A 2^d-Tree-Based Blocking Method for Microaggregating Very Large Data Sets*

Agusti Solanas, Antoni Martínez-Ballesté, Josep Domingo-Ferrer and Josep M. Mateo-Sanz
Universitat Rovira i Virgili,
Dept. of Computer Engineering and Maths,
Av. Països Catalans 26,
E-43007 Tarragona, Catalonia, Spain
e-mail {agusti.solanas,antoni.martinez}@urv.net
{josep.domingo,josepmaria.mateo}@urv.net

Abstract

Blocking is a well-known technique used to partition a set of records into several subsets of manageable size. The standard approach to blocking is to split the records according to the values of one or several attributes (called blocking attributes). This paper presents a new blocking method based on 2^d-trees for intelligently partitioning very large data sets for microaggregation. A number of experiments has been carried out in order to compare our method with the most typical univariate one.

Keywords: *Microaggregation, Blocking, Statistical Disclosure Control, Privacy.*

1 Introduction

Due to the new information technologies and the massive use of computers for organizing and distributing data, the amount of stored data has had a spectacular increase in a very diverse set of fields, namely finance, health care and engineering. This new impressive amount of data has to be managed in order to perform a wide variety of tasks from knowledge discovery to statistical disclosure control.

It is not straightforward to manage this big amount of data and generally the data sets must be divided into smaller ones in order to apply different techniques over each generated data subset. To this end, a well known statistical technique called "blocking" is used to divide a huge data set into a number of smaller ones. This technique is based on selecting a number of attributes of the data set, called blocking attributes (*i.e.,* some columns or variables), sort them and divide them as many times as needed to obtain manageable subsets.

Some drawbacks of blocking using blocking attributes are the following:

- The choice of blocking attributes may not be obvious;
- Blocking may fail to adapt to the distribution of data (very heterogeneous blocks);
- The size of some blocks may be too small or too big for some purposes (*e.g.,* privacy preservation).

In this paper, we propose a method based on a 2^d-tree for dividing very large data sets into a number of smaller ones and we concentrate on using this method to adapt the classical microaggregation to manage these huge data sets. The proposed method considers a user-definable number of dimensions d in order to divide the original data set into a number of smaller ones.

1.1 The microaggregation problem

Microaggregation is a statistical disclosure control (SDC) technique and its purpose is to cluster a set of elements in groups of at least k elements, being k a user-definable parameter. As a result, a set of generated groups called k-partition is obtained. Once all the elements in a data set are clustered, a microaggregated data set is built by replacing each original element by the centroid of the group which it belongs to. After this process, the microaggregated data set can be released and the privacy of the individuals which form the original data set can be guaranteed because the aggregated data set will contain at least k identical and indistinguishable elements.

*This work was partially supported by the Government of Catalonia under grant 2005 SGR 00446 and by the Spanish Ministry of Science and Education under project SEG2004-04352-C04-1 "PROPRIETAS".

1.2 Related work

Microaggregation has been used for several years in different countries: it started at Eurostat [3] in the early nineties, and has since then been used in Germany [17] and several other countries [9]. These years of experience and practice devoted to microaggregation have bequeathed us a variety of approaches, which we next briefly summarize.

- Optimal methods:
  - Univariate case: In [12] a polynomial algorithm for optimal univariate microaggregation was presented.
  - Multivariate case: Optimal multivariate microaggregation was shown to be NP-hard in [16]. So the only practical multivariate microaggregation methods are heuristic.
- Heuristic methods:
  - Fixed-size heuristics: The best-known example of this class is Maximum Distance to Average Vector (MDAV) [5, 7, 13]. MDAV produces groups of fixed cardinality k and, when the number of records is not divisible by k, one group with a cardinality between k and $2k-1$. MDAV has proven to be the best performer in terms of time and one of the best regarding the homogeneity of the resulting groups.
  - Variable-size heuristics: These yield k-partitions with group sizes varying between k and $2k-1$. Such a flexibility can be exploited to achieve higher within-group homogeneity. Variable-size methods include the genetic-inspired approach for small data sets in [18] and also [4, 14, 6].

Typically, the clusters that a microaggregation technique generates are evaluated with a compactness or homogeneity measure. The homogeneity of a group can be measured as the inverse of its sum of square errors (SSE) which is computed as

$$SSE = \sum_{i=1}^{s} \sum_{j=1}^{n_i} (\mathbf{x}_{ij} - \bar{\mathbf{x}}_i)'(\mathbf{x}_{ij} - \bar{\mathbf{x}}_i) \qquad (1)$$

where s is the number of generated sets, n_i is the number of elements in the i-th set, $\mathbf{x}_{ij}$ is the j-th element in the i-th set and $\bar{\mathbf{x}}_i$ is the centroid of the i-th set.

SSE is the most utilized measure in clustering and related areas [8, 10, 11, 15, 19]. The main objective of microaggregation is to reduce the SSE and, thus, maximize the within-group homogeneity.

Symbol	Meaning
d	Dimensions
D	A data set
$D_1, D_2, \dots D_G$	The Generated subsets
E_g	Number of elements in group g
G	Number of generated subsets.
k	Min. Elements per group
L	Max. Elements per leaf

Table 1. Table of main symbols

1.3 Contribution and plan of the paper

In this paper a new method for microaggregating very large data sets is presented and compared with the previously utilized univariate blocking technique. The most relevant aspects of our method regarding group size constraints and SSE have been studied in detail.

The rest of the paper is organized as follows. In Section 2 our method is detailed. Next, Section 3 shows the experimental results. Finally in Section 4 the article concludes.

2 The Method

2.1 Problem definition

We are addressing a problem that consists in dividing a large data set D into a number of smaller ones $D_1, D_2, \dots D_G$. We claim that in order to maintain the spatial relations that may exist between the elements in the original data set, it is necessary to consider several dimensions for partitioning the data set into smaller ones. Thus, we propose to use a very well-known data structure such as a 2^d-tree for dividing the data taking into account a user-definable number d of dimensions.

We do not only want to divide the original data, but we want to finally obtain a microaggregated set in order to allow the data to be safely released by minimizing the SSE. To that end, the leaves of the generated 2^d-tree must be modified for properly applying a microaggregation technique over each one. Once all leaves are prepared, a microaggregation method is applied.

Specifically, the MDAV heuristic [5, 7, 13] has been used to obtain the empirical results presented in Section 3.

A complete explanation about how to design and implement a 2^d-tree can be found in [2], thus, we will only elaborate on the most relevant aspects of the 2^d-tree generation, and the other main steps of the method. Table 1 shows the notation used in the paper.

2.2 2^d-tree generation

The first step of our method consists of building a 2^d-tree. This kind of structure is commonly found in image analysis with d equaling two (*i.e.,* Quadtree) and virtual reality or satellite image analysis with d equaling three (*i.e.,* Octree).

The number of dimensions d is a parameter to our method and must be defined by the user. Although any value of d is possible, it is not a good idea to use high values of d because the children of each node in the tree grows exponentially with d (*i.e.,* each node has 2^d children). In this paper we have used data sets with a dimensionality between 2 and 10. Thus, all the dimensions were used to build the 2^d-tree. When the number of dimensions is bigger, the user must specify which dimensions have to be used to generate the 2^d-tree. Figure 1 shows a typical partition obtained by applying a 2^d-tree with d equaling 2.

In Table 2 our structure of a node is shown in C style. All the elements in the structure are common to any tree structure in which there must be relations between the father and the children, a node identifier, etc. However, the most relevant element in the node structure of a 2^d-tree is the ***dimension limits vector***. The dimension limits vector contains an upper-bound and a lower-bound for each dimension. Thus, all the records contained in a node must fall inside these limits.

Initially, the 2^d-tree has a single leaf and it is divided into 2^d leaves if the number of records inside its bounds is bigger than L. This division of the leaves into 2^d children leaves is recursively applied until all the leaves have a maximum number of records, within its dimension limits, lower than L.

The dimension limits vectors of child nodes are computed as follows. The range of each of the d dimensions in the parent node is split into two subranges: one from the parent's lower bound for that dimension to the midpoint of the parent's range and another from the midpoint of the parent's range to the parent's upper bound. This is done for all d dimensions, so that eventually 2^d d-dimensional subranges are obtained, each of which is assigned to a child node.

Moreover, there are some computational issues which may be emphasized:

- Each node in the 2^d-tree must know which records fall inside the space region that it represents. However, each record is not really stored in terms of the value of all its dimensions; instead, a reference to its position (*i.e.,* an ordinal) in the original data set is stored. This results in a better use of the physical memory.

- During this step no distance computations are needed and all the process can be completed with simple and computationally cheap comparisons.

Table 2. A node structure

```
struct node
{
   int nodeID;            //Node identifier
   int level;             //Depth of the node
   int Nrecords;          //Number of records
   int *records;          //Reference to the records
   int Nchildren;         //Number of children
   node *children;        //A reference to the children
   node *parent;          //The parent node
   float **dimLimits;     //The dimensional limits
};
```

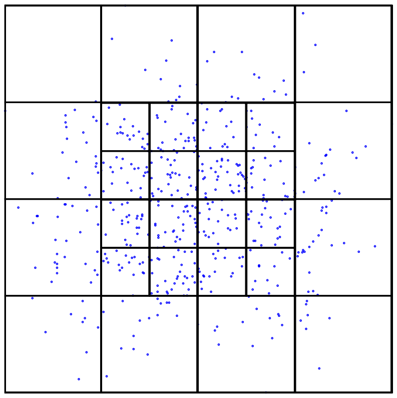

Figure 1. Example of a quad-tree partition of a data set of 500 elements with $L = 50$.

2.3 Fusing low-cardinality leaves

Once the 2^d-tree is generated, we must pay attention to its leaves. By construction, the number of elements in each leaf will not exceed L. However, we must guaranty the number of elements in each leaf to be, at least, k because that is a constraint imposed by microaggregation. To that end, we firstly detect the leaves with low-cardinality (*i.e.,* with less than k elements) as shown in Figure 2. Secondly,the centroid of each non-empty leaf is computed using the next Equation:

$$C_g = \frac{\sum_{i=0}^{E_g} x_i^g}{E_g}, \tag{2}$$

where x_i^g is the $i-$th record of leaf g and E_g is the number of records in the leaf.

After computing the centroids, the closest centroid to each low-cardinality leaf centroid is found and the corresponding leaves are fused as shown in Figure 3. This pro-

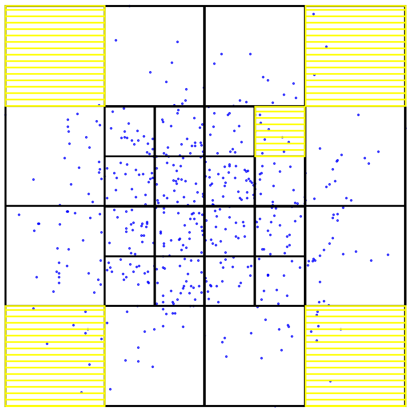

Figure 2. Example of low-cardinality leaves detection with $k = 10$. Highlighted squares indicate low-cardinality leaves.

cess is repeated until there are no low-cardinality leaves. Note that the number of elements contained in a leaf after fusing two of them can be bigger than L. Thus, we can only guarantee the existence of, at least, k elements but we can say nothing about the size of the finally obtained leaves[1].

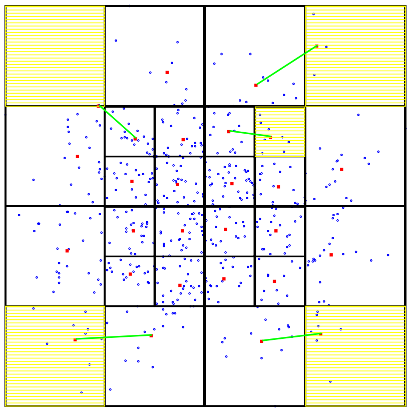

Figure 3. Example of centroid computations and leaves fusion. The lines link the leaves to be fused

2.4 MDAV microaggregation

In this final step we are able to apply the MDAV [5, 7, 13] microaggregation algorithm over each generated group of records obtained from each non-empty leaf.

[1]If the goal is to guarantee the upper limit L, then the fusing step should be avoided or supervised.

Table 3. Number of blocks for a data set with $n = 2.5 \cdot 10^6$ records, $d = 2$ attributes and minimum block size $k = 3$

L	500	1000	2000	5000	10000
Univ.Block.	5000	2500	1250	500	250
2^d-tree	16384	4096	4096	1024	1024

The resultant SSE_{D_i} values obtained from the application of MDAV over each group D_i are accumulated to obtain the final SSE, which is the measure we use to evaluate the result of the microaggregation.

3 Experimental Results

The experiments we have carried out consist in microaggregating a large microdata set using: i) univariate blocking, as described in the Introduction; ii) 2^d-tree based blocking.

In order to compare our method with the plain univariate blocking approach we have generated large data sets. Attribute values were drawn from the $[-10000, 10000]$ range by simple random sampling. Given a block size L, plain univariate blocking of a data set with n records divides the range of the blocking attribute into n/L equal intervals; then the records whose blocking attribute falls into the i-th interval are included in the i-th block.

The number of records in the simulated data sets range from $n = 100000$ to $n = 2500000$, and the number of dimensions d ranges from 2 to 10.

In Section 3.3 we analyze the influence of blocking over the SSE by studying its effects over a small data set that can be microaggregated without blocking. Note that this comparison can only be done by using a small data set. The reason is that MDAV can not deal with very large data sets in a reasonable time without storing a matrix of distances in memory, thus, the size of the data set is limited by the available memory.

3.1 Study of L

L is an important value that has to be tuned by the user in order to properly obtain the groups for microaggregating. Table 3 shows that, while plain univariate blocking always yields n/L blocks, the 2^d-tree method described in this paper results in a number of blocks higher than n/L. Certainly, there are empty blocks among those generated by the 2^d-tree methods. However, if we look only at the useful blocks, their number is still higher than n/L, as shown in Figure 4.

If the purpose of blocking is to apply microaggregation on the resulting blocks, we must bear in mind that the best

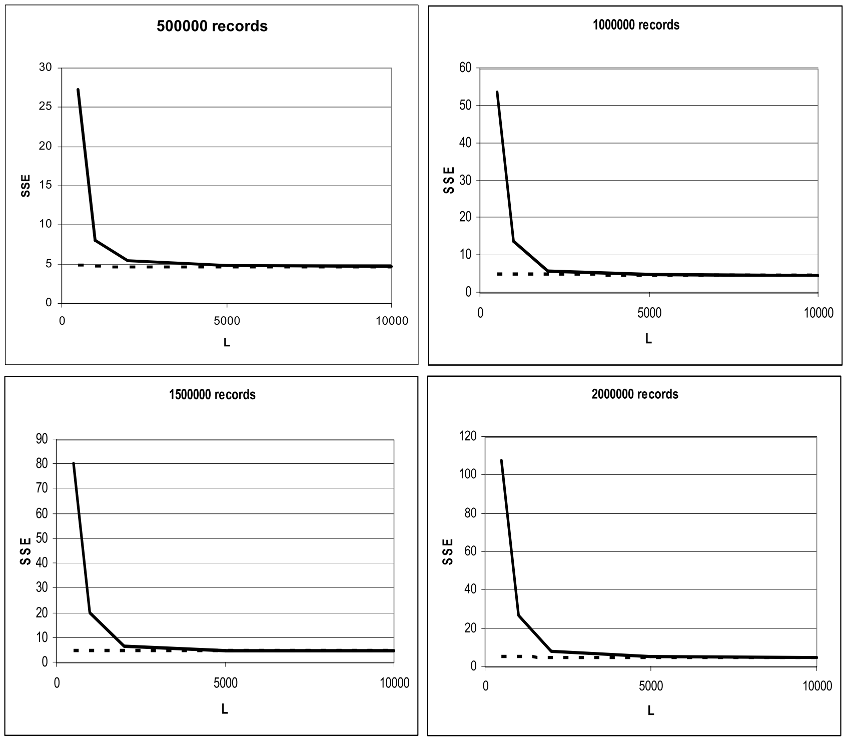

Figure 5. SSE evolution for different L values applied to 500000, 1000000, 1500000 and 2000000 records with 2 dimensions for $k = 3$. The continuous line represents the univariate blocking approach, the dashed line represents the 2^d-tree blocking approach.

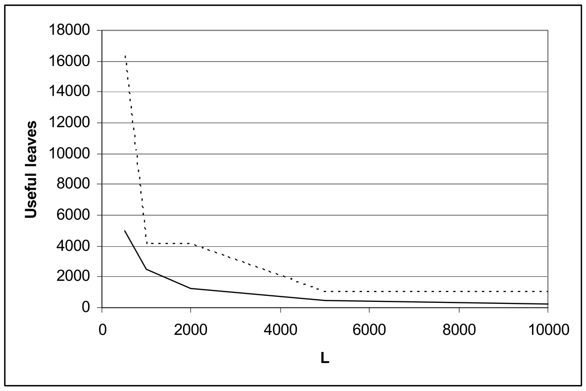

Figure 4. Number of useful blocks for a data set with $n = 2.5 \cdot 10^6$ records, with $d = 2$ attributes and minimum block size $k = 3$. The continuous line represents univariate blocking while the dashed line represents 2^d-tree based blocking

microaggregation heuristics (*e.g.,* MDAV) have a computational cost $O(n^2)$. Thus, sizes of blocks should be as similar as possible, without causing a dramatic increase of the within-group sum of squares SSE as a result of blocking. Our 2^d-tree blocking achieves a trade-off between size balance and SSE after microaggregation. For MDAV microaggregation, Figure 5 shows that the SSE is actually much lower than the one obtained with univariate blocking: the reason is that our method yields blocks which are much more homogeneous than the blocks formed by using one arbitrarily chosen block attribute.

In order to analyze the behavior of L we have studied how the obtained SSE evolves depending on the number of records and the blocking technique used. Specifically, Figure 6 shows that quite similar values of SSE are obtained for a given L whatever the number of records is when the utilized blocking method is based on a 2^d-tree. On the other hand, Figure 7 shows that the SSE grows with the number of records when the univariate blocking is used, specially when L is small.

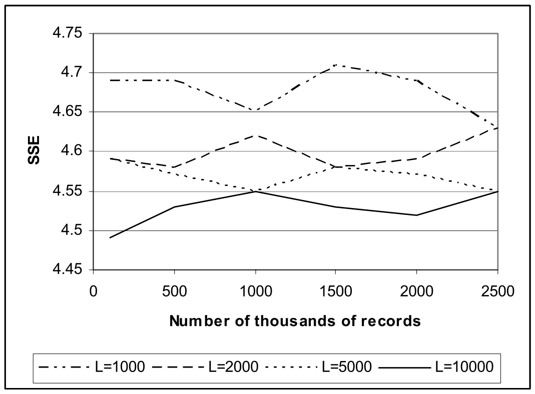

Figure 6. SSE evolution for different values of L and different number of records using $k = 3$ and the 2^d-tree approach before microaggregating.

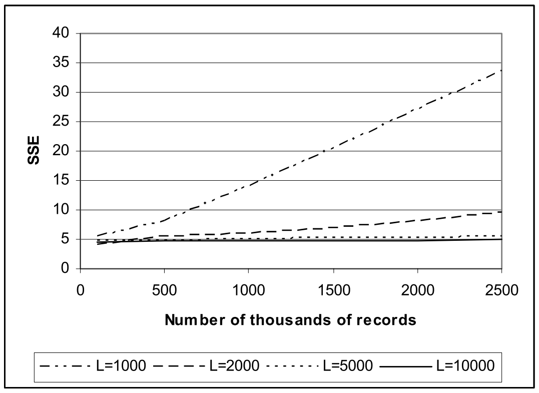

Figure 7. SSE evolution for different values of L and different number of records using $k = 3$ and the univariate blocking approach before microaggregating.

d	2	3	4	5	10
2^d-tree	4.55	593.64	7953.22	37269.3	916119
Univ.Block.	4.9	1155.54	21004.5	99876.3	1985030

Table 4. SSE of the microaggregated data set for $k = 3$ and 2, 3, 4, 5 and 10 dimensions. The data set has 2.5 million records.

L	k	$SSE_{univ.block.}$	SSE_{2^d-tree}
100	3	663.435	456.846
200	3	503.281	464.589
100	5	1651.86	713.095
200	5	1179.25	734.925

Table 5. SSE results for EIA data set (4096 records and $d = 11$).

3.2 The importance of d

The dimensionality d is a key factor in our algorithm because the number of leaves/blocks exponentially increases with d. In fact, as d increases, the advantage of our method over univariate blocking in terms of block homogeneity dramatically increases.

Figure 8 and Table 4 show the evolution of SSE for different values of dimensionality of the data set. It can be observed that both blocking techniques result in a growing of the SSE of the microaggregated data set. However, the microaggregation with the 2^d-tree blocking method always outputs a lower SSE.

3.3 The influence of blocking

In the previous sections we have analyzed the differences between univariate blocking and our proposed 2^d-tree based blocking over very large data sets. As it has been explained in the Introduction, blocking is necessary when working with large data sets. However, it can unintentionally break some natural clusters during the partition of the space. Thus, the resulting SSE could become worse than the obtained without blocking.

In order to study this hypothesis, we use a small data set (*i.e.,* EIA data set), that can be microaggregated without blocking, and we compare the SSEs which are obtained with each technique. The EIA data set [1] has 4096 records with 11 attributes and it has become a usual reference data set for testing multivariate microaggregation [5, 6, 14]. When MDAV is applied over the whole data with $k = 3$ an $SSE = 217.38$ is obtained. When $k = 5$ the resulting SSE is 750.21. Table 5 shows the SSE results over this data set using the studied blocking techniques for different values of L and k.

Comparing the results in Table 5 with the SSE obtained without blocking, we can see that the univariate blocking is clearly disruptive while our 2^d-tree approach makes the SSE to be worse for $k = 3$ but improves it for $k = 5$. This behavior of the SSE can be explained because the EIA data set is known to be naturally clustered for a $k = 5$ and the partition that the 2^d-tree based blocking obtains helps MDAV to improve the resulting SSE.

From these results we can conclude that the univariate blocking technique is more disruptive than the 2^d-tree based one. Moreover, in some cases in which the data set behaves as a clustered data set, the 2^d-tree-based blocking can help MDAV to improve the SSE.

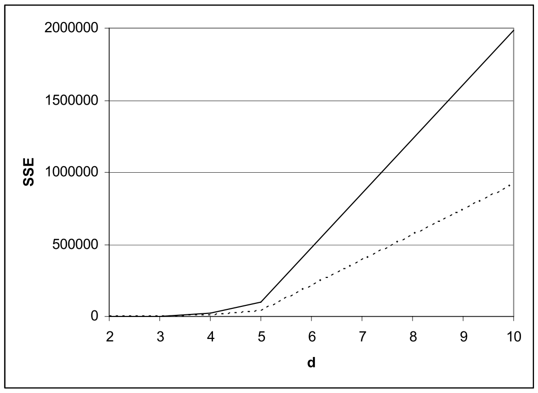

Figure 8. SSE evolution for different d values applied to 2500000 records with dimensions from 2 to 10, for $k = 3$. The value for L is 10000. The continuous line represents the univariate blocking approach, the dashed line represents the 2^d-tree blocking approach.

4 Conclusions

We have presented a new blocking method based on 2^d-trees for intelligently dividing a very large data set prior to microaggregation. We have shown that our method outperforms the previous one based on univariate blocking in terms of SSE. Thus, it can be considered an appropriate tool for microaggregating huge data sets.

Some research lines for a further work remain open:

- Expand the method to non-numerical data sets
- Find the optimal value of L for a given data set
- Analyze the impact of the selected dimensions d over the obtained blocking.

References

[1] R. Brand, J. Domingo-Ferrer, and J. M. Mateo-Sanz. Reference data sets to test and compare sdc methods for protection of numerical microdata, 2002. European Project IST-2000-25069 CASC, http://neon.vb.cbs.nl/casc.

[2] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, MA, USA, 1990.

[3] D. Defays and P. Nanopoulos. Panels of enterprises and confidentiality: the small aggregates method. In *Proc. of 92 Symposium on Design and Analysis of Longitudinal Surveys*, pages 195–204, Ottawa, 1993. Statistics Canada.

[4] J. Domingo-Ferrer, A. Martínez-Ballesté, and J. M. Mateo-Sanz. Efficient multivariate data-oriented microaggregation. *Manuscript*, 2005.

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

[6] J. Domingo-Ferrer, F. Sebé, and A. Solanas. A polynomial-time approximation to optimal multivariate microaggregation. *Manuscript*, 2005.

[7] J. Domingo-Ferrer and V. Torra. Ordinal, continuous and heterogenerous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery*, 11(2), 2005.

[8] A. W. F. Edwards and L. L. Cavalli-Sforza. A method for cluster analysis. *Biometrics*, 21:362–375, 1965.

[9] E. C. for Europe. Statistical data confidentiality in the transition countries: 2000/2001 winter survey. In *Joint ECE/Eurostat Work Session on Statistical Data Confidentiality*, 2001. Invited paper n.43.

[10] A. D. Gordon and J. T. Henderson. An algorithm for euclidean sum of squares classification. *Biometrics*, 33:355–362, 1977.

[11] P. Hansen, B. Jaumard, and N. Mladenovic. Minimum sum of squares clustering in a low dimensional space. *Journal of Classification*, 15:37–55, 1998.

[12] S. L. Hansen and S. Mukherjee. A polynomial algorithm for optimal univariate microaggregation. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):1043–1044, July-August 2003.

[13] A. Hundepool, A. V. de Wetering, R. Ramaswamy, L. Franconi, A. Capobianchi, P.-P. DeWolf, J. Domingo-Ferrer, V. Torra, R. Brand, and S. Giessing. *μ-ARGUS version 4.0 Software and User's Manual*. Statistics Netherlands, Voorburg NL, may 2005. http://neon.vb.cbs.nl/casc.

[14] M. Laszlo and S. Mukherjee. Minimum spanning tree partitioning algorithm for microaggregation. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):902–911, 2005.

[15] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[16] A. Oganian and J. Domingo-Ferrer. On the complexity of optimal microaggregation for statistical disclosure control. *Statistical Journal of the United Nations Economic Comission for Europe*, 18(4):345–354, 2001.

[17] M. Rosemann. Erste ergebnisse von vergleichenden untersuchungen mit anonymisierten und nicht anonymisierten einzeldaten amb beispiel der kostenstrukturerhebung und der umsatzsteuerstatistik. In *G. Ronning and R. Gnoss (editors) Anonymisierung wirtschaftsstatistischer Einzeldaten, Wiesbaden: Statistisches Bundesamt*, pages 154–183, 2003.

[18] A. Solanas, A. Martínez-Ballesté, J. M. Mateo-Sanz, and J. Domingo-Ferrer. Multivariate microaggregation based on a genetic algorithm. *Manuscript*, 2006.

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