling into an interval around their corresponding original values. This measure was introduced in [1].

The second considered one is the entity disclosure risk. This measure is usually implemented using Record Linkage RL methods, which consider the scenario where intruders have access to an external dataset containing some non-confidential quasi-identifiers a_{nc} of a set of original records ($r \in R$). Then, they try to link each record of their with the corresponding protected record $r' \in R'$. The percentage of correct links achieved by the intruder is taken as the RL measure. Basically, there are two families of such methods [19]:

- *Distance based Linkage Disclosure risk (DLD).* Each original record obtained from an external dataset is linked to the closest protected record using the Euclidean distance. Generally, it is assumed that the intruder has several different sets of non-confidential quasi-identifiers and the risk is computed as the average risk of all those sets.

- *Probabilistic Linkage Disclosure risk (PLD).* The link between the original record and the protected record is assigned in a probabilistic way, according to some criterias on a coincidence vector.

When computing disclosure risk, the importance of record interval disclosure and record linkage is distributed evenly. Then, the record linkage is defined as the average of the two methods defined above. Therefore, the overall DR is computed as follows:

$$DR = 0.5ID + 0.25DLD + 0.25PLD$$

Regarding record linkage, it is important to say that it is used for other different purposes. For instance, when two organizations want to merge their databases, record linkage is used to identify information belonging to the same entity, so that the merge process ends with a consistent database.

1.3 Goals

The main purpose of this project is to design and implement a genetic algorithm to find quasi-optimal solutions to the problem of attribute grouping in multivariate microaggregation, along with different mechanisms to monitorize it and thoroughly analyze its behavior. Our genetic optimizer is called GOMM, which stands for Genetic Optimizer for Multivariate Microaggregation. In more detail, the objectives of this project are:

- To study the state of the art of multivariable microaggregation: before starting the design of the genetic optimizer, the problem and some approaches proposed in the literature will be thoroughly studied.
- To design and implement GOMM: the procedure of any genetic algorithm is well known: a collection of instances (population) that represent solutions to the problem suffers a set of transformations (genetic operations) to generate new instances and evaluate their quality, discarding the worst ones; thereby after several iterations (generations) the survivor instances represent near-optimal solutions. This objective also includes:
  - To study and implement a representation of the solution (chromosome): a way to encode any solution to the problem will be found in order to facilitate the task of genetic operators.
  - To propose and implement the genetic operations: two types of operators will be defined: a crossover operation, which creates new instances who inherit properties from two parent instances, and mutation operations, which introduce new properties to the generated instances and allow the exploration of the entire solution space.

  - To define and implement a cost function: its job is to evaluate the quality of a particular solution; since the methods used in literature to evaluate the information loss and the disclosure risk are complex and inefficient, we will study different alternatives for their efficient calculation.

- To perform a thorough analysis of the evolution of the quality of the solutions and the behavior of operators during the execution of the algorithm: to understand the mechanisms that lead the algorithm to a particular solution we will implement monitorization tools, which will generate useful information in order to propose improvements.

- To propose new dynamic techniques to improve the optimizer in terms of quality and time convergence: when an early version of the optimizer is running, we will seek measures to improve its performance; for example, as operators will be monitored, we will have information to determine its behavior and thus, to introduce mechanisms that dynamically control how many operations of each type are made.

- To make a comparison of the experimental results and evaluate their quality: once the algorithm is working and providing solutions, the coherence and quality of the solution will be compared with other methods proposed in the literature.

- To write the project thesis: we will write this document, that includes all the work done, explaining the properties of the final genetic algorithm and the analysis of its behavior, along with the discarded alternatives.

1.4 Document Structure

The structure of this document is organized as follows. Chapter 2 gives some theoretical background about microaggregation and evolutionary algorithms, which are the two main pilars of this project. Afterwards, Chapter 3 presents the Genetic Optimizer for Multivariate Microaggregation (GOMM), with a detailed explanation of its implementation. In Chapter 4, an inside analysis of our genetic approach is given in order to understand its behavior and propose new features to improve its performance. The experimental results given by GOMM and a comparison with other attribute grouping strategies are presented in Chapter 5. Then, Chapter 6 shows the project scheduling and its economic analysis. This report finishes with Chapter 7, where the conclusions and future work are presented.

Chapter 2

Preliminaries

This chapter explains the basic concepts to understand the work done in this project. First of all we introduce microaggregation and algorithms to implement this technique, and specially, we describe multivariate microaggregation and its advantages and disadvantages. Moreover, we give a complete view of the problem of grouping attributes in order to achieve a protected dataset with both low information loss and disclosure risk, presenting related work found in the literature. Secondly, we present a brief description of evolutionary algorithms and the reasons for choosing this kind of approach to solve our problem.

2.1 Microaggregation

Microaggregation is an anonymization technique which allows the publication of the entire dataset, although intead of exact values, modified values are given. This method achieves data protection from a twofold perspective. Firstly, if data is modified, reidentification by means of record linkage is harder and uncertain. Secondly, even when an intruder is able to re-identify an individual, she cannot be confident that the released data is consistent with the original data.

Microaggregation was first proposed in [2] as a statistical disclosure method for numerical variables. If we consider a dataset R containing m records of n attributes, the idea is to replace an observed record r with the centroid computed on a small cluster of individuals, including the investigated one. The clusters contain a minimum predefined number of k records, which will be represented in the released file by the same value. Thus, the microaggregation mechanism achieves data privacy by ensuring that for every protected record r' there are at least k - 1 records with the same value in the dataset, which means that the dataset is k-anonymous.

Regarding data utility, the issue consists in determining the partition of the whole dataset in clusters of at least k individuals minimizing the informa-

tion loss. Therefore, the clusters are constructed according to a criterion of maximum similarity between them. Each record r can be viewed as a point in a multidimensional space defined by the number of attributes. If records are characterized as points, $r = \{a_1, \ldots, a_n\}$, similarity between them can be measured using a distance measure such as the Euclidean ditance [15].

A common measure to evaluate the information loss of a protected dataset using microaggregation is the total Sum of the Square Error SSE. This measure is the sum of squared distances from the centroid of each cluster to every record in the cluster, and is defined as:

$$SSE = \sum_{i=1}^{N} \sum_{r_j \in N_i} (r_j - \bar{r}_i)^T (r_j - \bar{r}_i),$$

where r_j are the records of the dataset R, N is the total number of clusters, N_i is the i-th cluster and $\bar{r}_i$ is the centroid of N_i. The restriction is $|N_i| \geq k$, for all $i = 1, \ldots, N$.

Exactly solving the microaggregation problem, that is, finding the way to cluster the records which minimizes the SSE where clusters have a size at least k, was shown to be a NP-hard problem [14]. Because of this, it is necessary to resort to heuristic approaches.

2.1.1 MDAV Algorithm

The best-known example of heuristic microaggregation methods is the Maximum Distance to Average Vector (MDAV) [6]. MDAV produces clusters of fixed cardinality k and, when the number of records is not divisible by k, one cluster with a cardinality between k and $2k$-1. The MDAV algorithm is described in Algorithm 1.

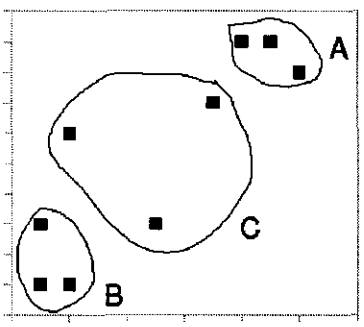

Figure 2.1: Clusters generated by the MDAV algorithm.

A simple example of the clustering strategy followed by the MDAV algorithm is shown in Figure 2.1, assuming a k value of 3. First of all, the

average point of all the points is calculated. Then, the algorithm searches the most distant point r_r to the average point, which in this case is the rightmost upper point. The next step is to built cluster A, that contains r_r and the k - 1 closest points to it. Afterwards, MDAV searches the furthest point r_s to r_r, which is the leftmost lower point. Then, it builds cluster B around it. Finally, as only three points are ungrouped, MDAV creates cluster C.

Algorithm 1: MDAV

```
Data: R: original microdata, k: integer
Result: R': protected microdata
1  begin
2      while (|R| > k) do
3          Compute the average record r̄ of all records in R
4          Consider the most distant record r_r to the average record r̄
5          Form a cluster around r_r. The cluster contains r_r together
           with the k − 1 closest records to x_r
6          Remove these records from R
7          if (|R| > k) then
8              Find the most distant record r_s from record r_r
9              Form a cluster around r_s. The cluster contains r_s
               together with the k − 1 closest records to r_s
10             Remove these records from file R
11     Form a cluster with the remaining records
12 end
```

MDAV algorithm can be instantiated for different data types, using appropriate definitions for distance and average. As we focus on numerical attributes, the most distant record and the closest records are computed using the Euclidean distance, and the average record is defined as the arithmetic mean of the records. The average record is used to replace the original records when building the protected dataset.

MDAV has proven to be one of the best performers in terms of time and one of the best regarding the homogeneity of the resulting clusters. Its computational complexity is $O(\frac{n^2}{2k})$, although some minor changes can be done to improve its performance; as we will see later on.

2.1.2 Multivariate Microaggregation

As we have explained before, if a microaggregation method is applied to all the attributes of the original dataset R at the same time; then, the resulting protected dataset R' satisfies the property of k-anonymity. Nevertheless, when the amount of attributes is large, the data utility of the protected

dataset is diminished. This is so because the larger the number of attributes, the larger the distance between the original records in the dataset and their corresponding centroids.

Then, in order to increase the data utility of the protected dataset, multivariate microaggregation is used: the dataset is split into smaller sets of attributes and the microaggregation method is independently applied to each one of the sets. In this way, the information loss is lower since the microaggregation method deals with less dimensions and the centroids are closer to the records. Nevertheless, this improvement in data utility is at the cost of a loss in the achieved level of privacy.

For example, k records which are in the same cluster taking into account the first set of attributes, might be in different clusters when all the other sets are considered. Therefore, when replaced by the respective centroids for all the attribute partitionings, these records are not undistinguishable anymore and, moreover, no k-anonimity will be achieved in general [12].

This problem is illustrated in Table 2.1, where the set of attributes has been splitted in two groups, containing an attribute each. After applying multivariate microaggregation, all the records in the protected dataset are different. Thus, this method does not guarantee k-anonimity.

Original Data R		Protected Data R'	
a_1	a_2	a'_1	a'_2
1	2	2	1.5
3	6	2	7
5	3	6.5	1.5
8	9	6.5	9.5
12	8	12.5	7
13	10	12.5	9.5

Table 2.1: Example of applying multivariate microaggregation with two attribute groups and clusters of $k = 2$ records.

There is a special case called univariate microaggregation, when each attribute is microaggregated separately. This is the easiest configuration in terms of attribute grouping complexity, and there are polynomial time algorithms to obtain the optimal microaggregation [9]. However, the main drawback of univariate microaggregation is that it provides a bad level of privacy, due to its high number of attribute sets. The case depicted in Table 2.1 is also an example of univariate microaggregation. For this reason, multivariate microaggregation is the most widely used.

Nevertheless, when using multivariate microaggregation, other factors influence the quality of the protected dataset. Beyond the value of the k parameter and the specific microaggregation method, the microaggregation problem has to take into account the number of groups into which the dataset

is split, the number of attributes in each group and, definitely, how to select which attributes form each group. The way the attributes are grouped is a problem as difficult as the microaggregation itself. Note that the search space grows exponentially with regard to the number of attributes and, therefore, finding the most appropriate attribute grouping configuration is far from trivial.

Two approaches were presented for attribute selection [13], both based on the correlations among attributes. The first approach is based on cluster in the same group attributes that are highly correlated, minimizing in this way the intra-cluster distance between the centroid and the records. This approach has a low information loss but a high disclosure risk as in the case of univariate microaggregation.

The second approach is based on cluster highly correlated attributes in different groups. The goal of this second approach is to increase the resulting anonymity. The rationale of this approach is the following one: If two records r_i and r_j are in the same cluster for some blocks, this means that the first attribute values of these records are more or less close to each other, and the same for the second attribute of the block, etc. Then, when we consider another block, if the j-th attribute of this new block is (highly) correlated with the j-th attribute of the latter block, records r_i and r_j will probably be close to each other as well, with respect to the attributes in the second block. Therefore, with some non-negligible probability, r_i and r_j will fall in the same cluster, reducing in this way the disclosure risk and obtaining a similar level of privacy than the basic microaggregation. Of course, the information loss of this latter approach is larger than the former one.

These two approaches show the importance of finding an appropriate attribute grouping in order to achieve a protected dataset with both low disclosure risk and information loss. Opposite to the optimal microaggregation, this problem has been disregarded in the literature assuming that attributes are grouped considering some background knowledge, such as attribute correlations when we are interested on minimizing the information loss, or considering that if the intruder only has access to a subset of attributes, then such attributes are grouped together.

2.2 Evolutionary Algorithms

Inspired by the theory of natural selection, evolutionary algorithms use some mechanisms based on evolutionary biology such as inheritance, mutation, selection and crossover to solve an optimization problem.

Genetic algorithms are one of most used evolutionary approaches. During the execution of the algorithm, a population of representations of possible solutions to the optimization problem evolves towards better solutions.

These representations are called chromosomes and the evolution process is realized by the application of genetic operations. Every iteration, also called generation, new chromosomes are created from the current population, either recombining or modifying the existing chromosomes. Then, the chromosomes with the worst fitness are discarted. This fitness is a measure of the quality of the solution, provided by the cost function. The algorithm proceeds until a termination condition is reached or a maximum number of generations has been produced. Then, the chromosome with the best fitness of the resulting population is taken as a quasi-optimal solution to the problem. Algorithm 2 describes this process more schematically.

Algorithm 2: Genetic Algorithm

```
1 begin
2     Generate the first population of individuals
3     Evaluate the fitness of each member in that population
4     while stop condition not met do
5         Breed new individuals through recombination and
          mutation operations to give birth to offspring
6         Evaluate the fitness of the new generated members
7         Select the members that will form next population
8     Extract the best-fitted member of the current population as
      the optimal solution
9 end
```

Typically, the first population is generated at random. The idea is that the first chromosomes should be as diverse as possible in order to let the algorithm explore the entire search space.

Following, new chromosomes are generated by using genetic operations. These operators are the crossover operation, which combines properties of two existing members of the population, and some mutations, which introduce transformations in a single individual. When all the offspring is created, the selection operation chooses the chromosomes that will be used in the next generation. Normally, the chromosomes with a better fitness survive, thus, the average fitness of the population increases.

Every generation this process is repeated until the stop condition is met. This condition can be a fixed number of generations, a minimum solution quality expected or a number of generations without improvements in the population.

The whole evolution process of any genetic algorithm is sustained by three basic pilars: the genetic representation of the solution, the genetic operations and the cost function.

Encoding

The classic way to map the solutions in a chromosome when using genetic algorithms is with an array of symbols over a given alphabet. Hence, the chromosomes can be seen as strings where each symbol, also called gene, refers to an object of the solution.

For instance, in the attribute grouping problem of multivariate microaggregation, every gene of a chromosome represents an attribute. Depending on the symbol, this attribute would belong to a group or another. For example the chromosome

ABCABA

encodes the solution where the first attribute is in group A, the second in group B, third in C, fourth in A, fifth in B and sixth in A. To represent all possible solutions to our problem, six symbols should be enough.

With this encoding, all the chromosomes have the same length. This property permits the genetic operations to be fast and very simple to implement.

The genetic operations

Basically, two operators are used to generate new chromosomes. The first one is the crossover operation, which aims at recombining existing properties of the population. Normally, two parent chromosomes are seleted at random from the population and two child chromosomes are produced by this operation.

The most commonly used crossover operation is the two-point crossover, which works as follows. First of all, a crossing site is randomly delimited in each one of the parent chromosomes. A crossing site is a subsection of a parent chromosome, defined by a lower bound and an upper bound chosen at random. For example, the following parent chromosomes

A—BC—ABA and E—FF—EEE

have the crossing site delimited by the first and the third genes. Then, the offsping is generated by exchanging the contents of the respective crossing sites. Thus, the previous chromosomes would produce

A—FF—ABA and E—BC—EEE

as their child chromosomes. Notice that each child has inherited properties from both parents.

As crossover operations only combine existing properties from the current population, a way to introduce new information is needed in order to explore the whole search space. The second genetic operation is the mutation, whose objetive is to introduce properties not present in the current

population to guarantee that any possible chromosome in the search space can be generated. This operator transforms a single parent into a new child chromosome.

The simplest mutation is defined as the smallest random modification of a chromosome. This means that this operator randomly changes the symbol of a gene. For exemple, the parent chromosome

ABCABA

could be mutated into the child chromosome

ABAABA

when the third gene changes its symbol from C to A.

The cost function

A genetic algorithm needs a way to compare the goodness of all the possible solutions, specially in the selection process. The cost function evaluates the quality of a given chromosome, which is normally known as fitness. The goal of any genetic algorithm could be either minimizing or maximizing the fitness function.

The cost function of a genetic algorithm should be fast and efficient because for every generation, it is required to calculate the fitness of every member of the population. Without the fitness evaluation, the selection method could not discard the worst chromosomes and the evolution process would not be possible.

This is one of the main drawbacks of the genetic algorithms since most of the real life problems have high computationally cost functions, which make the evaluation of these problems almost prohibitive. In order to obtain efficient genetic algorithms, an approximate cost function could be used instead of a real expensive fitness evaluation.

2.2.1 The Grouping Genetic Algorithm

The classic genetic algorithms are not suitable for solving grouping problems, where a set of objects must be divided into disjoint groups in an optimal way, such as the attribute partitioning problem of multivariate microaggregation. This happens because genetic algorithms give importance to the objects, that is, the genes of the chromosome represent objects; and the useful entities in grouping problems are the groups.

An example of this drawback can be observed when performing a crossover operation. As the genes of the classic encoding of a genetic algorithm represent objects, the recombination of two chromosomes with interesting groups normally ends with with the destruction of these groups. For instance, if we perform the two-point crossover to the parent chromosomes

A—ABA—BC and D—EFE—FE

they will generate

A—EFE—BC and D—ABA—FE

as their children. If group A and group F are interesting groups to find the optimal solution, the crossover is not able to bring them together because it works in the object level. Moreover, like in the example above, the crossover may destroy these groups in most cases and thus, the population would not evolve.

This problem is automatically solved if the crossover works with groups rather than objects. For example, if the crossover could select groups A and C from the first parent chromosome and group F from the second, the child chromosome produced would inherit the two interesting groups. To do this, a new encoding and new genetic operations are required.

The Grouping Genetic Algorithm *GGA* [8] is an evolution of the classic genetic algorithm where the chromosomes are group oriented rather than object oriented. This implies that the chromosomes have different length depending on the number of groups of the solution they encode. The classic genetic operations do not support variable-length chromosomes, therefore, special genetic operations that manipulate whole groups of items are required.

GOMM uses the encoding and some genetic operations defined by the Grouping Genetic Algorithm. We will give more details of their implementation when we describe GOMM in Chapter 3.

Chapter 3

GOMM: Genetic Optimizer for Multivariate Microaggregation

In this chapter we present and describe the Genetic Optimizer for Multivariate Microaggregation (GOMM), designed and implemented to find quasi-optimal solutions to the attribute partitioning problem of multivariate microaggregation. First of all, we introduce the basic scheme of the algorithm, which has been implemented in nearly 2000 lines of code. In the next section, we explain how the solutions to the attribute grouping problem are represented in GOMM. Afterwards, the parameters which form the cost function are detailed. Also, the different genetic operations (crossover, mutation and selection) are specified. Finally, we include a performance analysis to study possible improvements to deal with larger datasets. One of these improvements has been implemented in a version called D-GOMM, which is also presented in this chapter.

3.1 Methodology

GOMM is a genetic algorithm that follows the typical structure of this kind of algorithms, described in Chapter 2.

The first step of any genetic algorithm is to create the first population. GOMM builds the first population randomly from scratch. A number n of possible groups are considered, being n the number of attributes, and for every attribute a group is randomly selected. A total number P of chromosomes are created and stored in a population vector. Due to the random distribution of the attributes into groups, the first generated chromosomes represent solutions with a significant number of groups.

Once the first population is created, GOMM evaluates the fitness of all the initial chromosomes through a cost function. Then, the population is

sorted according to this fitness, so that the chromosomes with a better cost occupy the top positions of the population vector. The objective of this sorting is to simplify the work of the selection method, which is described later on.

At this point, the evolution process starts: during successive generations, the algorithm proceeds from one population to the next, this last being obtained by the application of various genetic operations. These operations are crossover operations, which combine properties of the existing members in the population; mutation operations, which introduce new properties to the population; and a selection method, which chooses the chromosomes that will form the next population.

GOMM performs C crossover operations, which choose two random parent chromosomes and generate two child chromosomes; and M mutation operations, which generate a child chromosome from a single parent. After this process, an offspring vector has been filled with $2C + M$ new chomosomes. Then, like in the first population, the fitness of all the new generated chromosomes is evaluated and the offspring vector sorted, so the best fitted children are placed in the top positions.

In order to keep the size P of the population constant, the selection operation is used to discard the worst fitted members. In GOMM; the selection method merges the population and the offspring vectors, so that it obtains the P best fitted chromosomes within the old and the new generated ones. These survivors will form the population of the next genetion, which the selection method stored already sorted.

This process is repeated iteratively until a stop condition is met. GOMM has no defined stop conditions and performs a concrete number G of generations. When this number is reached, the best solution is taken from the top of the final population.

Even though there is not a stop criterion defined in GOMM, the algorithm includes mechanisms to easily implement it. For example, if the algorithm is wanted to perform until no improvements are achieved in sucessive generations, GOMM is provided with a method which displays the mean fitness of the current population. If two successive populations have the same average fitness, no improvements have been introduced in the new population and the algorithm can stop.

3.2 Encoding

A genetic algorithm works with representations of solutions rather than the solutions themselves. Thus, a way to map any solution to a chromosome is required. The solutions of the attribute grouping problem of multivariate microaggregation are, as the name states, groups of attributes.

GOMM uses the same encoding scheme as the Grouping Genetic Algo-

rithm (GGA) [8]. Every chromosome c contains an object part c_{op}, where each gene represents the membership of the corresponding attribute in a group, and a group part c_{gp}, which contains the groups. For example, the chromosome shown in Figure 3.1 represents a solution where the attributes 1, 2 and 5 are grouped together and attributes 3 and 4 are in another group, with the group part written after the thick line.

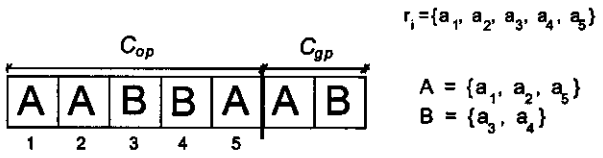

Figure 3.1: Parts of a simple chromosome.

The chromosomes used by GOMM have variable length. If the original dataset has n attributes, the shortest chromosomes have $n + 1$ genes and correspond to the solution where all the attributes are in a single group. On the contrary, the longest chromosomes, which represent the solution where every attribute is grouped alone (univariate microaggregation), have $2n$ genes.

The goal of this encoding is to facilitate the work of the operators which treat groups rather than objects. This is so because the groups are the meaningful parts of a solution.

It is important to remark that this encoding allows for the presence of *clones*, which are different chromosomes that represent the same attribute grouping and have the same associated fitness. For instance, Figure 3.2 represents clone chromosomes which map exactly the same solution as the chromosome shown in Figure 3.1.

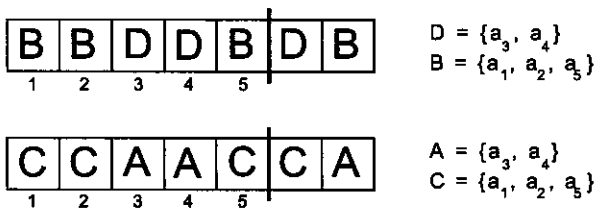

Figure 3.2: Clone chromosomes.

The alphabet used is the set of characters provided by ANSI C, thus, GOMM's chromosomes are strings with a maximum number of 128 different symbols. This implies that GOMM supports datasets with a maximum number $n = 128$ attributes.

3.3 The Cost Function

After encoding a solution in a chromosome c^i, a genetic algorithm needs a fitness function $F(c^i)$ to evaluate its quality. Chromosomes in our case represent attribute groupings for multivariate microaggregation. Hence, first of all a protection method is needed in order to generate a protected dataset and, secondly, a way to evaluate this protected dataset is required.

Microaggregation method

Before evaluating the quality of a protected dataset, GOMM's cost function needs to generate this protected dataset. Every chromosome encodes attribute partitionings to whom the microaggregation process will be applied. Therefore, the first step of GOMM's cost function is the application of a multivariate microaggregation method.

For this purpose, we chose the MDAV algorithm, previously described in Section 2.1.1. This algorithm is one one of the most well-known and flexible multivariate microaggregation methods and also one of the best regarding time performance.

However, the algorithm can be implemented in a faster way. Following, we propose two mechanisms to improve MDAV's performance:

- Re-calculating the dataset centroid from the cluster centroid: Every iteration of the MDAV algorithm the centroid of the remaining records $\bar{r}$ is calculated. This implies that the algorithm has to explore the whole dataset again. Nevertheless, when a cluster N_i is built, MDAV also calculates its centroid $\bar{r}_i$ in order to substitute the value of the clustered records. We use the calculation of $\bar{r}_i$ to re-calculate the centroid $\bar{r}$ without visiting all the remaining records again.

  To compute the cluster centroid $\bar{r}_i$ requires three loops:

$$\bar{r}_i = \frac{\sum_{r_l \in N_i} r_l}{k} = \{r_{i1}, \ldots, r_{in}\}$$

  1. Initialize to zero the value of every component r_{ij} of the centroid
  2. Calculate the total sum of every attribute for all the k records in the cluster
  3. Divide the previous addition by the number k of elements in the cluster

  To re-calculate the dataset centroid $\bar{r}$, we use loops 1 and 3. In the first loop, we multiply every component of the centroid for the number n_r of records of the dataset in the begining of the iteration. Thus, we get the sum of every attribute for all the n_r records in the dataset.

Afterwards, in the third loop, we subtract the cluster sum calculated in loop 2 from the dataset sum, obtaining the sum of every attribute for the $n_r - k$ values that will remain in the dataset when the clustered records are deleted. Then, we only have to divide this value by $n_r - k$ in order to obtain the new dataset centroid. Formally:

$$\bar{r}_{new} = \frac{n_r \bar{r} - \sum_{r_l \in N_i} r_l}{n_r - k} = \frac{\sum_{r_u \in R} r_u - \sum_{r_l \in N_i} r_l}{n_r - k}$$

- Matrix of distances: There are three steps of the MDAV algorithm where the distance between records is calculated. The first one is when the algorithm searches the k - 1 closest records to r_r. Then, the distances are calculated again to find the furthest record r_s to r_r and a third time to search the k - 1 closest records around it.

  All these distance calculations, repeated in every iteration of the algorithm, could be avoided if a matrix of distances is filled at the begining of the algorithm.

GOMM incorporates the first optimization, but the second one is not implemented. This is so because at the moment, GOMM deals with small datasets and mantaining a matrix of distaces could be counterproductive. However, it is important to take this optimization into account if the algorithm has to work with larger datasets.

Microaggregation quality

It is well-know that a good anonymization method is the one that minimize the trade-off between information loss and disclosure risk. In Chapter 1 we introduced the score measure, which computes this trade-off as the arithmetic mean of both measures. The score is one of the most used measures to evaluate any protection method. Therefore, we define our fitness function the same way:

$$F(c^i) = \frac{IL + DR}{2},$$

where IL stands for information loss measure and DR stands for disclosure risk measure. Like in the score case, GOMM aims at minimizing this function; therefore, lowest fitnesses imply better solutions.

However, because of the frequent use of the fitness function during the evolutionary process, we cannot use such a high computationally intensive function. Thus, we use some simplifications for reducing the complexity of the original IL and DR measures in our fitness function $F(c^i)$.

Several proposals have been used in the literature in order to calculate the information loss. Depending on the alternatives, they take into account several general parameters of the data distribution such as the average vector, covariance matrix, variance vector or correlation matrix.

Since the goal of microaggregation is to minimize the SSE, which is a specific information loss measure for any k-anonymity model, we use it as the IL measure. However, the SSE itself is not suitable for the $F(c^i)$ formulation since it is not upper bounded. The most common way to normalize the SSE range into the [0..1] interval is to divide it by the sum of squares total SST of the original dataset R. This approach is used in many other works as [3, 6]. The SST is defined as

$$SST = \sum_{i=1}^{N} \sum_{r_j \in N_i} (r_j - \bar{r})^T (r_j - \bar{r}),$$

where $\bar{r}$ is the centroid of all the original records. Note that, the SST must be computed only once during the execution since its value is constant. Then, the IL component in the fitness function of our genetic algorithm is computed as

$$IL = \frac{SSE}{SST} \cdot 100,$$

in this way, IL range is between $[0, 100]$.

In order to compute DR, we explained in Chapter 1 that two approaches are considered. The first considered disclosure risk measure is the Interval Disclosure Risk ID which is the average percentage of protected values falling into an interval around their corresponding original values. Usually, the interval length is a parameter defined by the user. In our case, the interval is defined as $[(r_j - r_j \cdot 10\%), (r_j + r_j \cdot 10\%)]$.

The second one is the entity disclosure risk and , basically, there are two families of such methods, on the one hand, those based on (in)conditional probabilities and, on the other hand, those based on distances calculations. The former (probability based family) are too inefficient to be used inside a cost function of a genetic algorithm. For this reason, we only use distance-based methods as it was done in [16] for similar reasons. This is not a problem since distance-based methods outperform probabilistic ones for numerical attributes [5].

Generally, it is assumed that the intruder has several different sets of non-confidential quasi-identifiers and the risk is computed as the average risk of all those sets. However, for reducing the execution time of the record linkage process, we consider only the worst, and unrealistic, scenario, *i.e.* when the intruder has access the complete original dataset R (all records r and non-confidential quasi-identifiers a_{nc}) and she links them with the protected dataset R'. In this case, if the closest record to a known record r in R is the corresponding one r' in R', we assume that the intruder is able to find the correct link, and then, she is able to breach the privacy of such data owner. Again, as we are interested in a value fitted between $[0, 100]$,

Distance Linkage disclosure (DLD) is computed as

$$DLD = \frac{links}{|R|} \cdot 100,$$

where $links$ is the total number of correct links achieved and $|R|$ is the number of records of the dataset R.

To calculate the number of links, the intruder has to compute the Euclidean distance between the record r_i she wants to deanonymize and all the protected records r' and then check if the closest protected record is r'_i to establish a correct link. This means that in order to compute the distance-based disclosure risk the Euclidean distance function is called m^2 times, being m the number of records.

In GOMM's cost function, we implemented an improvement to speed-up the whole process. The distance is initialized with the value of the link distance, that is, with the distance between r_i and r'_i. Then, the search algorithm looks for a smaller distance and, when found, the search loop is stopped. In conclusion, every time there is not a link the Euclidean distance function will be called less times.

Finally, the overall DR is computed as

$$DR = (0.5ID + 0.5DLD)$$

3.4 Genetic Operations

The evolution process of any genetic algorithm is based in three operations. The number of crossover and mutation operations performed in each generation can be manually modified to adapt the algorithm to different situations and allowing the study of its effect. Following, we describe the crossover operation, the set of mutation operations and the selection operation designed and implemented in GOMM.

3.4.1 Crossover

The aim of the crossover operator $\phi(c^{p1}, c^{p2})$ is to generate new members by combinig properties from different chromosomes in the current population. Two parent chromosomes c^{p1} and c^{p2} are selected randomly from the population and two new child chromosomes c^{c1} and c^{c2} are produced containing information from both parents.

Figure 3.3 shows the three steps of our crossover operation $\phi(c^{p1}, c^{p2})$, which is an adaptation of the Grouping Genetic Algorithm (GGA) crossover [8]. In GOMM's crossover, first of all, two crossing sites are selected from the group part of both parents (1). Then, the groups contained in the crossing site of the first parent c^{p1} are injected at the group part of the first child c^{c1}, along with the attributes that belong to those groups (2). The rest

of the attributes are grouped as they were in the second parent c^{p2} (3), so the resulting child c^{c1} has inherited properties from both parents. For the second child c^{c2}, the process is repeated exchanging the role of the parents.

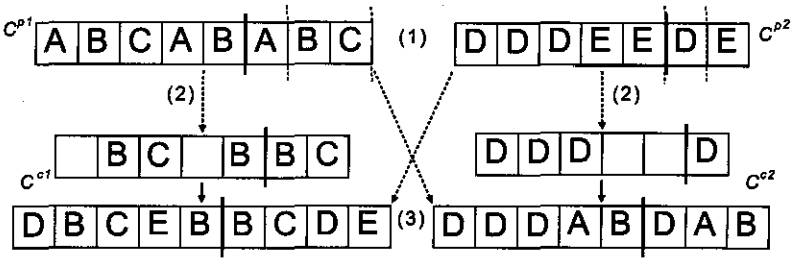

Figure 3.3: Steps of the crossover operation.

The main difference between the crossover proposed by the GGA and the one implemented in GOMM is in the order of the different steps. GGA's crossover first copies the whole parent chromosome c^{p1} into the child c^{c1}. Then, it injects the crossing section of the second parent c^{p2} in the group part of c^{c1}. Therefore, a fourth step is required to check that the old groups copied from c^{p1} are not empty because, when injecting the crossing section from c^{p2}, the attributes that belong to those groups are also injected. In conclusion, GOMM's crossover is implemented in a simpler way, although the resulting offspring of both crossover operations is the same.

In the example shown in Figure 3.3, the parent chromosomes do not share any group identifiers and these identifiers are directly injected in the child chromosomes. This is done to show in an easier way which groups are inherited from each parent. However, GOMM's crossover operation implements a renaming process of the groups as they are injected in the child. Thus, if the parent chromosomes share any group identifier, the resulting child chromosomes will not have coherence problems. Figure 3.4.a illustrates the problem when not renaming the groups.

The renaming process of the GOMM's crossover operation is depicted in Figure 3.4.b. Every time a group, along with the respective attributes, is injected in the child, its group identifier is changed, starting with the identifier A, then B and so respectively. Beyond avoiding the generation of incoherent childs, this method simplifies the injection process as no control of the identifier is required.

3.4.2 Mutations

As crossover operations only combine existing properties from the current population, a way to introduce new information is needed in order to explore the whole search space; *i.e.*, ensuring that any possible chromosome in the

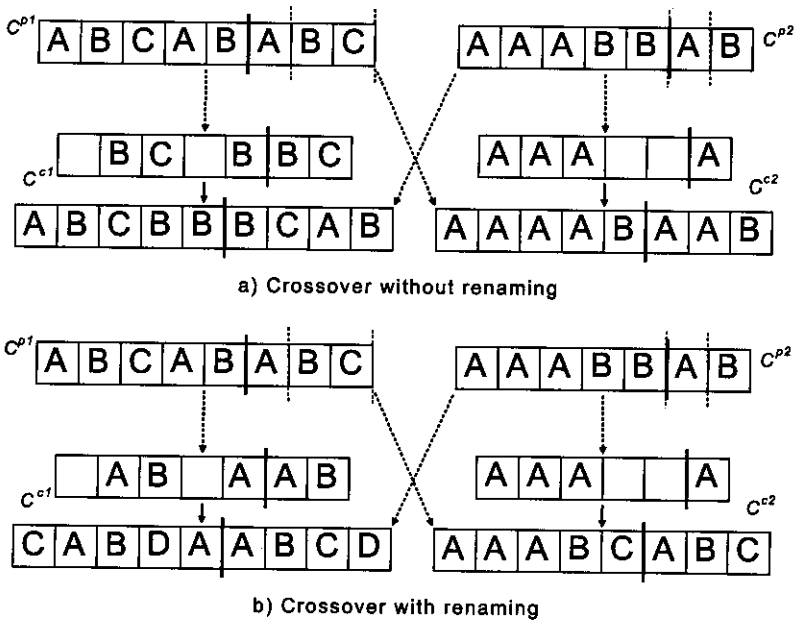

Figure 3.4: Group renaming of the crossover operation.

search space can be generated. Mutation operations $\varphi(c^p)$ proceed by performing random modifications to a chromosome c^p, thus generating a new member c^c with some characteristics not present in the current population.

We propose five different mutations in GOMM, which are represented in Figure 3.5; three working with the group part of the chromosome, and two with the object part. The former allow the exploration of distant parts of the search space, while the latter work in a finer manner.

- **Group Create(φ_{GC})** builds a new group of attributes randomly, which is injected in the child c^c. The rest of attributes are grouped as they were in the parent c^p. Figure 3.5.a shows a basic example of its procedure. It is the most aggresive of all the mutations. In an extreme case a single φ_{GC} operation can transform any chromosome c^p to the one that contains only one group. The goal of φ_{GC} is to make radical transformations to the members in the population to introduce diversity. However, it is in general very disruptive and may cause the new chromosomes to contain lethal traits that lead them to the immediate elimination. Like in the crossover operation, a renaming method is implemented in Group Create.

- **Group Eliminate(φ_{GE})** selects randomly a group from the Group Part of the parent c^p and distributes all its attributes between the

remaning groups, also randomly. As a result, the child c^c has a group less than its parent c^p, except in the base case that the parent has all the attributes in a single group, then a clone is generated. The steps followed by Group Eliminate are depicted in Figure 3.5.b.

- **Group Split**(φ_{GS}) chooses a group at random and splits it into two different groups with the same number of attributes (if possible). The resulting child c^c has a group more than its parent c^p, except in the cases where the group that is splitted is a singleton. The Group Split example illustration is found in Figure 3.5.c

- **Element Swap**(φ_{ES}) works with the object part of the chromosome; two attributes are selected at random and their groups are swapped, so the group part of the child c^c is identical to that of the parent c^p. Figure 3.5.d shows the functioning of Element Swap.

- **Element Move**(φ_{EM}) selects randomly an attribute from the object part of the parent c^p and moves it to a different existing group. The resulting child c^c has the same number of groups than his parent c^p or a group less if the element moved belonged to a singleton. Finally, an example of a simple Element Move operation is presented in Figure 3.5.e.

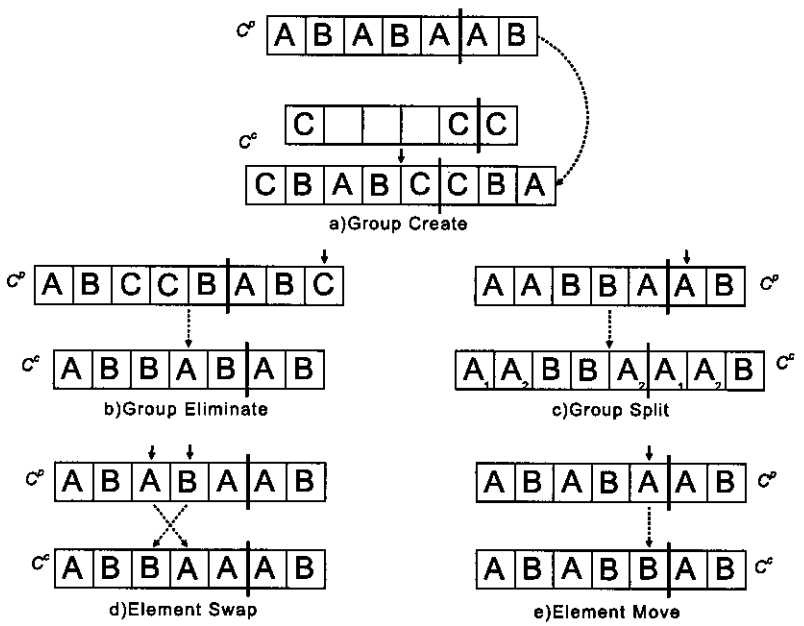

Figure 3.5: Steps of GOMM's mutation operations.

3.4.3 Selection

In order to preserve the number of members in the population and favor evolution, at the end of every iteration the chromosomes that will survive to the next generation are chosen. GOMM selects the best ones; *i.e.* those solutions with the lowest fitness. Formally, if we have two chromosomes c^1 and c^2 with an associated fitness $F(c^1)$ and $F(c^2)$ and $F(c^1) < F(c^2)$, then c^1 is more likeable to survive.

When all the C crossovers and the M mutations are performed, an offspring vector is filled with all the $2C + M$ new generated chromosome along with its associated fitness. Before applying the selection method, this vector is sorted according to the fitness value so that the best fitted chromosomes are placed in the top part of the vector.

GOMM's selection method acts the same way as the merge function of the merge sort algorithm. Given two sorted vectors, the offspring vector containing $2C + M$ chromosomes and the current population vector with P chromosomes, our selection algorithm outputs the next generation population vector with P and sorted by their associated fitness.

Algorithm 3: Selection

```
   Data: N: current population, O: offspring
   Result: N': next population
 1 begin
 2     while (length(N') < P) do
 3         if first(O) < first(N) then
 4             append first(O) to N'
 5             O = rest(O)
 6         else
 7             append first(N) to N'
 8             N = rest(N)
 9         if length(O) = 0 then
10             break
11     if length(N') < P then
12         while (length(N') < P) do
13             append first(N) to N'
14             N = rest(N)
15 end
```

Algorithm 3 shows GOMM's selection pseudocode. Two pointers are assigned to the first position of the input vectors, which are chromosomes with an associated fitness. Then, the pointed chromosome with a lower cost is copied to the next population vector, and the corresponding pointer

assigned to the next position. This process is repeated until the next population vector is filled with P chromosomes. It is worth pointing out that the offspring vector could be emptied before filling the next population vector. If this happens, the remaining empty positions of the next population vector are filled with chromosomes from the current one.

3.5 Performance Analysis

Finding optimal solution to complex high dimensional problems using genetic algorithms often requires very expensive fitness function evaluations. One single function evaluation may require several hours to compute a solution fitness. This happens in the attribute grouping problem of multivariate microaggregation since computing the classic score is very complex. This is why we propose an approximate calculation of this measure, to make the computation of the algorithm feasible.

However, when dealing with large datasets, our genetic approach may be too slow. In this section, we study GOMM's code to try to find the appropiate functions to be improved in order to increase performance. One of the measures studied has been implemented in a version called D-GOMM. The other proposals are out of scope of this project and are only stated.

First of all, we used the open source profiler Gprof to get a detailed view of all GOMM's functions. The results are shown in Table 3.1 and have been obtained performing 25 crossovers and 50 mutations (10 of each kind) for generation. Thus, 100 new chromosomes are generated every iteration during a total amount of 100 generations.

The first thing we notice is that the number of executed Euclidean distance functions is 714,112,812 whereas it should be executed 1,472,880,000 times. An initial population of 200 members plus 100 generations with 100 new generated members each, gives 10,200 executions of the cost function. For every cost function, 380^2 measures of the Euclidean distance were required, because the DR measure searches the closest protected record for every disclosed one and the Water-treatment dataset has $m = 380$ records. This means that the measure proposed in section 3.3 to reduce the execution time of computing DR is working as expected because the algorithm performs half of the Euclidean distance function executions.

The profile also shows that the 99% of the execution time is spend by three functions: the MDAV algorithm, the Euclidean distance to compute DR, and the whole computation of ID. All these methods are related to the cost function. The first conclusion we extract from this profile is that it is no sense to try to improve the other functions, since they represent less than the 1% of the execution time. This includes the genetic operations along with the IL computation.

In view of this profile, we propose two possible optimizations to the

% time	self	calls	self $\frac{sec}{call}$	total $\frac{sec}{call}$	function name
66.86	580.36	10200	0.06	0.06	MDAV
27.33	237.21	714112812	0.00	0.00	euclideanDistance
4.94	42.89	102000	0.00	0.00	computeSingleID
0.85	7.36	10200	0.00	0.03	score
0.02	0.19	1	0.19	0.19	initOriginalData
0.00	0.04	171000	0.00	0.00	add2Child1
0.00	0.01	1000	0.00	0.00	groupCreate
0.00	0.00	95000	0.00	0.00	add2Child2
0.00	0.00	10200	0.00	0.09	cost
0.00	0.00	5351	0.00	0.00	addChild2
0.00	0.00	5257	0.00	0.00	addChild1
0.00	0.00	2500	0.00	0.00	crossover
0.00	0.00	1000	0.00	0.00	elementMove
0.00	0.00	1000	0.00	0.00	elementSwap
0.00	0.00	1000	0.00	0.00	groupEliminate
0.00	0.00	1000	0.00	0.00	groupSplit
0.00	0.00	100	0.00	8.51	generateNewPopulation
0.00	0.00	1	0.00	17.02	populate
0.00	0.00	1	0.00	0.00	sort

Table 3.1: Flat profile of GOMM algorithm using the Water-treatment dataset.

GOMM algorithm:

- *Dynamic control.* In a genetic algorithm it is normal to expect that some operators stop doing useful work after a given generation. If a monitoring mechanism is capable of detecting the generation this happens, the execution of this genetic operation can be stopped. This directly implies that the number of new genereted chromosomes decreases, hence, the number of fitness evaluations also decreases.

- *Parallelization.* During the GOMM's execution there is an interesting point where to open parallelism. This is when the fitness of the new generated chromosomes is calculated. The cost evaluation of a child chromosome is independent from the fitness evaluation of the other children. Therefore, the execution of several threads would not produce conflicts.

The second optimization is out of scope of this project and has not been implemented. The first one is included in a version of our genetic approach called D-GOMM.

The procedure of D-GOMM is identical to the one of GOMM, but a control function is executed every generation. After generating the offspring

and selecting the chromosomes that will survive to the next generation, the control function checks if among the discarted offspring there are all the child chromosomes created by a single operator. If this happens during five successive generations, this genetic operation is not performed anymore.

The performance comparision between GOMM and D-GOMM can be found in Chapter 5, along with the results obtained in this project.

Chapter 4

GOMM Analysis

Now we perform a comprehensive analysis of the genetic operations used by GOMM. The experiments realized were proposed in [10] in order to understand any genetic programming based optimizer, but they can be applied to genetic algorithms like GOMM as well. The aim is to evaluate the effect that each operation has over the population for each generation of the execution. This analysis is useful to know the behaviour of every operator in order to study new techniques to improve the optimizer's performance. Five aspects have been studied: the number of chromosomes discarded without being used, the average chromosome lifetime, the operator's efficacy and efficiency, and the best cost evolution; for every genetic operation used alone or combined with other genetic operations.

4.1 Testing Scenario

The tests have been performed with real data extracted from two datasets available in the Internet. The first one is the Water-treatment dataset, extracted from the UCI repository [11], which contains 38 attributes corresponding to 380 entries of records. The second one, called Census, it was extracted using the Data Extraction System of the U.S. Census Bureau [20]. A complete description about the details of the construction of this second dataset can be found in [7]. This dataset contains 1080 records and 12 attributes.

We executed GOMM with different values for the parameter k of the MDAV algorithm. In this chapter, we show the analysis for clusters of $k = 25$. This value is very common in multivariate microaggregation since it offers a good trade-off between information loss and disclosure risk. Also, to completely understand the behaviour when the record clusters are large, we include a brief analysis of the genetic operations when $k = 100$ in Section B.2.

The algorithm was executed during 100 generations, using 200 members

for the population and generating 100 new members for every iteration. Depending on the experiment, only a single genetic operation has been used in order to study its effect over the population. When the results have been obtained by combination of the genetic operations, GOMM has performed 25 crossover operations and 50 mutation operations (10 of each type) per generation.

4.2 Utilization

First we study the utilization, which is the amount of useful work that each type of operator can produce. If a genetic operation introduces interesting configurations in the population, the new members generated by this operation have a higher probability to survive and to be chosen on future generations. One way to analyze the capability of a genetic operator to introduce good properties into the population is to study the number of members discarded without being used. That is, the chromosomes that are not used to generate new chromosomes in a crossover or a mutation operation.

It is important to notice that the utilization results for the last generations are skewed since the chromosomes generated live shorter. For instance, all the chromosomes created in generation 99 will be discarded without being used because the algorithm ends in this generation.

Water-treatment dataset with clusters of 25 records

The charts presented in Figure 4.1 show the utilization results of every genetic operation when used alone. The usefulness of φ_{GC} improves in the first generations, but after generation 20 the number of discarded chromosomes without being used almost reach the total number of chromosomes created. This operator works like a simulated annealing process: as φ_{GC} can create large groups, in the first generations it explores very distanced points in the search space. When the value of the k parameter is low, solutions have a better quality when the number of groups of attributes is not very large; thus, φ_{GC} rapidly reaches a solution with few groups. Then, the groups created by the operation can only increase the number of groups in the solution by one, and in very rare occasions the number of groups is reduced since the solution has already few groups. Hence, Group Create cannot explore closer solutions.

The plots for φ_{GE} and φ_{GS} are very similar as they gradually lose potential and in the latter generations they only do useless work. These operations explore close points in the solution space since they only add or eliminate one group. If the dataset has a lot of attributes, it is coherent that Group Eliminate and Group Split end in a local maxima solution. Nevertheless, as the initital population has a significant number of groups, φ_{GS} has less

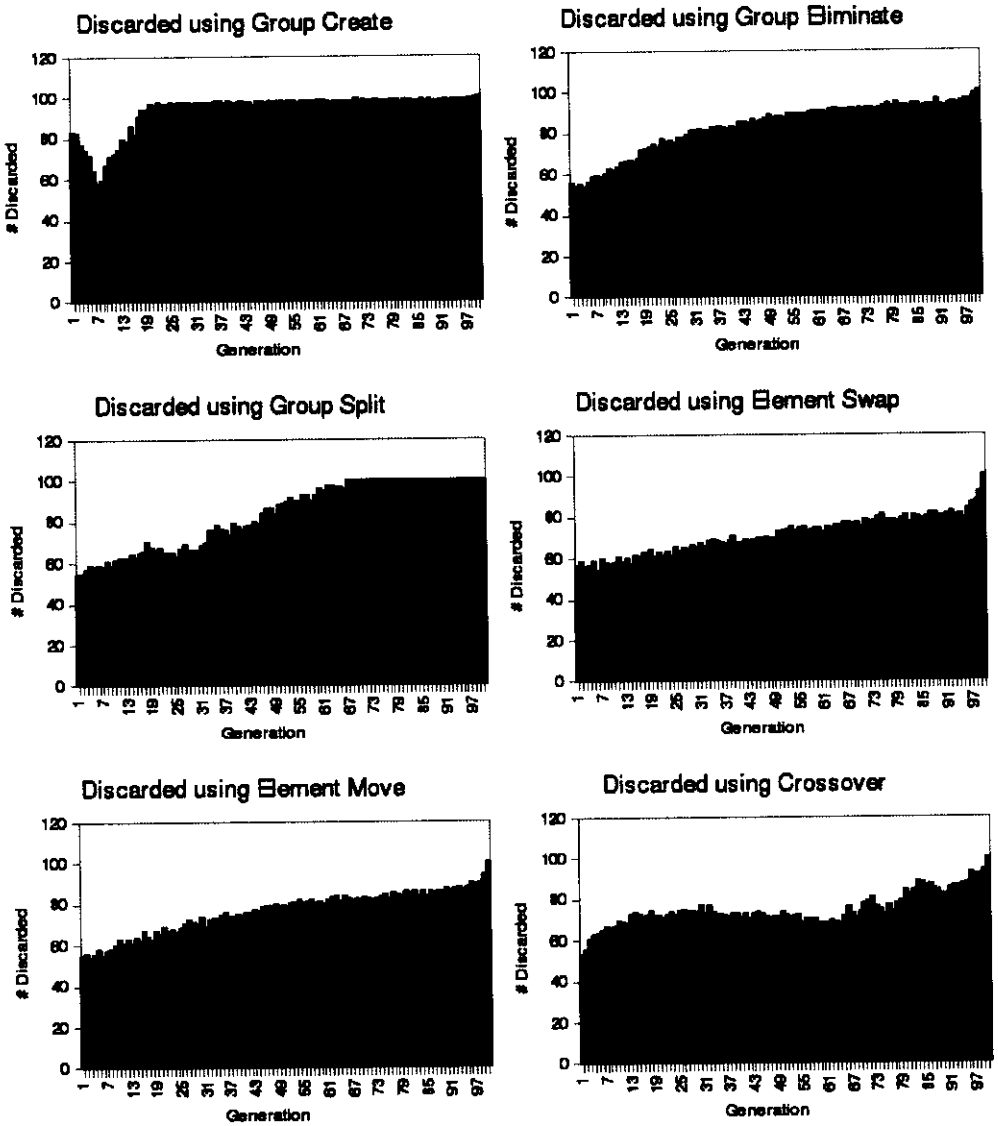

Figure 4.1: Utilization of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records.

space to explore. This is why this operation discards all the chromosomes it produces after generation 60.

Regarding φ_{ES} and φ_{EM}, they explore even closer points in the search space than φ_{GE} and φ_{GS} as they do not alter the number of groups of the solution (except when φ_{EM} moves an element which is a singleton). Even though they work in a finer grane, they can explore more solutions than φ_{GE} and φ_{GS} as the possible attribute groupings accessible fixing the number of groups is larger than increasing or decreasing the number of groups.

Finally, the crossover keeps the number of discarded chromosomes without being used around the 70%.

Combined execution

The plots in Figure 4.2 show the average number of chromosomes discarded without being used in a later operation for each genetic operation when combined with other genetic operations. Note that φ_{GC}, φ_{GE} and φ_{GS} are doing useless work more or less after generation 25 since 100% of the generated chromosomes are discarded without being used. In contrast, φ_{ES} and φ_{EM} gradually lose potential, but their best behaviour is shown between generations 20 and 30, where less than half of the chromosomes they produce are discarded without being used.

The results for the crossover operations show that the percentage of chromosomes discarded without being used is always around 75%, being a little lower in the first generations.

Census dataset with clusters of 25 records

The utilization results obtained when using the Census dataset are vey similar to the Water-treatment ones. The plots can be found in Section B.1. The only significant difference is that the charts stabilize faster with the Census dataset, because it has a lower number of attributes.

Water-treatment dataset with clusters of 100 records

When the k value is high, the information loss of all the possible solutions increases because more records are cluster together. Therefore, the genetic operations have a very different behaviour, as showed in Section B.2. Regarding utilization, the main difference between the executions with $k = 25$ and $k = 100$ can be found in the group-oriented mutations, which discard more chromosomes during the whole execution process. However, they are able to introduce changes in the last generations, which did not happen when $k = 25$.

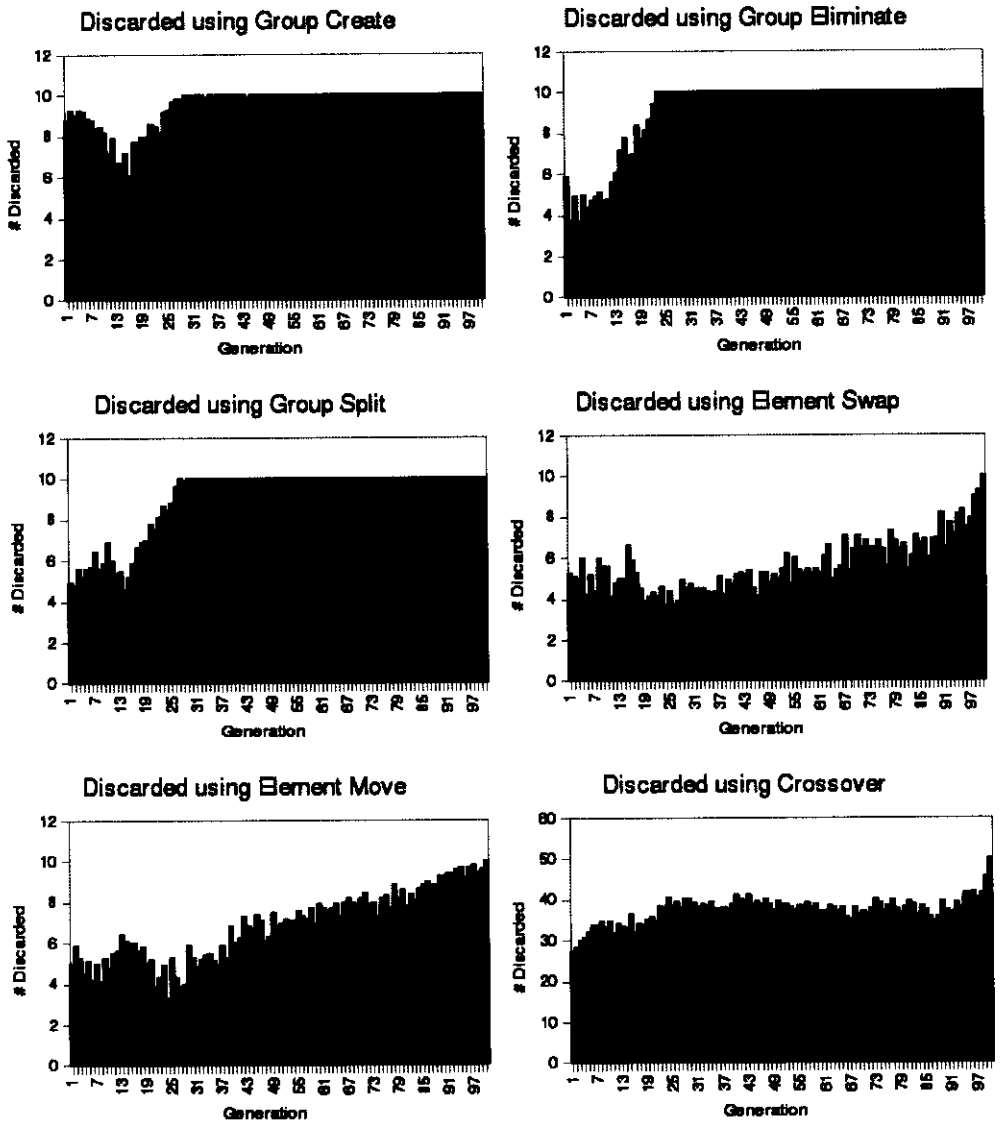

Figure 4.2: Utilization of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records when used combined.

4.3 Average Lifetime

Another way to evaluate the effect of a genetic operator is to analyze the average lifetime of the new generated chromosomes, which is the number of generations they survive. Longer lifetimes imply a higher probability for a given chromosome to be used in the next generations.

As happened in the utilization plots, the average lifetime is skewed in the last generations. Obviously, if a chromosome is created in the last generation, its lifetime is zero because the algorithm ends in the same generation.

Water-treatment dataset with clusters of 25 records

The average lifetime plots for every genetic operation when used alone are depicted in Figure 4.3, which shows the lifetime of the attribute grouping configurations grouped by the generation in which they were created. The first chart shows the average lifetime of the chromosomes generated by φ_{GC}. We observe that the chromosomes created in the first generations live a lot more than those created afterwards. This indicates that, after reducing the number of groups of the solution, Group Create has difficulties in evolving the population, that is, the number of succesful transformations introduced by φ_{GC} when the solutions have a reduced number of groups is very low.

The φ_{GS} plot shows two interesting features. Firstly, the chart presents a peak between generation 35 and 45 similar to the one produced by φ_{GC} and, in the successive generations, lifetime rapidly decreases until all the chromosomes die at the same generation they are created. The explanation to this phenomenon is that φ_{GS} evolves the population until it reaches a local maxima solution. At this point, it only generates worse chromosomes or clones, which are the ones that survive during the last generation.

The crossover operation and the finer grane mutations φ_{ES} and φ_{EM} along with φ_{GE} have very similar charts, keeping the average lifetime of their generated chromosomes always around 2. This indicates that these operators slowly, but continuously, introduces interesting properties in the population, and the new generated chromosomes substitute the ones generated more or less two generations ago.

Combined execution

The information on the average lifetime for each genetic operation when used combined with other operators is presented Figure 4.4. Again, the plots show that φ_{GC}, φ_{GE} and φ_{GS} carry out unnecessary operations after generation 25 as the average lifetime of all the chromosomes created by these operations is zero, that is, they die as soon as they are created. On the other hand, the other two mutations, φ_{EM} and φ_{ES} , improve their performance after generation 20 since the chromosomes they create live longer. The

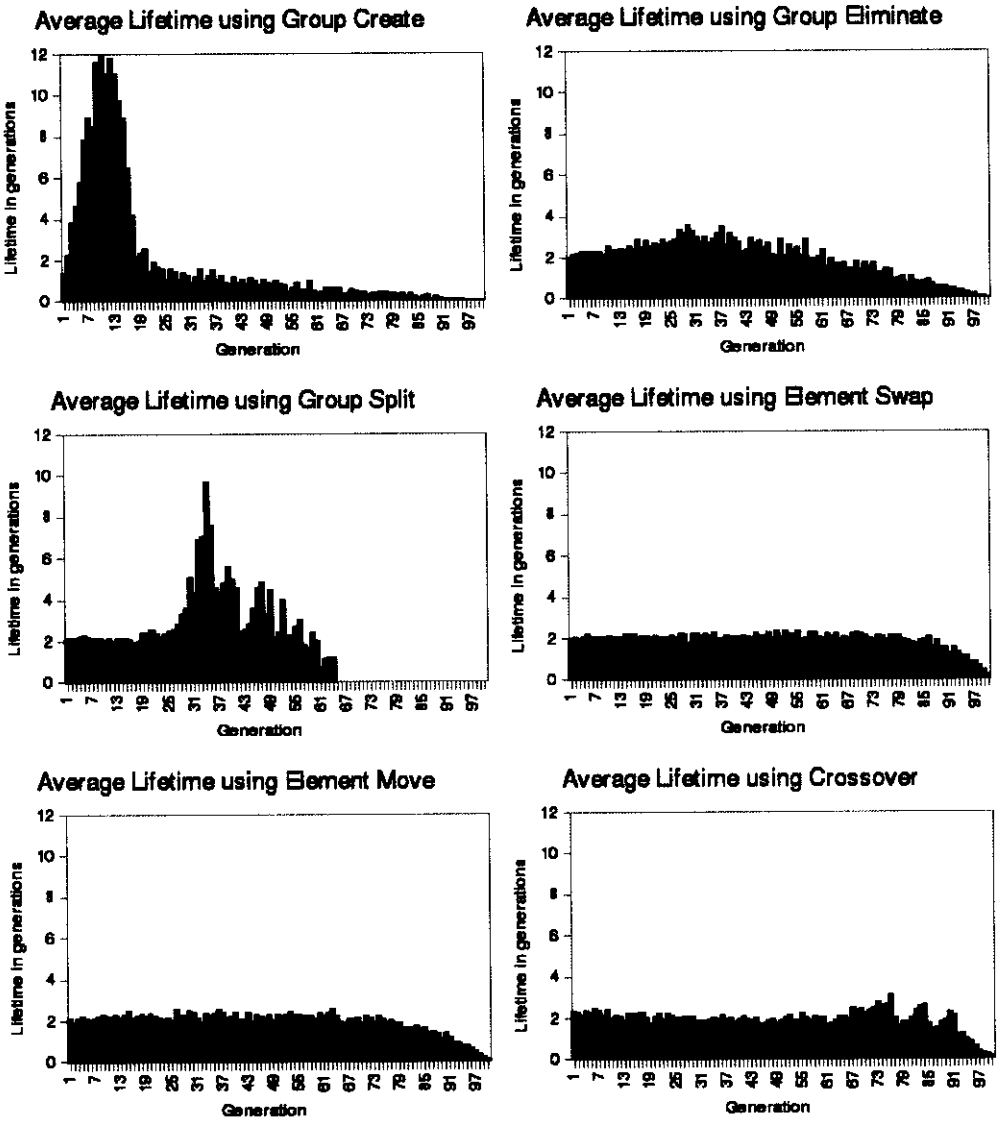

Figure 4.3: Average Lifetime of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records.

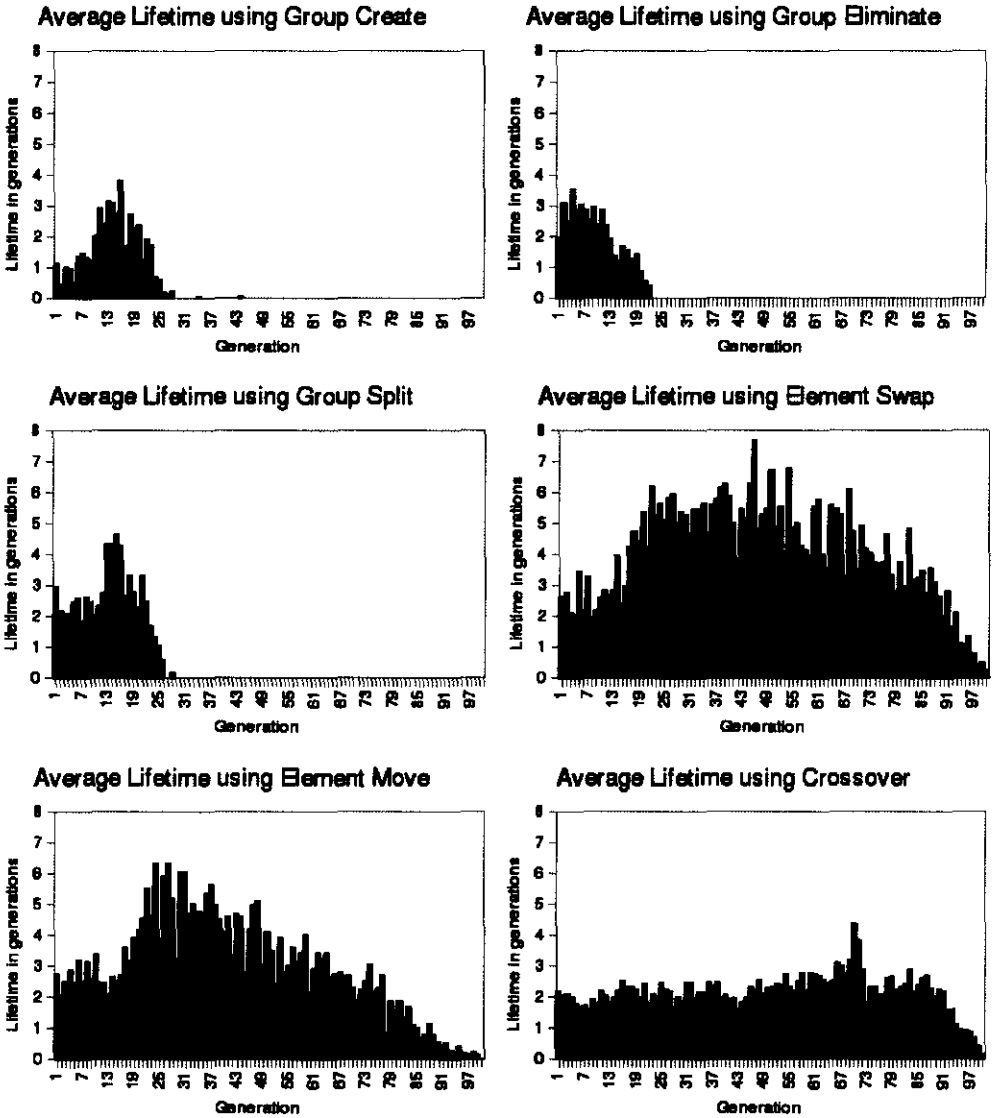

Figure 4.4: Average Lifetime of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records when used combined.

average lifetime of the chromosomes produced by φ_{EM} does not decrease until generation 50, and in the case of φ_{ES} until generation 70.

Regarding the crossover operation, we can see that average lifetime is always very stable in values between 2 and 3.

Census dataset with clusters of 25 records

The average lifetime charts for the Census dataset are found in Section B.1. Again, when executing GOMM with the Census dataset, the plots are very similar to the ones generated when using the Water-treatment dataset. However, the group-oriented mutations (φ_{GC}, φ_{GE} and φ_{GS}), start performing useless work a few generations before.

Water-treatment dataset with clusters of 100 records

Section B.2 presents the average lifetime plots for the Water-treatment dataset when $k = 100$. Like in the utilization analysis, the important changes are in the behaviour of the group-oriented mutations. When $k = 25$, all the chromosomes generated by these mutations after generation 25 were discarted in the same generation they were born. Now, these operators present more regular charts: the average lifetime of the members produced by φ_{GC} is very low because this operation is very destructive when dealing with chromosomes that have a higher number of groups. In the case of φ_{GE} and φ_{GS}, their plot is very similar to the one presented by the crossover operation, which means that they behave in a more conservative way.

4.4 Efficacy

The utilization and the average lifetime of the chromosomes give us a first impression of the amount of useful work that a genetic operation is producing during all the algorithm execution. However, these two approaches do not show if the operations are really introducing good properties into the population; that is, if the new generated chromosomes have a better cost than their parents, in the case of mutation operations; or a better cost than the average fitness of both parents, in crossover operations. This measure is called efficacy.

The efficacy plots presented show the number of improved chromosomes along with the number of worsened ones for every generation. It seems reasonable to assume that the addition of the two parts should be the number of chromosomes generated for every operation in a certain generation (10 for each mutation and 50 for the crossover when used combined and 100 for every operation when used alone). Nevertheless, this is not true when the generarated chromosomes contain clones as they have the same cost of their parents.

Water-treatment dataset with clusters of 25 records

The genetic operation efficacy charts when used alone are shown in Figure 4.5. The first thing we notice is that the φ_{GS} chart is thinner that the other operations, meaning that Group Split generates a lot of clones when the number of groups of the solution is high. After generation 50, only worse chromosomes or clones are generated as we pointed out before.

φ_{GC} produces better chromosomes in the first generations and from generation 20 in very few cases improved solutions are generated. This strenghtens the theory that Group Create does not favor evolution when the number of groups of the solutions is low.

The rest of the mutations φ_{GE}, φ_{ES} and φ_{EM} , present a similar chart patron: the efficacy slightly decreases during the evolution process.

Regarding the crossover operation, all the generations have similar improvements during the evolution process. However, the decrease of the worsements in the last generations point that the diversity of the population decreases. This happens because crossover combines existing properties of the population, so no new properties are introduced.

Combined execution

Figure 4.6 shows the efficacy plots when the genetic operations are used in combination among them. The first thing we notice is that, again, φ_{GC}, φ_{GE} and φ_{GC} only generate worse chromosomes from generation 25. However, the only operation that loses potential gradually is φ_{GE}, as φ_{GC} and φ_{GS} show a peak near generations 18-19. This is because the average number of groups of the first population is large, and creating small groups or spliting them does no generate better chromosomes. However, as the number of groups decreases due to the creation of large groups and the elimination of others, these operations start working better. In the case of φ_{ES}, the plot shows the following effect: this is an operation that generates a lot of clones, but until the end of the execution is capable of generating improved chromosomes. This also happens in φ_{EM}, where we also notice that near generations 16-17 the number of clones generated increases. The only case that φ_{EM} produces clones is when the parent chromosome has a single group, which is exactly when φ_{GS} and φ_{GC} start working well.

The crossover results show that after generation 20 the number of improvements is very constant, standing in the 5% of the chromosomes generated.

Census dataset with clusters of 25 records

The most important difference in the efficacy plots between using the Census or the Water-treatment dataset is that, in the first generations for the Census dataset, all the genetic operations achive a higher number of improvements.

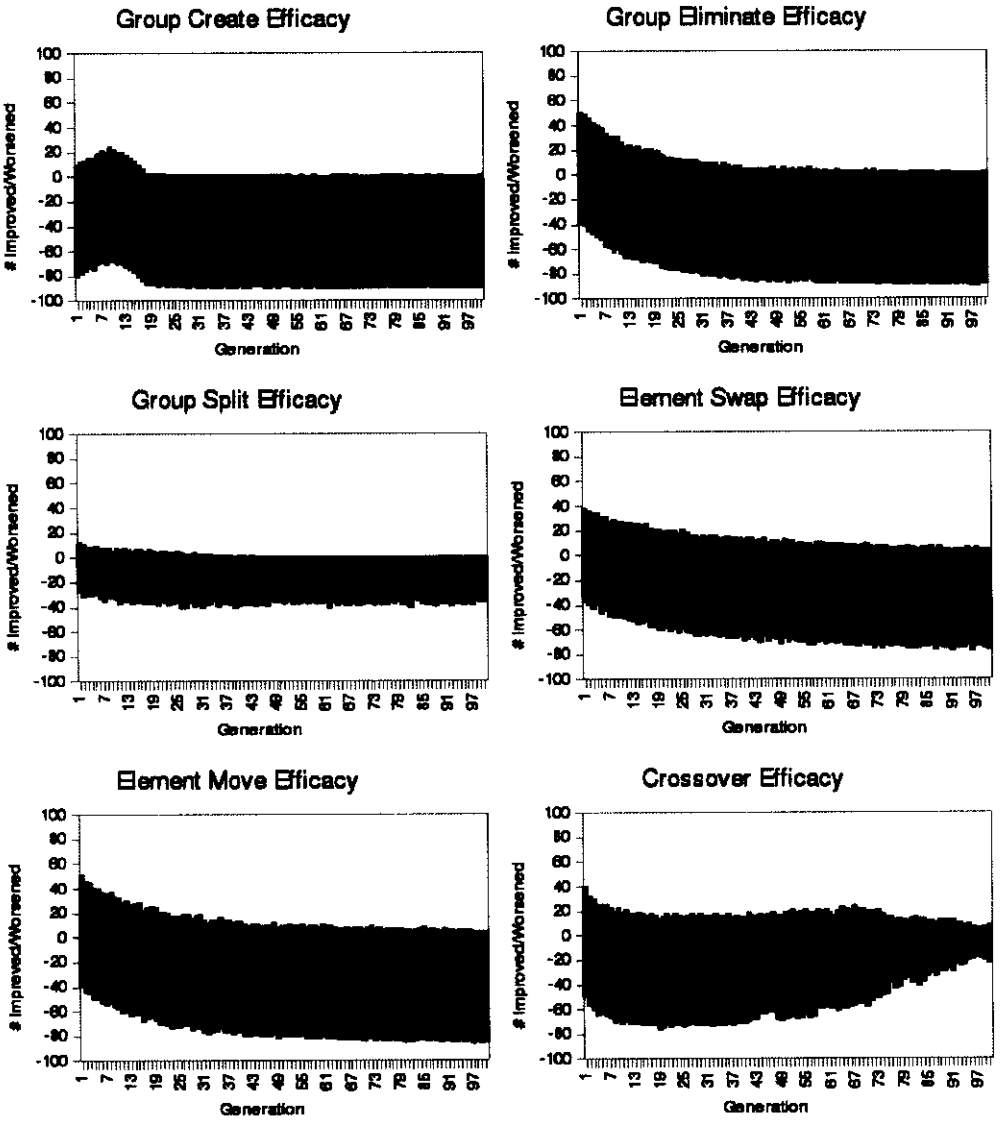

Figure 4.5: Efficacy of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records.

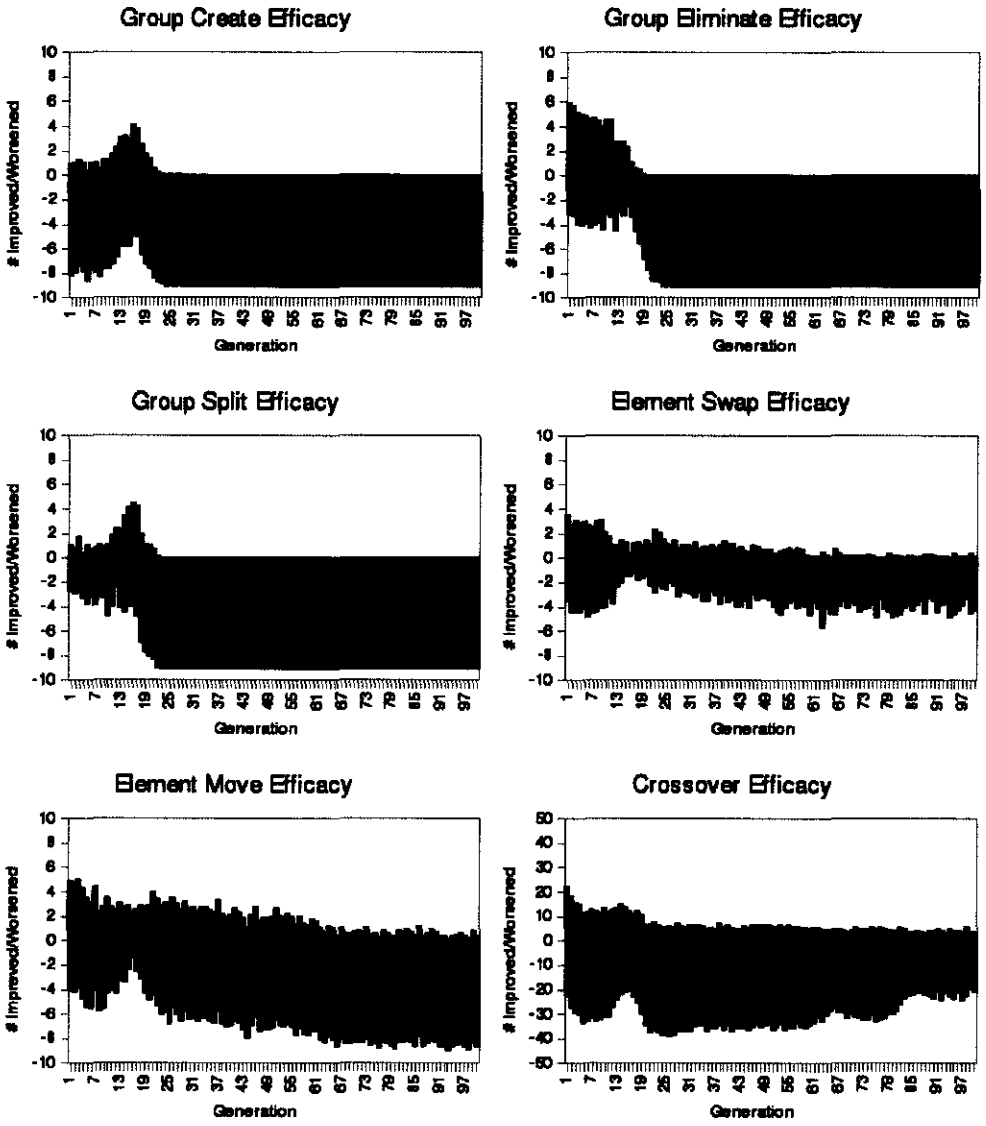

Figure 4.6: Efficacy of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records when used combined.

Nevertheless, this improvements last a lower number of generations. For instance, even the crossover operation only produces clones or worse offspring in the last generations. This is so because of the fewer number of attributes of the Census dataset, which also causes the search space to be smaller. The efficacy plots for the Census dataset can be found in section B.1.

Water-treatment dataset with clusters of 100 records

The efficacy plots of the Water-treatment dataset with $k = 100$ can be found in Section B.2. Along with the group-oriented mutations, the crossover operation also presents changes with respect to the efficacy charts when using a k value equal to 25. The number of improvements of the group-oriented mutations in the first generations is lower, even though they last longer. The increasement of the k parameter causes the information loss of all the possible solutions to increase too, thus, the solutions with a small number of groups are not preferred anymore. The case of the crossover operation is very interesting because it presents a decreasing pattern, which reveals that the recombination plays an important role in these conditions.

4.5 Efficiency

Finally, even though the previous analysis provide us with an approximate picture of the behavior of genetic operations, it does not directly reveal how much better or worse is the cost of the new generated chromosomes. For this reason, we evaluate the efficiency of the operations by immediately calculating the percentage of improvement or worsement of the chromosome costs after the application of the genetic operation.

To calculate the average percentage of maximum improvement and worsening for mutation operations we use Formula (4.1). c^c is the child chromosome and c^p the parent chromosome. For crossover operations we use (4.2) and (4.3) where c^{p1} and c^{p2} are the parents. The idea behind the equation for the crossover operation is to calculate whether the new chromosome is better than the average cost between both parents.

$$\%_M = \begin{cases} \frac{cost(c^c)}{cost(c^p)} \cdot 100 & \text{if } cost(c^c) \leq cost(c^p) \\ -\frac{cost(c^p)}{cost(c^c)} \cdot 100 & \text{if } cost(c^c) > cost(c^p) \end{cases} \tag{4.1}$$

$$r_{imp} = \frac{2 \cdot cost(c^c)}{cost(c^{p1}) + cost(c^{p2})} \tag{4.2}$$

$$\%_C = \begin{cases} (1 - r_{imp}) \cdot 100 & \text{if } r_{imp} \leq 1 \\ (\frac{1}{r_{imp}} - 1) \cdot 100 & \text{if } r_{imp} > 1 \end{cases} \tag{4.3}$$

In the efficiency plots we present the average efficiency results for each mutation operation, showing the average improvement and the average worsement along with the maximum and the minimum for every generation.

Water-treatment dataset with clusters of 25 records

Figure 4.7 shows the efficiency charts of the genetic operations when used alone. It is important to notice that Group Create performs in a different scale than the other operators. As φ_{GC} presents improvements up to 20%, the other genetic operations do not overcome 5%. That is because of the hability of Group Create to explore distant points in the search space: in the first generations it rapidly reaches promising zones and then it needs the other operations to explore them as it is not capable to improve. On the contrary, the other genetic operations cannot perform such large jumps in the search space and they normally end in local maxima solutions.

Combined execution

In Figure 4.8 we observe that φ_{GC} and φ_{GE} work very well in the first generations, with average maximum efficiency values around 10%. This is because the members of the first population are created randomly and they have a lot of groups, and reducing the number of groups reduces also the disclosure risk, between generation 10 and 20 φ_{GS} does some useful work because new chromosomes with a low number of groups have appeared in the population due to the effects of φ_{GC} and φ_{GE}. These three operations, as seen before, stop doing useful work after generation 25 as they only produce worse chromosomes. In the case of φ_{ES} and φ_{EM}, we notice that their efficiency values are lower than the other mutation operations, standing around 2% in their phase of best behaviour. This happens because these operations work with objects rather than groups, and the chromosomes they generate are very similar to the parent chromosome they used. That property is the one that allows φ_{ES} and φ_{EM} to continue doing useful work after generation 25, when the optimum number of groups has been found and φ_{GC}, φ_{GE} and φ_{GS} do useless work.

The crossover efficiency results show that in the first generations, the average maximum efficiency shows a peak and afterwards it quickly decreases, showing that the new chromosomes are worse or have the same cost as the parents. The explanation to this phenomenon is that, in the first generations, chromosomes with a large number of groups are crossed with others with few groups, so the resulting children have less groups than the first parents. In the next generations, when the number of groups of the members of the population has stabilized, the new chromosomes generated are very similar to their parents or have more groups, which mean they have a worse cost.

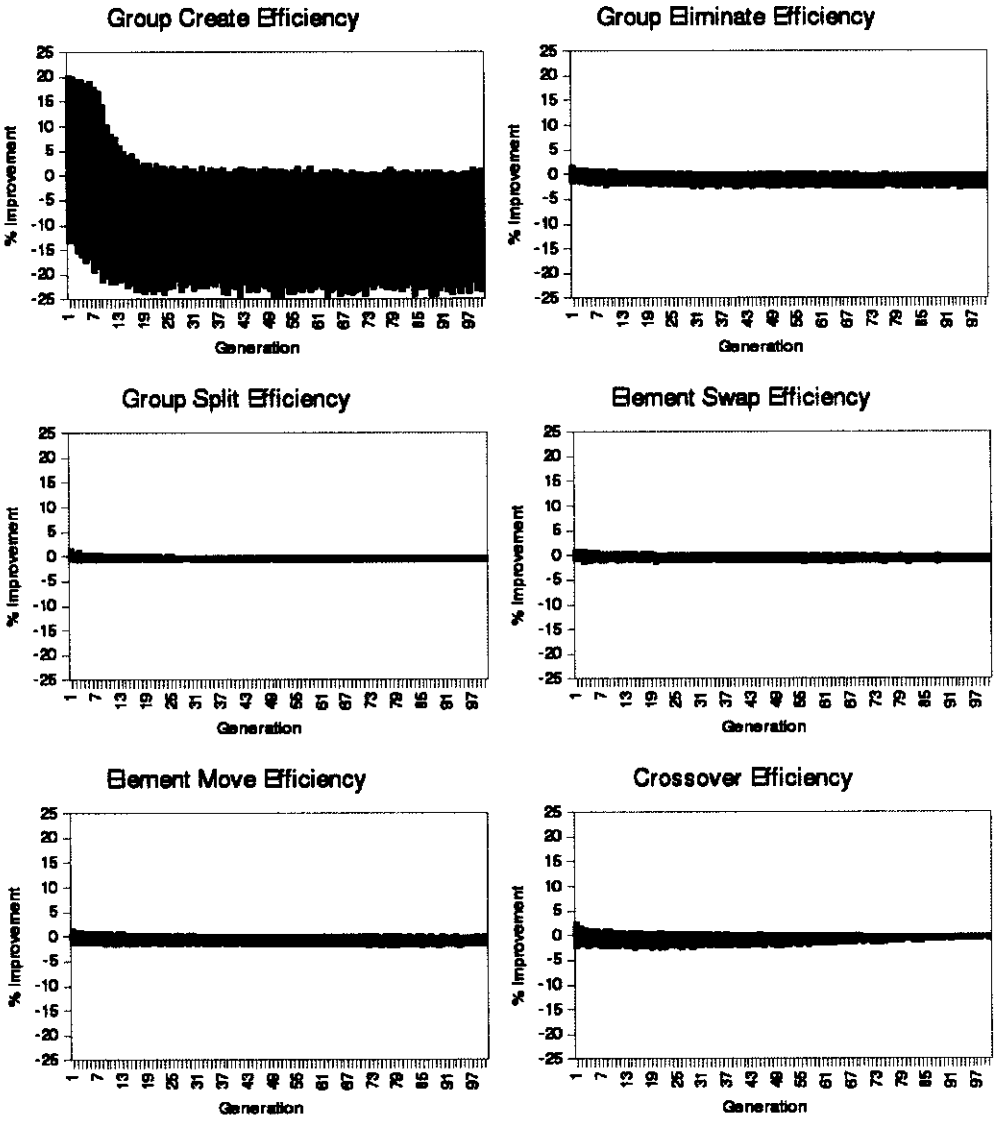

Figure 4.7: Efficacy of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records.

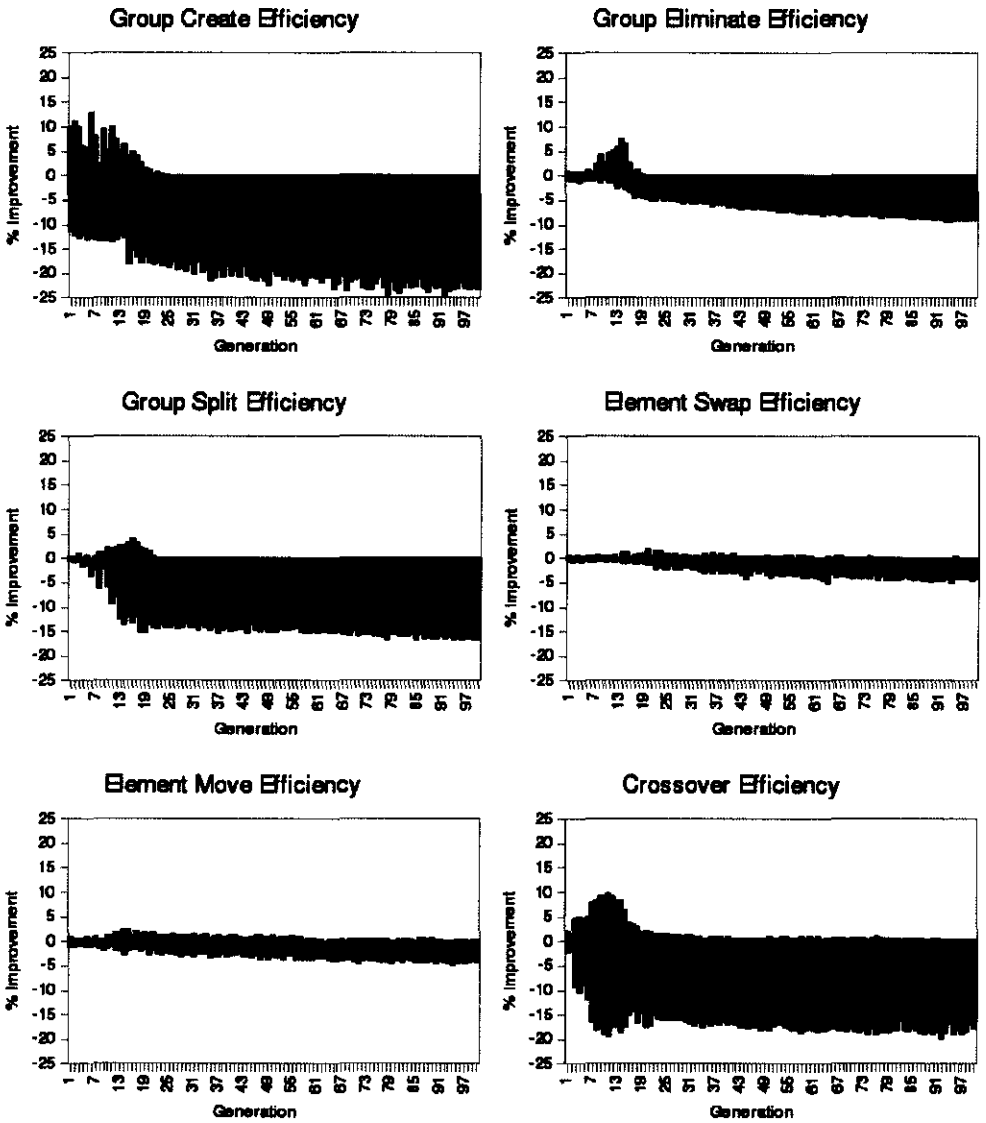

Figure 4.8: Efficacy of crossover and mutation operations for the Water-treatment dataset with clusters of 25 records when used combined.

Census dataset with clusters of 25 records

The efficiency charts for the Census dataset can be found in Section B.1. Again, they are very similar to the plots for the Water-treatment dataset. However, they differ in the scale, that is, the percentage of improvement for the Census dataset is bigger than the same percentage when using the Water-treatment dataset. This is caused by the difference in the number of records between datasets. With the same value of the k parameter, the information loss for Water-treatment is larger than the same measure when using Census since the number of clusters of the former is lower.

Water-treatment dataset with clusters of 100 records

We present the efficiency charts of the Water-treatment dataset with $k = 100$ in Section B.2. The group-oriented mutations introduce improvements in a lower scale than the case where $k = 25$. The information loss of all the possible solutions has increased and the range of scores is now lower, hence, solutions with a similar number of groups, have a similar fitness as well.

4.6 Best Cost Evolution

After understanding the behaviour of every genetic operation, this last analysis aims at studying the contribution of every operator in the final solution. In other words, the objective of this analysis is the demonstrate that the combination of all genetic operations gives better solutions than the executions where the operators are used alone.

We monitored the average best cost evolution obtained by every genetic operation when used isolated in the whole optimization process. Thus, dividing this results by the average best cost evolution of the combined execution, we obtained a reliable comparison.

Figure 4.9 presents the best cost evolution analysis obtained when executing GOMM with the Water-treatment dataset. The leftmost plot shows the results using a k parameter of 25, while the rightmost chart uses a k value of 100. Depending on the value of this parameter, the solution proposed by GOMM has a different number of groups and, therefore, the genetic operations have behave differently. The reason to this phenomenon is explained in Section 5.1.

With clusters of size $k = 25$, we notice that the only operator capable of achieving a cost close to the solution proposed by the combined execution is φ_{GC}. This genetic operation is able to quickly reduce the number of groups of a chromosome, but afterwards, it cannot explore the promising zones it has reached because the transformations it performes are too disruptive. This behaviour is the reason that leads φ_{GC} to perform even better than

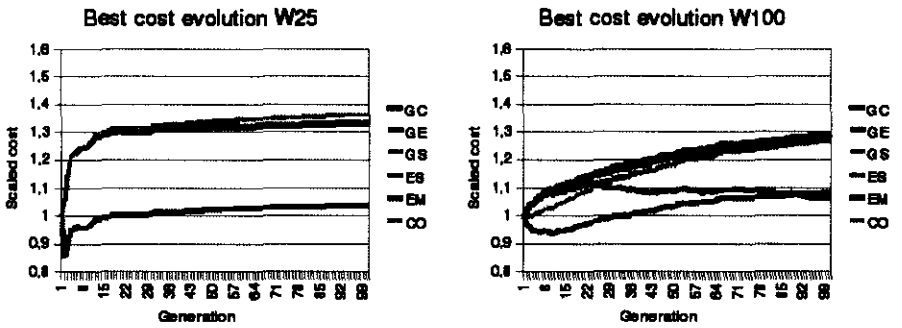

Figure 4.9: Scaled best cost evolution compared to the combination of all the genetic operations for the Water-treatment dataset.

the combined execution in the first generations, but then it is not able to improve.

The operations which increase or mantain the number of groups of the solution cannot make the population evolve towards the optimal solution. These include φ_{GS}, φ_{ES} and φ_{EM} along with the crossover operation. Regarding φ_{GE} , the high number of attributes of the Water-treatment dataset causes it to end in local maxima solutions, because the changes introduced allow it to only explore solutions with a very similar fitness.

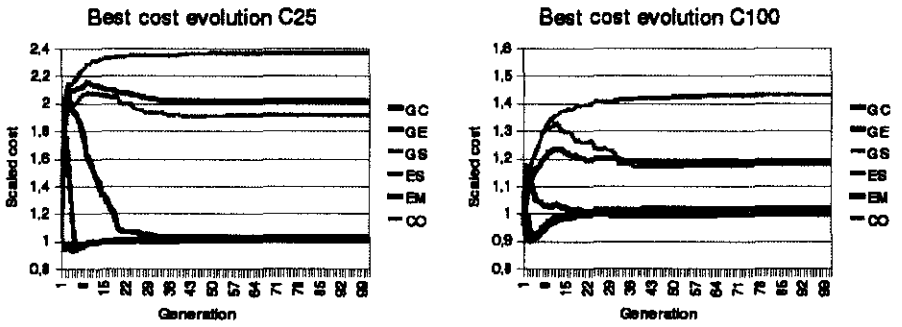

Figure 4.10: Scaled best cost evolution compared to the combination of all the genetic operations for the Census dataset.

This situation changes completely when GOMM is executed with $k = 100$. In this case, the quasi-optimal solution has more groups and the operations which reduce the number of groups are able to get close to the optimal solution. Therefore, φ_{GC} rapidly reaches again promising zones in the so-

lution space, but it cannot explore them. On the other hand, φ_{GE} slowly reduces the number of groups and step by step achieves interesting solutions. In the last generations, φ_{GE} improves the quality reached by φ_{GC}, which is not improving the fitness of the solution it found many generations ago when used alone.

Regarding the Census dataset, the best cost evolution plots can be found in Figure 4.10. When $k = 25$, we observe that along with φ_{GC}, the operators φ_{GE} and φ_{EM} get solutions very close to the quasi-optimal found by the combined execution. This dataset has a lower number of attributes and this property allows the operators that reduce the number of groups to reach interesting solutions by themselves. With a k parameter equal to 100, this also happens. However, the solution found by the combined executions is always better.

4.7 Conclusions of the Analysis

After analyzing the behaviour of the genetic operators, we present some general conclusions extracted from the results shown in this chapter.

First of all, we realize that when used combined, the genetic operations are able to find better solutions than when used in isolation. However, when the number of attributes of the original dataset and the value of the k parameter are low, the operators that decrease the number of groups of the solution give close solutions to the optimal when used alone. This happens because the search space is not very large and the optimal solution has a low number of groups.

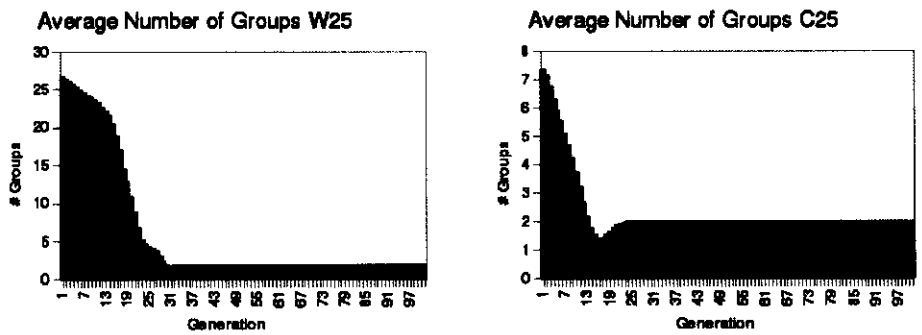

Figure 4.11: Average number of groups for the Water-treatment and the Census dataset with $k = 25$.

Afterwards, we observe that some operations carry out a lot of useless work after a given generation. As the algorithm proceeds we distinguish two separated phases: firstly, the group-oriented mutations (φ_{GC}, φ_{GE} and

φ_{GS}) contribute to evolve the population modifying the number of groups; secondly, these operators only generate useless work and the object-oriented mutations, along with the crossover, introduce improvements in a lower scale.

To illustrate these two phases, we monitored the average number of attribute groups in the population for the two considered datasets. The results are shown in Figure 4.11. Note that for both datasets there is a first phase where the number of groups of the population is adjusted and a second phase where it is constant. For the Water-treatment dataset, the second phase starts near generation 30 while for the Census dataset it starts earlier, in generation 20.

This behaviour shows that GOMM adjustes the number of groups in the first phase, and then, moves elements between groups or recombines them to generate the quasi-optimal solution. If a control mechanism is able to detect the point where the second phase starts, GOMM could stop performing the group-oriented mutations, which produce only useless work in this phase. This is exactly what D-GOMM (see Section 3.5), tries to achieve.

Chapter 5

Experimental Results

This chapter presents the execution results given by GOMM using the same datasets as in the previous chapter. In first place, we check the validity of the solutions given by the algorithm in order to ensure that they are coherent with the multivariate microaggregation context. Afterwards, the solutions are compared with other attribute grouping strategies proposed in the literature. Finally, we compare GOMM with the improved version proposed in Section 3.5, called D-GOMM, in terms of execution time.

5.1 Solution Validation

Before comparing the solutions given by GOMM with other attribute partitioning approaches, we must ensure that these solutions are correct. To do this, we want to demonstrate that when the value of the k parameter increases, the number of attribute groups of the solution given by GOMM also increases. Intuitively, when k increases, the size of the clusters grows producing an increment in the information loss of each cluster. This effect, combined with the fact that information loss also increases when a large number of attributes are grouped together, causes, with large k values, configurations with a large number of groups to be preferred to configurations with only one or two groups, as in the case of low k values.

Tables 5.1 and 5.2 show the components of the cost function and the number of groups of the solution proposed by GOMM with different values of the k parameter. The former presents the results for the Water-treatment dataset while the latter shows the results when using the Census dataset.

Note that in both tables, when k increases, the number of groups of the best solution also increases as we predicted. This means that the solutions given by GOMM are coherent with the multivariate microaggregation scenario.

Water-treatment					
k	IL	ID	DLD	*score*	*groups*
5	35.74	27.93	0.00	24.85	1
10	48.05	25.07	0.00	30.29	1
25	49.52	24.14	6.05	32.31	2
50	52.78	23.85	7.89	34.32	3
100	58.33	22.92	5.79	36.34	5

Table 5.1: Components, score and number of groups of the best configuration found by GOMM with diferent parameterizations using the Water-treatment dataset

Census					
k	IL	ID	DLD	*score*	*groups*
5	8.01	32.01	0.00	12.01	1
10	11.97	25.90	0.00	12.46	1
25	13.67	22.12	2.87	13.08	2
50	15.64	20.31	1.30	13.22	2
100	16.28	18.60	3.80	13.74	3

Table 5.2: Components, score and number of groups of the best configuration found by GOMM with diferent parameterizations using the Census dataset

5.2 Solution Quality

In this section we compare the solutions provided by GOMM with the hand-made solutions used in practice. To do that, we have executed GOMM with different k values (from 5 to 100) to study the impact of such parameter in the groups configuration of the best solution.

In order to compare our results with the different strategies presented in the literature, we manually split both datasets following the recommendations described in [13]. The resulting configurations are depicted in Tables A.1 and A.2.

Configurations vary from few groups of many attributes to many groups of few attributes. In such configurations, we also considered the correlation between attributes. On the one hand, we grouped correlated attributes together (to minimize the information loss), and on the other hand, we grouped correlated attributes in different groups (to minimize the disclosure risk). G_c^n stands for groups of n correlated attributes and G_{nc}^n stands for groups of n non-correlated attributes. Finally, we also consider two special scenarios: in the first one, all attributes are grouped together, ensuring k anonymity. In the second one, we consider the univariate microaggregation scenario where each attribute is microaggregated separately.

The results presented in Tables 5.3 and 5.4 show the fitness values of

Water-treatment					
	k=5	k=10	k=25	k=50	k=100
G^{38}	24.8 (1)	30.2 (1)	36.2 (1)	41.7 (1)	50.4 (1)
G^{1}	49.3 (38)	48.9 (38)	47.2 (38)	45.0 (38)	46.1 (38)
G_c^{10}	43.8 (4)	45.3 (4)	44.0 (4)	40.9 (4)	40.4 (4)
G_{nc}^{10}	43.4 (4)	45.9 (4)	44.0 (4)	41.1 (4)	44.7 (4)
G_c^{5}	42.1 (8)	43.9 (8)	47.0 (8)	49.5 (8)	45.5 (8)
G_{nc}^{5}	42.7 (8)	45.3 (8)	48.7 (8)	49.8 (8)	49.7 (8)
G_c^{3}	42.4 (13)	43.2 (13)	46.6 (13)	50.1 (13)	52.3 (13)
G_{nc}^{3}	42.2 (13)	43.3 (13)	46.1 (13)	49.7 (13)	52.2 (13)
$GOMM$	24.8 (1)	30.2 (1)	32.3 (2)	34.3 (3)	36.3 (5)

Table 5.3: Fitness of some hand-made grouping and the solution given by GOMM using the Water-treatment dataset with different values of the k parameter. Number of groups are depicted in parenthesis.

Census					
	k=5	k=10	k=25	k=50	k=100
G^{12}	12.0 (1)	12.4 (1)	14.2 (1)	15.5 (1)	17.9 (1)
G^{1}	48.9 (12)	48.1 (12)	45.3 (12)	41.8 (12)	37.6 (12)
G_c^{4}	39.7 (3)	37.3 (3)	31.4 (3)	24.4 (3)	16.6 (3)
G_{nc}^{4}	34.5 (3)	27.8 (3)	19.6 (3)	14.9 (3)	15.1 (3)
G_c^{3}	40.4 (4)	38.0 (4)	33.1 (4)	27.0 (4)	20.1 (4)
G_{nc}^{3}	38.0 (4)	34.3 (4)	27.5 (4)	20.9 (4)	16.2 (4)
$GOMM$	12.0 (1)	12.4 (1)	13.0 (2)	13.2 (2)	13.7 (3)

Table 5.4: Fitness of some hand-made grouping and the solution given by GOMM using the Census dataset with different values of the k parameter. Number of groups are depicted in parenthesis.

all the hand-made configurations along with the fitness values obtained by GOMM for different values of k. If we compare the results obtained using the same k value, we observe that GOMM has always the best cost, unless when the quasi-optimal solution given by GOMM solution is one of the hand-made grouping we executed. For instance, if we compare the Water-treatment results (Table 5.3) obtained with $k = 50$, we see that GOMM achieves a fitness equal to 34.3 splitting the dataset in three different groups, whilst the best hand-made configuration only achieves a fitness value equal to 40.9 dividing the dataset into four groups. Similar results are obtained with the remaining k values.

Also, if we compare GOMM results with the Census dataset, we observe very similar results. For example, with $k = 25$ GOMM achieves a fitness value equal to 13.0 dividing the Census dataset into two groups and the best hand-made solution is equal to 14.3 without splitting the dataset.

5.3 Performance Results

The aim of this last section is to compare the two versions of our genetic approach in terms of execution time. In Section 3.5 we introduced D-GOMM, which is a version of the GOMM algorithm that detects when a genetic operation is producing useless work and in this case, it stops executing it. Therefore, the number of new generated chromosomes per generation decreases along with the number of fitness evaluations of this offspring. As 99% of the execution time of GOMM algorithm is spent in cost function evaluations, we expect that this reduction helps to improve the algorithm performance.

It is important to notice that D-GOMM only stops the execution of the genetic operations that perform unnecessary work during several generations. Thus, for all the executions performed the quality of the solution given by D-GOMM is always equal to the one resulting of GOMM's execution.

The execution time comparison between the two genetic optimizers is depicted in Figure 5.1 for different values of the k parameter. The left chart has been obtained by executing the algorithms with the Census dataset and the right plot with the Water-treatment. First of all, if we compare the two datasets we observe that the executions with the Census dataset are slower than the Water-treatment ones. This is because Census has a higher number of records, which causes the MDAV algorithm to take longer execution time.

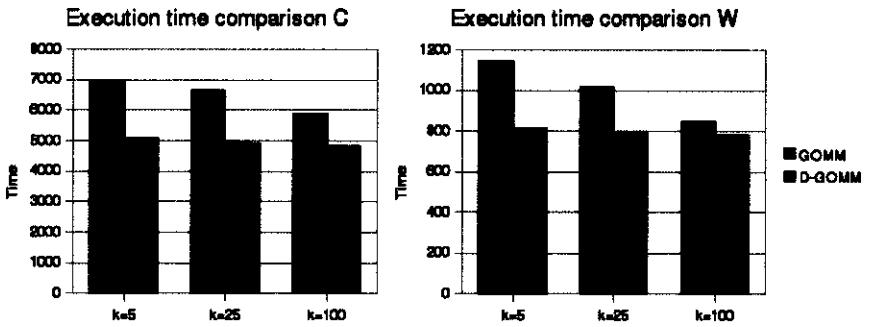

Figure 5.1: Execution time of GOMM and D-GOMM using the Water-treatment dataset and the Census dataset for different values of the k parameter

Regarding the execution time of the genetic optimizers, D-GOMM is faster in all the considered scenarios. However, the speed-up achieved varies depending on the value of the k parameter. We computed the speed-ups in all the scenarios and the results show that as k increases, the speed-up

decreases.

The explanation to this phenomenon is that when k is small, the solution given by the algorithm has less groups of attributes, thus, the optimizer converges faster and the group-oriented mutations stop doing useful work before.

Chapter 6

Economic Analysis and Project Schedule

After presenting the design, the implementation, the analysis and the results given by GOMM, it is time to give a detailed view on the economic issues related to this project. In first place, we describe the project schedule and present it in a Gantt chart. Afterwards, we analyze the specific cost of each one of the tasks defined in the scheduling, along with other costs associated to the project in order to calculate the total project economic budget.

6.1 Project Schedule

Before starting any engineering project, a schedule has to be defined to demonstrate its viability. This schedule shows a prediccion of the begin and the end dates of every task that has to be realized during the project.

When the project advances, some deviations from the original plan may appear. Then, the schedule has to be adapted and corrected in order to deliver the project in the appointed date.

An Informatics Engineering Degree Project has to be completed in one semester (15 weeks) working full time (40 hours per week). However, our project has been realized during a year working half time (20 hours per week) in DAMA-UPC reasearch group. Thus, the workload of this project is a little higher than a typical Informatics Engineering Degree Project.

Figure 6.1 shows the schedule of this project. To simplify, the deviations from this schedule are not presented because they were not so important as the project and the documentation have been delivered in time.

Regarding the tasks, note that the fact that this project has been realized in a research group has marked the way to proceed. The methodology followed includes analysis, design, implementation and testing for every version of the algorithm implemented.

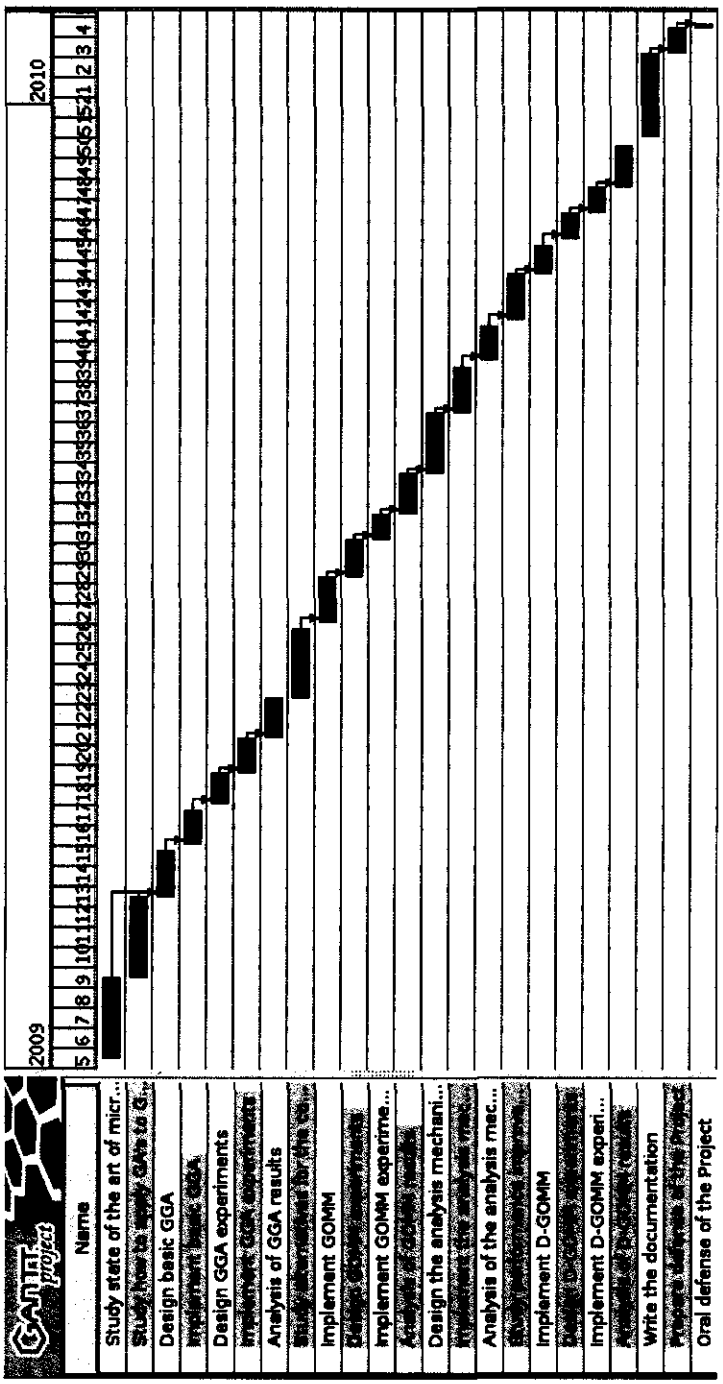

Figure 6.1: Gantt chart showing the final schedule of this project

6.2 Project Costs

To evaluate the total cost of the project we have to study it from two different points of view. Firstly, we calculate the cost of every task defined in the project schedule, which is assigned to a human role that has an associated salary. Afterwards, we take into account the material resources cost, which includes both software and hardware used to develop this project.

Human resources

After presenting the project schedule, we have to assign human resources to every task in order to calculate its cost. As this is an Informatics Engineering Degree Project, it has been realized by a single student under the supervision of a tutor.

Therefore, to obtain a reliable cost estimation we assume that this student plays different roles depending on the task realized. These roles are: analyst, designer and developer. In this economic analysis, we assume that the project manager role corresponds to the analyst.

Table 6.1 displays the typical salaries of the three roles considered to calculate the cost of every task defined in the scheule.

Role	Salary
Analyst	40 €/hour
Designer	30 €/hour
Developer	20 €/hour

Table 6.1: Typical salary of the different roles considered

The assignation of human resources to every task defined in the schedule is presented in Table 6.2, along with the duration (in hours) and the total cost of every task. In the project schedule, the task duration is presented in days. As we assume half-time dedication to this project, every day is counted as four hours.

The last row of the table shows the total amount of hours dedicated to the project and the total cost. This project has been realized in 1.017 hours of work, having a cost of 33.480 €, which means it is a middle-size project.

Material resources

To complete the economic analysis, we have to evaluate the cost of the material resources employed to realize the project. These resources can be classified into two categories:

- *Software resources.* All the software used to complete this project is open source, so its cost is zero. This software includes Linux Ubuntu

Name of the task	Human Resource	Hours	Total cost
Study state of the art of microaggregation problems	Analyst	80	3.200 €
Study how to apply GA's to Grouping Problems	Analyst	80	3.200 €
Design basic GGA	Designer	48	1.440 €
Implement basic GGA	Developer	40	800 €
Design GGA experiments	Designer	36	1.080 €
Implement GGA experiments	Developer	32	640 €
Analysis of GGA results	Analyst	40	1.600 €
Study alternatives for the cost function	Analyst	72	2.880 €
Implement GOMM	Developer	48	960 €
Design GOMM experiments	Designer	36	1.080 €
Implement GOMM experiments	Developer	28	560 €
Analysis of GOMM results	Analyst	40	1.600 €
Design the analysis mechanism	Designer	60	1.800 €
Implement the analysis mechanism	Developer	48	960 €
Analysis of the analysis mechanism	Analyst	40	1.600 €
Study performance improvements	Analyst	48	1.920 €
Implement D-GOMM	Developer	32	640 €
Design D-GOMM experiments	Designer	28	840 €
Implement D-GOMM experiments	Developer	28	560 €
Analysis of D-GOMM results	Analyst	40	1.600 €
Write the documentation	Analyst	84	3.360 €
Prepare defense of the Project	Analyst	28	1.120 €
Oral defense of the Project	Analyst	1	40 €
TOTAL		1.017	33.480 €

Table 6.2: Cost of every task defined in the schedule

operating system, GCC compiler, Vim text editor, Gprof profiler, Latex Kile text editor and Gantt project.

- *Hardware resources.* Regarding hardware, all the executions have been performed in a Acer Veriton 6800 PC with Intel Pentium D processor, which has a cost of 900 €. Moreover, all the code and the documentation have been writen in the same machine.

Thus, the total cost of the material resources is the cost of the hardware resources, which is 900 €.

Total cost

To calculate the total cost of the project we have to add the cost of the human resources and the cost of the material resources:

$$Totalcost = 33.480 + 900 = 34.380 €$$

Chapter 7

Conclusions

This last chapter concludes this project from a twofold perspective. Firstly, we present the work done along with the results achieved. Secondly, we explain the conclusions extracted from this project from the point of view of the student. Finally, we draw some proposals of future work.

7.1 Conclusions

In this project, we have designed and implemented a novel methodology to find the best way to group the attributes when anonymizing a dataset with multivariate microaggregation, in order to obtain a protected dataset with both low disclosure risk and information loss.

Our approach is based on the use of genetic algorithms, which have proven to be a good alternative when dealing with grouping problems. The cost function to evaluate the quality of the different solutions has the same principles as the score measure [4], although some simplifications have been implemented in order to make the algorithm feasible. The result is GOMM, the Genetic Optimizer for Multivariate Microaggregation.

Along with the algorithm, we have designed and implemented mechanisms to analyze the behaviour of every genetic operation in order to understand the evolution process performed by the optimizer. This analysis shows the amount of work each operator is producing and its contribution to the final solution. This allows the study of new techniques to improve the optimizer's performance.

We have proposed two possible improvements, one of which has been implemented in a version called D-GOMM. This new approach is able to detect when a genetic operation is producing useless work and then it stops its execution. The performance results have shown that this improvement reduces the execution time of the optimizer while it gives solutions of the same quality.

Regarding the solutions given by our algorithm, we have compared them

with other grouping strategies proposed in the literature, using our approximate cost function. The results shown that the solutions given by GOMM outperform these grouping approaches for different parametrizations of the multivariate microaggregation method.

7.2 Personal conclusions

As a student that is finishing his Informatics Engineering Degree, this project has been the first important challenge I have ever faced during my years as a student.

It has allowed me to apply some of the knowledge adquired during my studies such as software engineering, system performance evaluation or artificial intelligence. Moreover, it has opened my eyes to topics I was not aware of, like data anonymization.

The fact that this project has been realized in DAMA-UPC research group has introduced me to the laboral world. A year ago I had no idea of the way I would follow when I finish my studies; now I feel I would like to continue in the research world and to contribute to advance the science.

Finally, the accomplishment of all the objectives proposed in the beginning of the project has been a moral injection. I have enjoyed every step from the initial idea to the final product and this is very important to feel truly satisfied.

7.3 Future work

The main objective at this moment is to publish a paper to present the contributions proposed in this project.

Later on, the monitoring mechanisms implemented in GOMM allow to easily explore new approaches to evaluate the quality of the solutions. Other approximate cost functions may be tested in order to improve the final result.

Moreover, the implementation of the performance improvements proposed would allow the algorithm to deal with larger datasets, which would increase its usefulness in the real world.

Appendix A

Hand-made Attribute Groupings

A.1 Census dataset

	Census
G_c^4	$(a_2, a_4, a_6, a_7)(a_5, a_{10}, a_{11}, a_{12})$ (a_1, a_3, a_8, a_9)
G_{nc}^4	$(a_1, a_2, a_4, a_5)(a_3, a_6, a_{10}, a_{11})$ (a_7, a_8, a_9, a_{12})
G_c^3	$(a_2, a_4, a_7)(a_3, a_{11}, a_{12})$ $(a_5, a_6, a_{10})(a_1, a_8, a_9)$
G_{nc}^3	$(a_2, a_3, a_5)(a_8, a_{10}, a_{11})$ $(a_7, a_9, a_{12})(a_1, a_4, a_6)$

Table A.1: Hand-made group configuration for the Census dataset

A.2 Water-treatment dataset

	Water-treatment
G_c^{10}	$(a_1, a_4, a_5, a_6, a_7, a_{11}, a_{13}, a_{17}, a_{20}, a_{36})$ $(a_3, a_9, a_{10}, a_{15}, a_{16}, a_{18}, a_{21}, a_{22}, a_{23}, a_{29})$ $(a_2, a_8, a_{12}, a_{14}, a_{19}, a_{26}, a_{28}, a_{31}, a_{33}, a_{37})$ $(a_{24}, a_{25}, a_{27}, a_{30}, a_{32}, a_{34}, a_{35}, a_{38})$
G_{nc}^{10}	$(a_1, a_2, a_3, a_{24}, a_4, a_9, a_8, a_{25}, a_5, a_{10})$ $(a_{12}, a_{27}, a_6, a_{15}, a_{14}, a_{30}, a_7, a_{16}, a_{19}, a_{32})$ $(a_{11}, a_{18}, a_{26}, a_{34}, a_{13}, a_{21}, a_{28}, a_{35}, a_{17}, a_{22})$ $(a_{20}, a_{36}, a_{29}, a_{23}, a_{31}, a_{33}, a_{37}, a_{38})$
G_c^5	$(a_1, a_4, a_5, a_6, a_7)(a_{11}, a_{13}, a_{17}, a_{20}, a_{36})(a_3, a_9, a_{10}, a_{15}, a_{16})$ $(a_{18}, a_{21}, a_{22}, a_{23}, a_{29})(a_2, a_8, a_{12}, a_{14}, a_{19})$ $(a_{26}, a_{28}, a_{31}, a_{33}, a_{37})(a_{24}, a_{25}, a_{27}, a_{30})(a_{32}, a_{34}, a_{35}, a_{38})$
G_{nc}^5	$(a_1, a_2, a_3, a_{24}, a_4)(a_9, a_8, a_{25}, a_5, a_{10})(a_{12}, a_{27}, a_6, a_{15}, a_{14}, a_{30})$ $(a_7, a_{16}, a_{19}, a_{32})(a_{11}, a_{18}, a_{26}, a_{34}, a_{13}, a_{21})$ $(a_{28}, a_{35}, a_{17}, a_{22})(a_{20}, a_{36}, a_{29}, a_{23})(a_{31}, a_{33}, a_{37}, a_{38})$
G_c^3	$(a_1, a_4, a_5)(a_6, a_7, a_{11})(a_{13}, a_{17}, a_{20})(a_3, a_9, a_{10})$ $(a_{15}, a_{16}, a_{18})(a_{21}, a_{22}, a_{23})(a_2, a_8, a_{12})(a_{14}, a_{19}, a_{26})(a_{28}, a_{31}, a_{33})$ $(a_{29}, a_{36}, a_{37})(a_{24}, a_{25}, a_{27})(a_{30}, a_{32}, a_{34})(a_{35}, a_{38})$
G_{nc}^3	$(a_1, a_2, a_3)(a_{24}, a_4, a_9)(a_8, a_{25}, a_5)(a_{12}, a_{27}, a_6)$ $(a_{15}, a_{14}, a_{30})(a_7, a_{16}, a_{19})(a_{11}, a_{18}, a_{26})(a_{34}, a_{13}, a_{21})(a_{28}, a_{35}, a_{17})$ $(a_{10}, a_{32}, a_{22})(a_{20}, a_{36}, a_{29})(a_{23}, a_{31}, a_{33})(a_{37}, a_{38})$

Table A.2: Hand-made group configurations for the Water-treatment dataset

Appendix B

Extended Analysis

Due to the extension of the analysis performed on GOMM, in Chapter 4 we decided to present only the results for the Water-treatment dataset with $k = 25$. In this appendix we present the rest of the plots corresponding to the Census dataset with $k = 25$; and the Water-treatment dataset with $k = 100$ which have already been analyzed and commented.

B.1 Census Dataset with $k = 25$

Utilization analysis

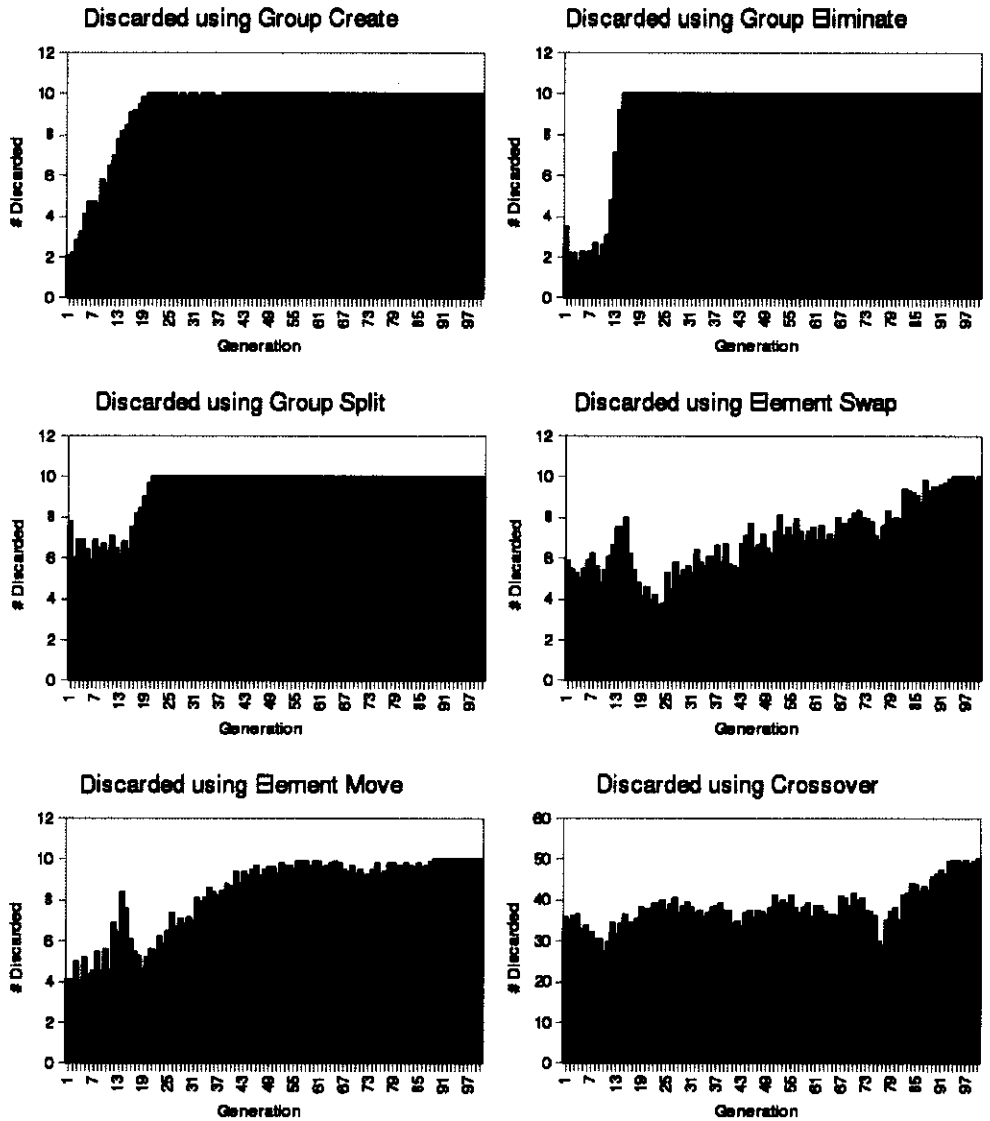

Figure B.1: Utilization of crossover and mutation operations for the Census dataset with clusters of 25 records when used combined.

Averafe Lifetime analysis

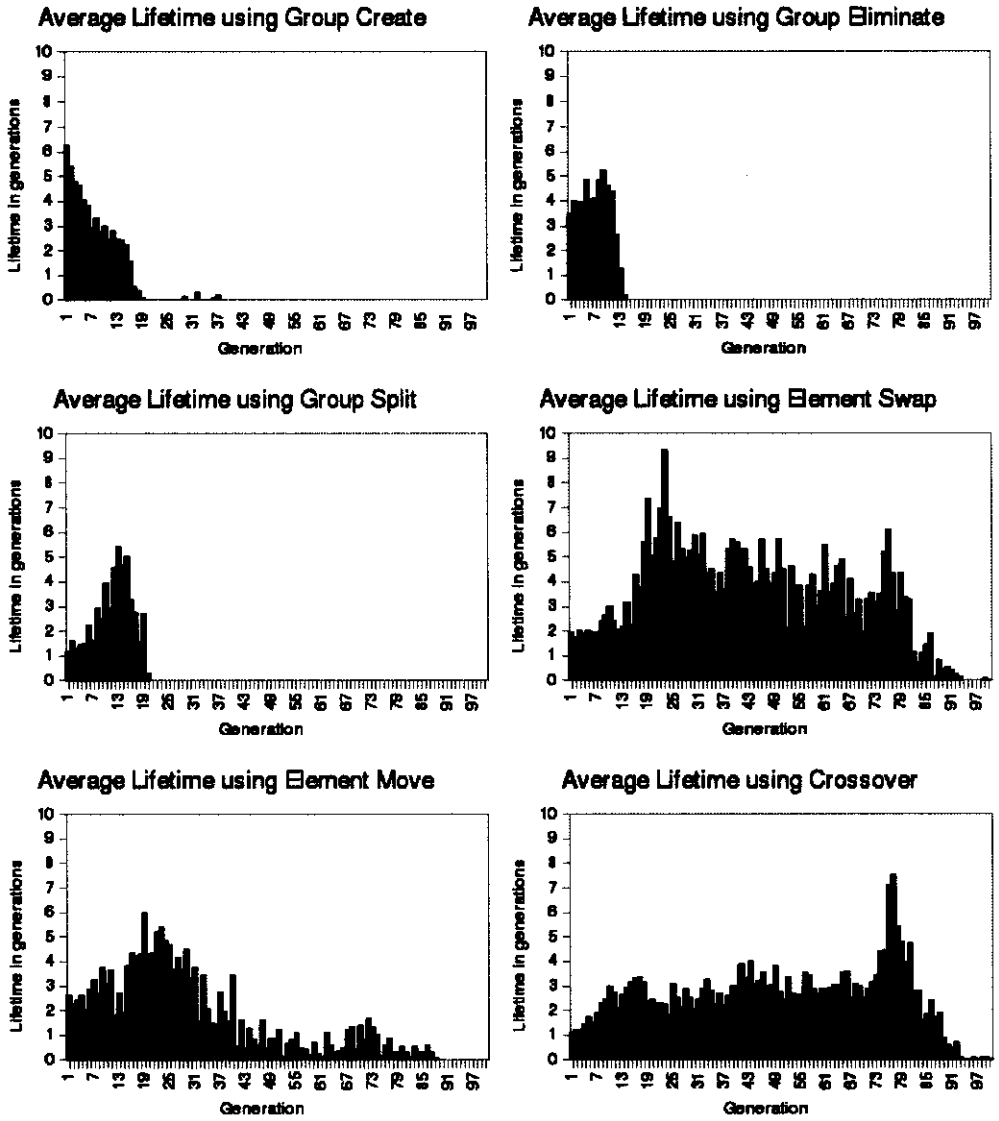

Figure B.2: Average Lifetime of crossover and mutation operations for the Census dataset with clusters of 25 records when used combined.

Efficacy analysis

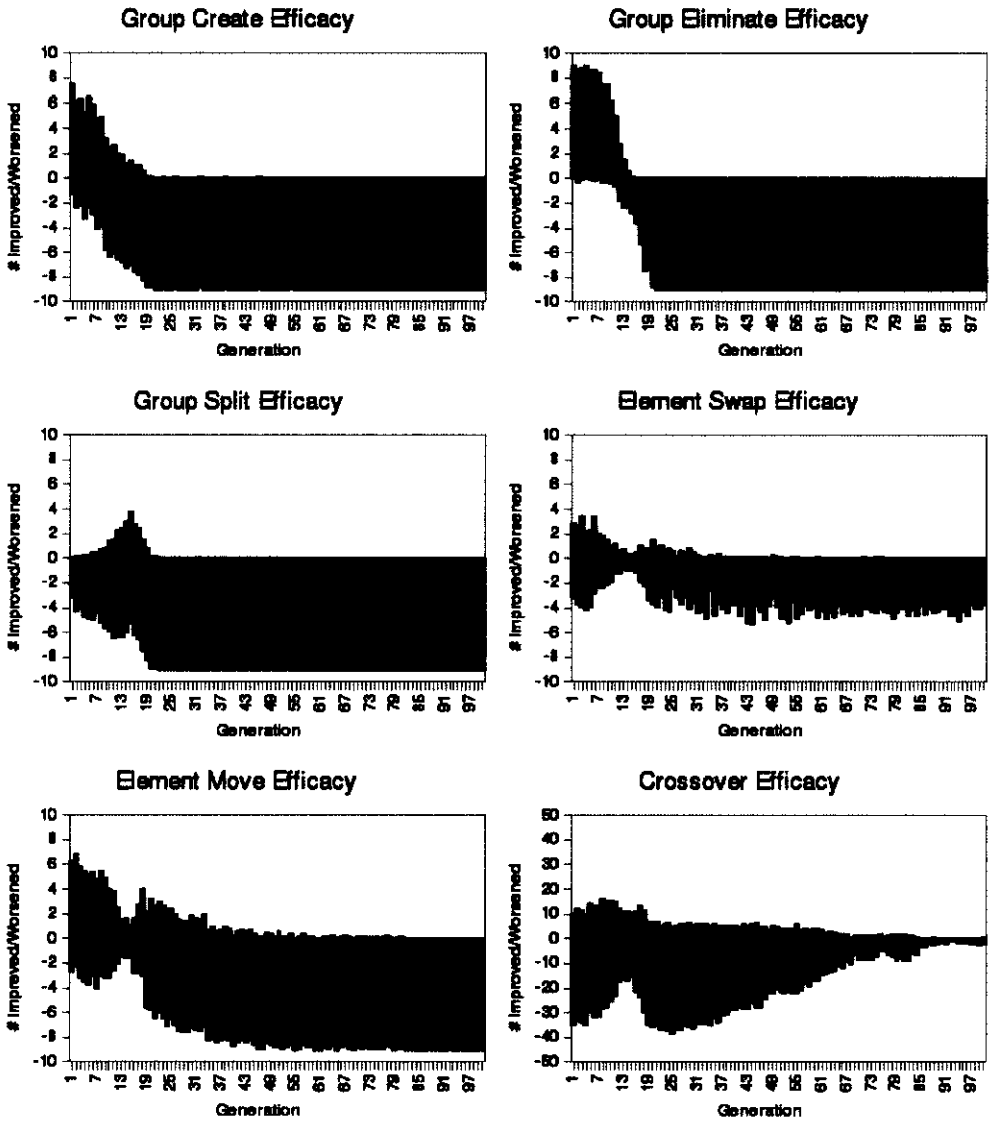

Figure B.3: Efficacy of crossover and mutation operations for the Census dataset with clusters of 25 records when used combined.

Efficiency analysis

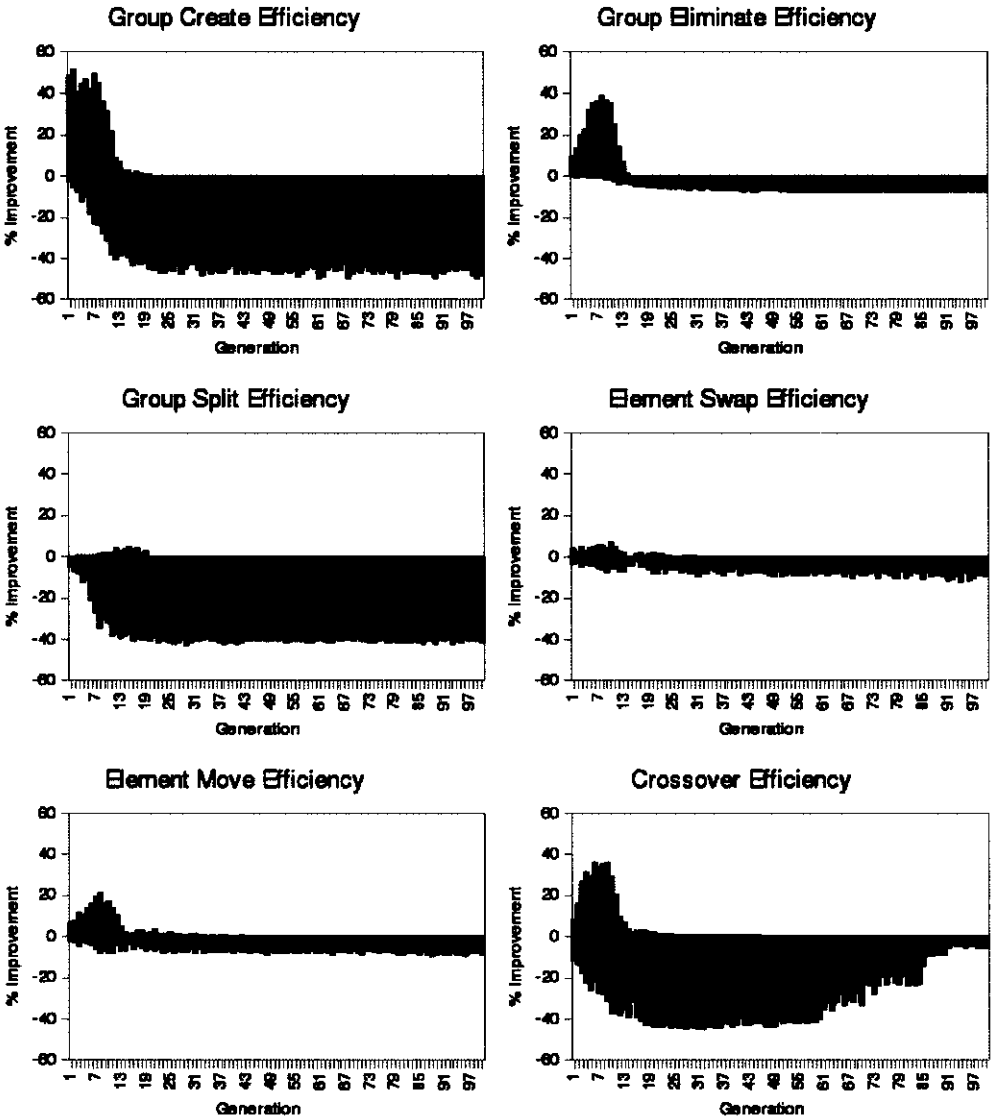

Figure B.4: Efficacy of crossover and mutation operations for the Census dataset with clusters of 25 records when used combined.

B.2 Water-treatment Dataset with $k = 100$

Utilization analysis

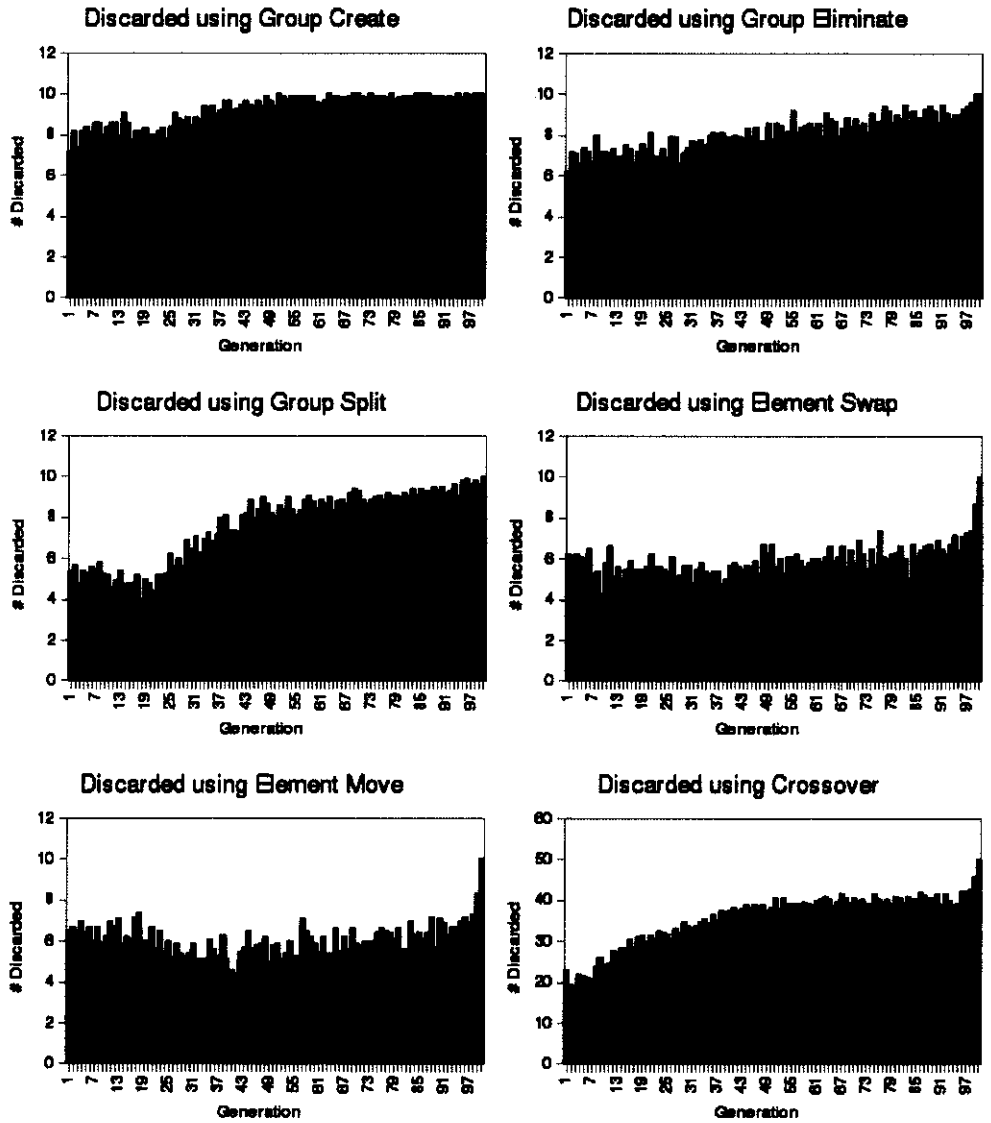

Figure B.5: Utilization of crossover and mutation operations for the Water-treatment dataset with clusters of 100 records when used combined.

Averafe Lifetime analysis

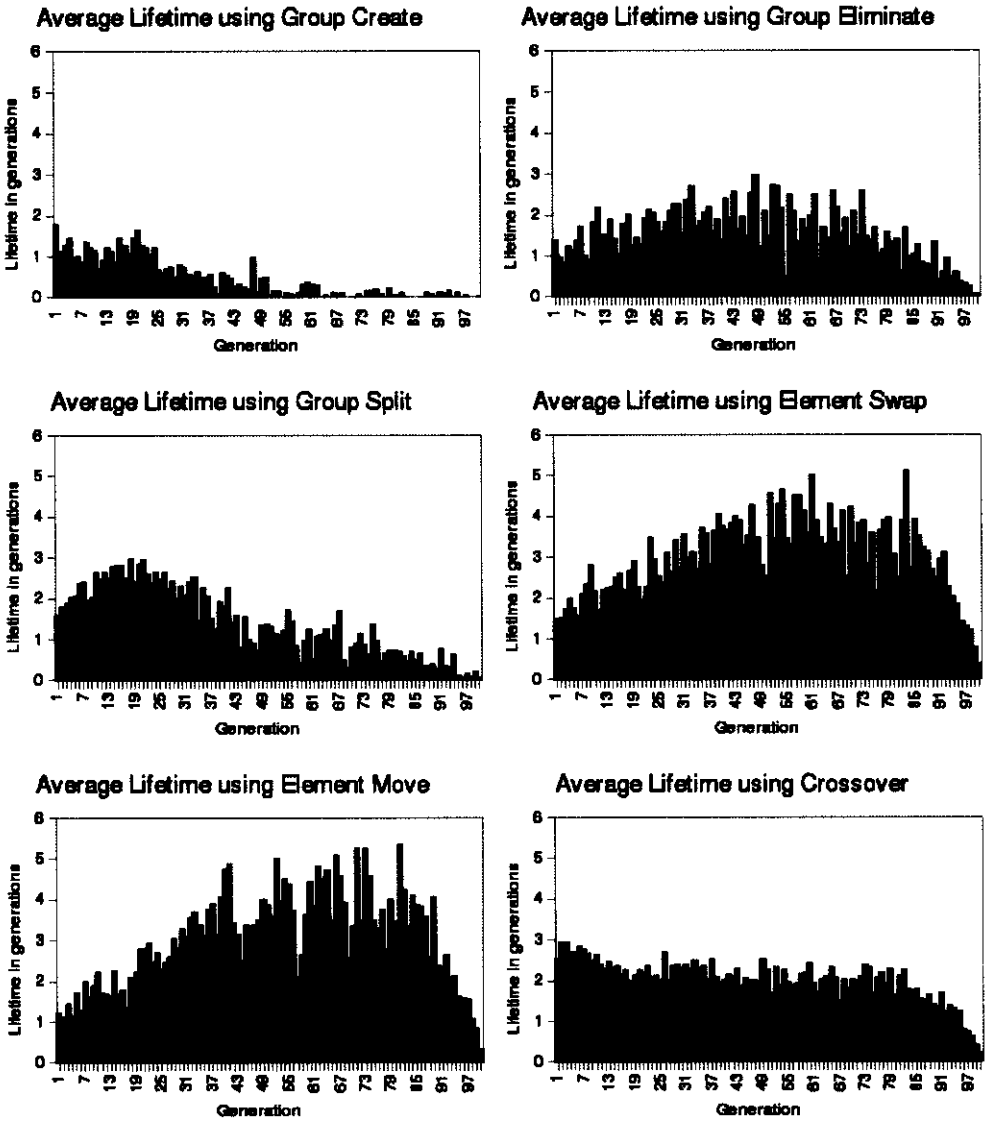

Figure B.6: Average Lifetime of crossover and mutation operations for the Water-treatment dataset with clusters of 100 records when used combined.

Efficacy analysis

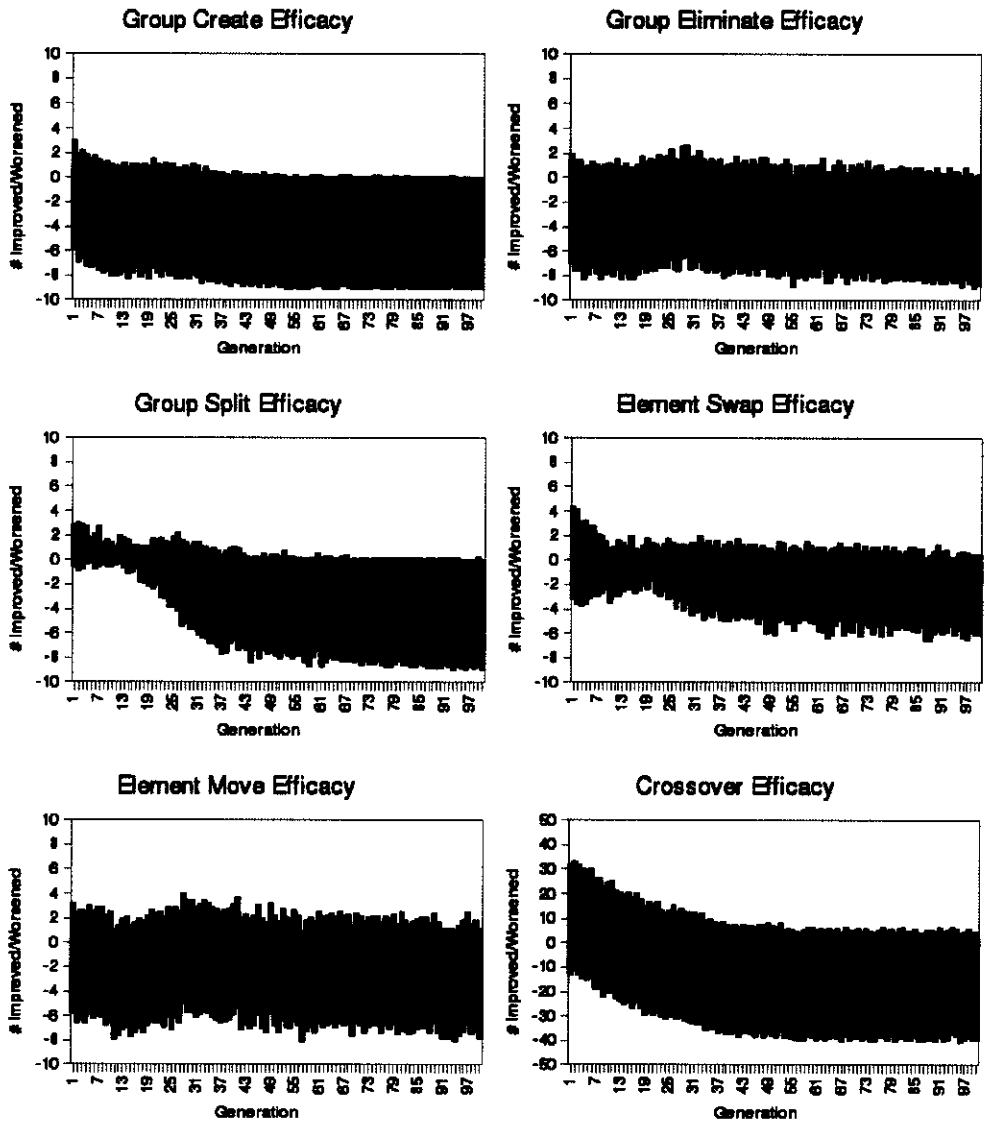

Figure B.7: Efficacy of crossover and mutation operations for the Water-treatment dataset with clusters of 100 records when used combined.

Efficiency analysis

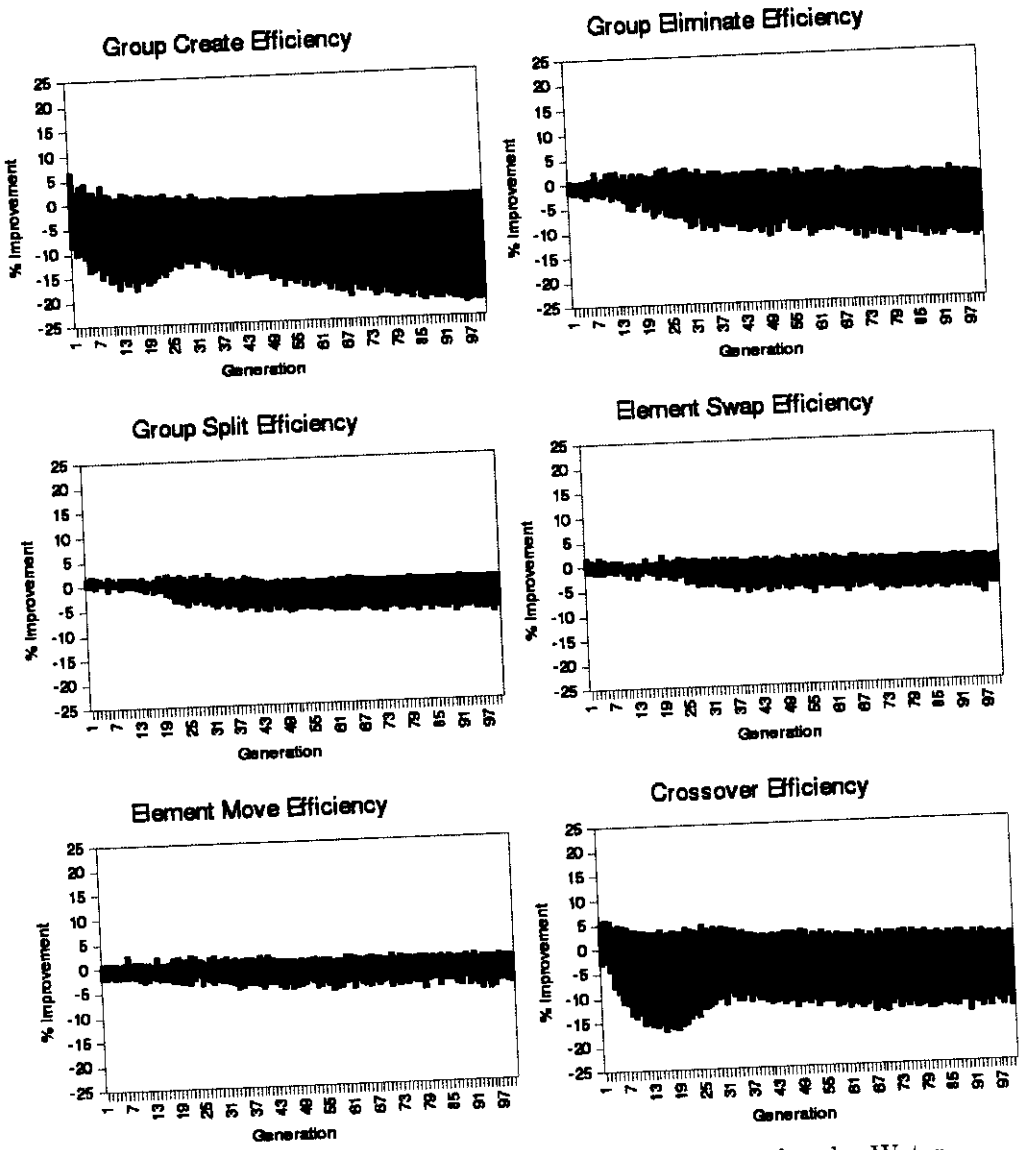

Figure B.8: Efficacy of crossover and mutation operations for the Water-treatment dataset with clusters of 100 records when used combined.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, 2000.

[2] T. Dalenius and S.P. Reiss. Data-swapping: a technique for disclosure control. *Journal of Statistical Planning and Inference*, 6:73–85, 1982.

[3] J. Domingo-Ferrer and J. M. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Trans. on Kwnoledge and Data Engineering*, 14(1):189–201, 2002.

[4] J. Domingo-Ferrer and V. Torra. Disclosure control methods and information loss for microdata. In *Confidentiality, disclosure, and data access : Theory and practical applications for statistical agencies*, pages 91–110. Elsevier, 2001.

[5] Josep Domingo-Ferrer and Vicenç Torra. Validating distance-based record linkage with probabilistic record linkage. In *Proc. of the Catalan Conference on Artificial Intelligence*, pages 207–215, 2002.

[6] Josep Domingo-Ferrer and Vicenç Torra. Ordinal, continuous and heterogeneous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery*, 11(195–212), 2005.

[7] Josep Domingo-Ferrer, Vicenç Torra, Josep M. Mateo-Sanz, and Francesc Sebé. Systematic measures of re-identification risk based on the probabilistic links of the partially synthetic data back to the original microdata. Technical report, 2005.

[8] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[9] S. Hansen and S. Mukherjee. A polynomial algorithm for optimal univariate microaggregation. *IEEE Trans. on Kwnoledge and Data Engineering*, 15(4):1043–1044, 2003.

[10] Victor Muntés-Mulero. *Genetic Optimization for large join queries*. PhD thesis, Universitat Politècnica de Catalunya, 2007.

[11] P. Murphy and D. Aha. UCI Repository machine learning databases. *Irvine, CA: University of California, Department of Information and Computer Science*, 1994.

[12] J. Nin, J. Herranz, and V. Torra. Attribute selection in multivariate microaggregation. In *Post-Proc. of 11th ACM International Conference on Extending Database Technology (EDBT)*, pages 51–60, 2008.

[13] J. Nin, J. Herranz, and V. Torra. How to group attributes in multivariate microaggregation. *Int. J. of Unc., Fuzz. and Knowledge Based Systems*, 16(1):121–138, 2008.

[14] A. Oganian and J. Domingo-Ferer. On the complexity of optimal microaggregation for statistical disclosure control. *Statistical Journal United Nations Economic Commission for Europe*, 18(4):345–354, 2000.

[15] D. Pagliuca and G. Seri. Some results of individual ranking method on the system of enterprise accounts annual survey. Technical report, Esprit SDC Project, Deliverable MI-3/D2, 1999.

[16] F. Sebé, J. Domingo-Ferrer, J. M. Mateo-Sanz, and V. Torra. Post-masking optimization of the tradeoff between information loss and disclosure risk in masked microdata sets. In *Inference Control in Statistical Databases, Lecture Notes in Computer Science 2316*, pages 187–196. J. Domingo-Ferrer (Ed.), 2002.

[17] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. of Unc., Fuzz. and Knowledge Based Systems*, 10(5):571–588, 2002.

[18] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. of Unc., Fuzz. and Knowledge Based Systems*, 10(5):557–570, 2002.

[19] Vicenç Torra and Josep Domingo-Ferrer. Record linkage methods for multidatabase data mining. In *Information Fusion in Data Mining*, pages 101–132. Springer, 2003.

[20] U.S. Census Bureau, Data Extraction System, http://www.census.gov/.