ESSNET-SDC Deliverable
Blocking Methods for Microdata Protection

Josep Domingo-Ferrer
Universitat Rovira i Virgili,
Dept. of Computer Engineering and Maths,
Av. Països Catalans 26,
43007 Tarragona, Catalonia
e-mail josep.domingo@urv.cat

October 23, 2008

Contents

List of Figures

List of Tables

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 report presents a new blocking method based on 2^d-trees for intelligently partitioning very large data sets. Blocking makes sense whenever the treatment to be used on the data set is of complexity higher than linear. We take here microaggregation (which has quadratic complexity) as a treatment, but other superlinear treatments can benefit from the blocking method described (like record linkage, also of quadratic complexity).

Chapter 1

Introduction

Some SDC methods for microdata protection are very time consuming when very large surveys have to be protected. Some SDC microdata methods take linear time, while others (like microaggregation or optimal recoding) take time quadratic or, more generally, superlinear in the data set size. Also, disclosure risk assessment for microdata is often superlinear (*e.g.* record linkage is quadratic).

Blocking is a popular approach to applying superlinear methods to large microdata sets. The idea is to split large data sets into smaller pieces (blocks) of manageable size that can be separately treated in a reasonable time. Blocking should be done in such a way that its impact on data utility is as small as possible.

Usual blocking procedures involve selecting a number of variables in the data set, called blocking variables. Then records are sorted by the blocking variables and the sorted data set is divided as many times as needed to obtain manageable subsets. Normally, a block is defined as a subset of records sharing a particular combination of values of the blocking variables. Building blocks from blocking variables has several drawbacks:

- The choice of blocking variables may not be obvious;
- Blocks obtained in this way 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.

Some blocking approaches based on clustering theory have been proposed in the literature [2, 9] mainly designed for use with record linkage. They aim at obtaining blocks that are more similar to the natural clusters present in the data. We propose in this report this type of blocking via clustering and, more specifically, via 2^d trees. In [10], we were able to show that 2^d trees could be used to get very homogeneous blocks and thus reduce information loss caused by blocking.

1.1 Contribution and plan of this report

In this report, we propose a method based on a 2^d-tree for dividing very large data sets into a number of smaller ones. We show the usefulness of the proposed blocking for efficiently microaggregating very large data sets. Other superlinear treatments on the blocks, such as record linkage, can take advantage of this blocking procedure.

The rest of the report is organized as follows. In Chapter 2 our method is detailed. Next, Chapter 3 shows the experimental results. Finally Chapter 4 is a conclusion.

Chapter 2

A blocking method based on clustering

We are addressing a problem that consists in dividing a large data set D into a number of smaller ones $D_1, D_2, \ldots D_G$. 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 want to divide the original data in such a way that separately treating each block is not significantly worse in terms of quality than treating the entire data set.

The treatment considered in this report is microaggregation. In microaggregation, the quality of the microaggregated data is measured by the within-groups sum of squares SSE (the lower it is, the better is the quality of masked data).

Specifically, the MDAV microaggregation heuristic [4, 6, 7] has been used to obtain the empirical results presented in Chapter 3.

A complete explanation about how to design and implement a 2^d-tree can be found in [3]; thus, we will only elaborate on the most relevant aspects of the 2^d-tree generation and the other main steps of the method.

2.1 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 $d = 2$ (*i.e.* Quadtree) and virtual reality or satellite image analysis with $d = 3$ (*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 number of children of each node in the tree

Table 2.1: Notation used in this report

Symbol	Meaning
d	Number of dimensions
D	A data set
$D_1, D_2, \ldots D_G$	The generated blocks
E_g	Number of elements in group g
k	Min. elements per leaf
L	Max. elements per leaf
n	Number of records in D

Table 2.2: Structure of a node in the 2^d tree

```
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
};
```

grows exponentially with d (*i.e.* each node has 2^d children). Figure 2.1 shows a typical partition obtained by applying a 2^d-tree with d equaling 2.

In Table 2.2 the structure of a node is shown using a "C"-like syntax. 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 consists of a single leaf and it is divided into 2^d leaves if the number of records within the leaf upper and lower bounds is greater than L. This division of the leaves into 2^d children leaves is recursively applied until all leaves have a maximum number of records no greater than L within their lower and upper bounds.

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.

There are some computational issues which deserve a comment:

- Each node in the 2^d-tree must know which records fall inside the space

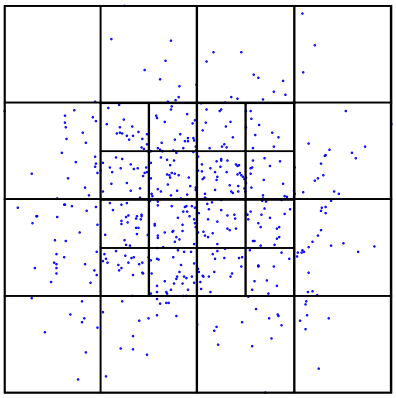

Figure 2.1: Example of a quadtree partition of a data set of 500 elements with $L = 50$.

region 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.

- No distance computations are needed during the generation of the 2^d-tree; only simple and computationally cheap comparisons are required.

2.2 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 guarantee the number of elements in each leaf to be, at least, k because this is a constraint imposed by microaggregation. To that end, we first detect the leaves with low-cardinality (*i.e.* with less than k elements) as shown in Figure 2.2. Then the centroid C_g of each non-empty leaf g is computed as

$$C_g = \frac{\sum_{i=0}^{E_g} x_i^g}{E_g}, \tag{2.1}$$

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 2.3. This process is repeated until no low-cardinality leaves are left. At the end of this second step, the records in each remaining non-empty leaf form a block. Let the resulting blocks be $D_1, D_2, \cdots, D_G$.

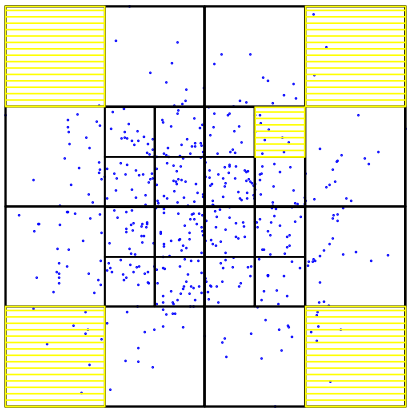

Figure 2.2: Example of low-cardinality leaf detection with $k = 10$. Highlighted squares indicate low-cardinality leaves.

2.3 MDAV microaggregation

In order to demonstrate the usefulness of the proposed blocking scheme, we apply MDAV [4, 6, 7] microaggregation algorithm to the block of records D_i obtained from each non-empty leaf.

The within-groups sum of squares SSE_{D_i} values obtained from the application of MDAV to each block D_i, for $i = 1, \cdots, G$. are accumulated to obtain the overall SSE, which is the measure that will be used to evaluate the impact of blocking on the quality of microaggregation.

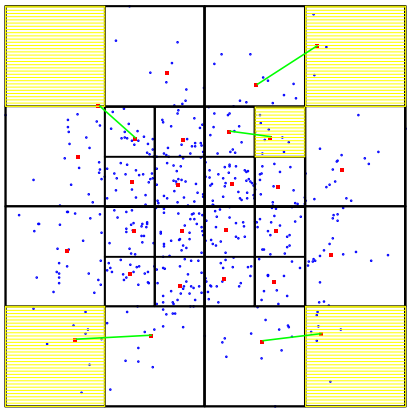

Figure 2.3: Example of centroid computation and leaf fusion. The lines link the leaves to be fused

Chapter 3

Experimental results

The experiments carried out consist of microaggregating a large microdata set using: i) univariate blocking by a single blocking variable (see Chapter 1); 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 ranged from $n = 100000$ to $n = 2500000$, and the number of dimensions d ranged from 2 to 10.

In Section 3.3 we analyze the influence of blocking over SSE by studying its effects on a small data set that can be microaggregated without blocking.

3.1 On the block size L

The (maximum) block size L is an important parameter that must be tuned by the user. Table 3.1 shows that, while plain univariate blocking always yields n/L blocks, the 2^d-tree method described in this report 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 3.1.

Table 3.1: 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

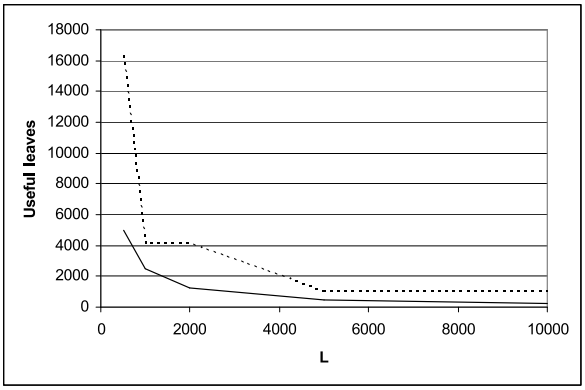

Figure 3.1: Number of useful blocks for a data set with $n = 2.5 \cdot 10^6$ records, $d = 2$ attributes and minimum block size $k = 3$. The continuous line represents univariate blocking while the dashed line represents 2^d-tree based blocking

Since the purpose of blocking is to be able to apply a superlinear treatment on each of the resulting blocks, the sizes of the latter should not be too different. If the superlinear treatment is microaggregation, blocking with similar block sizes should not cause any dramatic increase of the within-group sum of squares SSE. Our 2^d-tree blocking achieves a trade-off between size balance and SSE after microaggregation. For MDAV microaggregation, Figure 3.2 shows that the SSE is actually much lower than the one obtained with univariate blocking: the reason is that our method yields blocks whose records are much more homogeneous than those in the blocks obtained by arbitrarily selecting one blocking 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 3.3 shows that quite similar values of SSE are obtained for a given L whatever the number of records is when the blocking method used is based on a 2^d-tree. On the other hand, Figure 3.4 shows that SSE grows with the number of records when the univariate blocking is used, specially when L is small.

3.2 On the dimensionality 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 3.5 and Table 3.2 show the evolution of SSE for different values of dimensionality of the data set. It can be observed that both blocking techniques

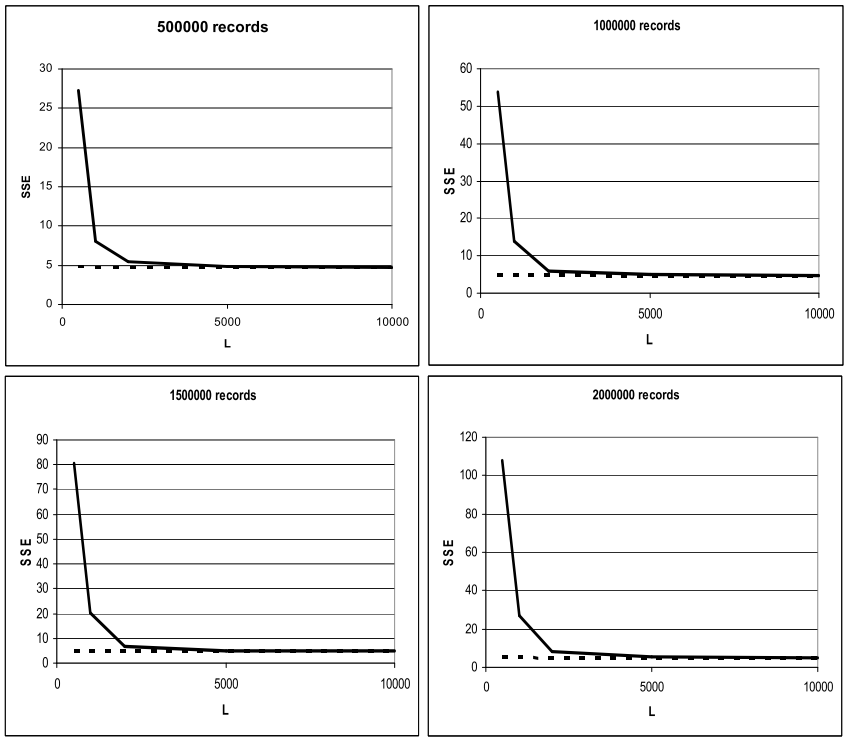

Figure 3.2: SSE evolution for different values of L applied to 500000, 1000000, 1500000 and 2000000 records with 2 dimensions for $k = 3$. The continuous line represents the univariate blocking approach; the dashed line represents the 2^d-tree blocking approach.

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 explained in Chapter 1, 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 SSE obtained after blocking could in principle be worse than the SSE without blocking.

If we take a small data set that can be microaggregated without blocking, we will be able to compare SSEs obtained with and without 2^d-tree blocking. We use the EIA data set [1], which has 4096 records with 11 attributes and has

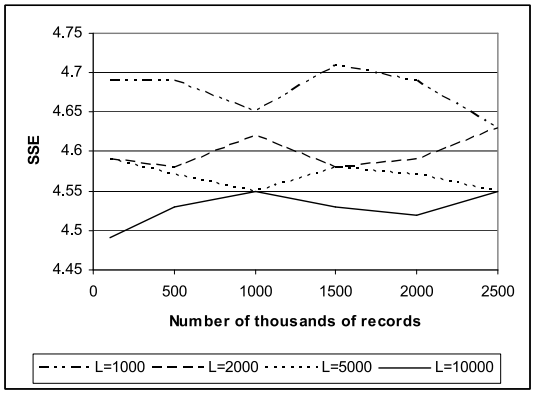

Figure 3.3: SSE evolution for different values of L and different numbers of records using $k = 3$ and the 2^d-tree approach before microaggregation

Table 3.2: 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.

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

become a usual reference data set for testing multivariate microaggregation [4, 5, 8]. When MDAV is applied over the entire data set with $k = 3$ an $SSE = 217.38$ is obtained. When $k = 5$ the resulting SSE is 750.21. Table 3.3 shows the SSE results over this data set using the discussed blocking techniques for different values of L and k.

Comparing the results in Table 3.3 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 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

Table 3.3: SSE results for EIA data set (4096 records and $d = 11$).

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

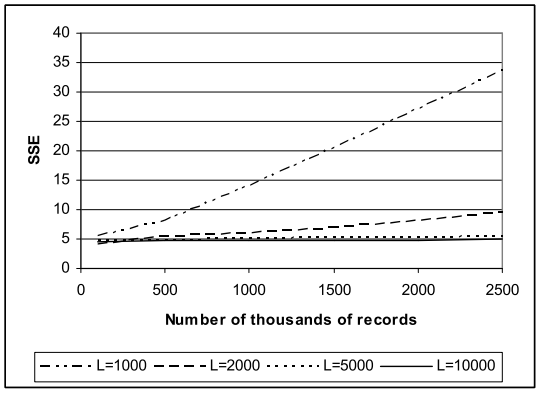

Figure 3.4: SSE evolution for different values of L and different numbers of records using $k = 3$ and the univariate blocking approach before microaggregation

the data set behaves as a clustered data set, 2^d-tree-based blocking can help MDAV to improve SSE.

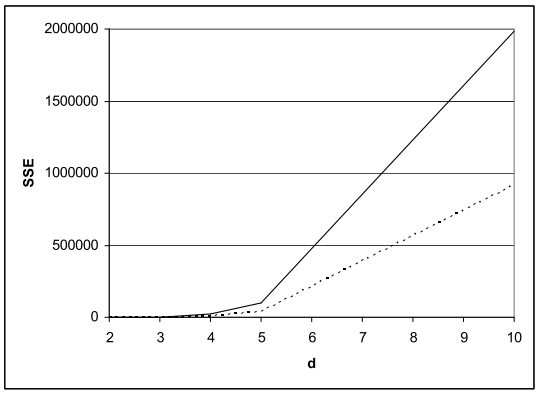

Figure 3.5: SSE evolution for different d values applied to 2500000 records with dimensions from 2 to 10, for $k = 3$. $L = 10000$ is taken. The continuous line represents the univariate blocking approach, the dashed line represents the 2^d-tree blocking approach.

Chapter 4

Conclusions

We have presented a new blocking method based on 2^d-trees to intelligently dividing a very large data set into blocks prior to a superlinear treatment.

For the case where the treatment is microaggregation, we have shown that our method outperforms 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 blocking method for non-numerical attributes;
- Find the optimal value of L for a given data set;
- Conduct empirical work to assess the performance of 2^d-tree blocking for other superlinear treatments, like record linkage.

Bibliography

[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] W. Cohen and J. Richman. Learning to match and cluster high-dimensional data sets for data integration. In *Proc. of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 475-480, 2002.

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

[4] 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.

[5] J. Domingo-Ferrer, F. Sebé, and A. Solanas. A polynomial-time approximation to optimal multivariate microaggregation. *Computers and Mathematics with Applications*, 55(4): 714–732, 2008.

[6] J. Domingo-Ferrer and V. Torra. Ordinal, continuous and heterogenerous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery*, 11(2): 195-212, 2005.

[7] A. Hundepool, A. Van 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.

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

[9] A. McCallum, K. Nigam and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 169-178, 2000.

[10] A. Solanas, A. Martínez-Ballesté, J. Domingo-Ferrer and J. M. Mateo-Sanz. A 2^d-tree-based blocking method for microaggregating very large data sets. In *Proc. of ARES/DAWAM 2006-Dependability Aspects of Data Warehousing and Mining Applications*, IEEE Computer Society, pp. 922-928, 2006.