Engineering Applications of Artificial Intelligence 90 (2020) 103531

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

The Fast Maximum Distance to Average Vector (F-MDAV): An algorithm for k-anonymous microaggregation in big data ☆

Ana Rodríguez-Hoyos [a,b], José Estrada-Jiménez [a,b], David Rebollo-Monedero [b], Ahmad Mohamad Mezher [b], Javier Parra-Arnau [c], Jordi Forné [b,*]

[a] Departamento de Electrónica, Telecomunicaciones y Redes de Información, Escuela Politécnica Nacional (EPN), Ladrón de Guevara, E11-253 Quito, Ecuador
[b] Department of Telematics Engineering, Universitat Politècnica de Catalunya (UPC), C. Jordi Girona 1-3, E-08034 Barcelona, Spain
[c] Department of Computer Engineering and Mathematics, Universitat Rovira i Virgili, Av. Països Catalans 26, E-43007 Tarragona, Catalonia, Spain

ARTICLE INFO

Keywords:
MDAV
k-anonymous microaggregation
Speedup
Data privacy
Big data

ABSTRACT

The massive exploitation of tons of data is currently guiding critical decisions in domains such as economics or health. But serious privacy risks arise since personal data is commonly involved. k-Anonymous microaggregation is a well-known method that guarantees individuals' privacy while preserving much of data utility. Unfortunately, methods like this are computationally expensive in big data settings, whereas the application domain of data might require an immediate response to make "life or death" decisions.

Accordingly, this paper proposes five strategies to simplify the internal operations (such as distance calculations and element sorting) of the maximum distance to average vector algorithm, the de facto microaggregation standard. For the sake of its usability in large-scale databases, they, e.g., reduce the number of operations necessary to compute distances from $3m$ to $2m$, where m is the number of attributes of the data set. Also, the complexity of sorting operations gets reduced from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$ where n is the number of records. Through extensive experimentation over multiple data sets, we show that the new algorithm gets significantly faster. Interestingly, the speedup factor by each technique is not greater than 2, but the multiplicative effect of combining them all turns the algorithm four times faster than the original microaggregation mechanism. This remarkable speedup factor is achieved, literally, with no additional cost in terms of data utility, i.e., it does not incur greater information loss.

1. Introduction

Big data is bringing new, unprecedented business opportunities to companies around the world. Currently, it is possible to collect and process vast amounts of information from which more, new, better and varied customer knowledge is mined. As a result, better decisions can be made in sectors like health care, banking, marketing and transportation (Raghupathi and Raghupathi, 2014; Li et al., 2017; Monaco, 2016).

Despite these benefits, within such an abundance of data, it is common to find personal sensitive information, which poses serious privacy risks. First, in the name of this data revolution, information is more prone to be openly published or shared with untrusted third parties. Also, although identifiers are typically suppressed, other demographic attributes, when combined, can be used to reidentify individuals (Sweeney, 2000; Narayanan and Shmatikov, 2008; AOL, 2006). Thus, sensitive attributes might be easily linked to the subjects to whom the disclosed information corresponds, which might lead to discrimination, retaliation, and blackmail (Assoc. Press, 2017).

Statistical disclosure control (SDC) aims at addressing these privacy issues in the special case of microdata files. Microdata are database tables whose records carry data concerning individual subjects. The typical scenario in microdata SDC is a data curator holding the original data set and perturbing the so-called *quasi-identifier* attributes (i.e., attributes that, in combination, may be linked with external information to reidentify individuals in the data set). The goal is to keep disclosure risk as low as possible, while ensuring that only useful statistics or trends are learned by the recipients of data. One of the most common strategies to keep this risk under control is the "privacy first" approach. Here, the data curator enforces a privacy model, which usually depends on a privacy parameter, to ensure an upper bound on

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.engappai.2020.103531.

* Corresponding author.
E-mail addresses: ana.rodriguez@epn.edu.ec (A. Rodríguez-Hoyos), jose.estrada@epn.edu.ec (J. Estrada-Jiménez), david.rebollo@entel.upc.edu (D. Rebollo-Monedero), ahmad.mohamad@upc.edu (A.M. Mezher), javier.parra@urv.cat (J. Parra-Arnau), jforne@entel.upc.edu (J. Forné).

https://doi.org/10.1016/j.engappai.2020.103531
Received 1 August 2019; Received in revised form 3 January 2020; Accepted 31 January 2020
Available online 10 February 2020

the re-identification risk. Some of the best-known privacy models comprise k-anonymity (Samarati, 2001; Sweeney, 2000) and ϵ-differential privacy (Dwork, 2006).

Privacy models rely on a variety of anonymization mechanisms, the common denominator binding all them is *data perturbation*. Essentially, all such mechanisms modify the original data set to guarantee the chosen privacy model, inevitably at the cost of some loss in data utility (Sankar et al., 2013). Evidently, a balance between privacy and utility should be found so that the protected data are useful in real practice, that is, they approximate well the original data. However, in a big data domain, privacy protection also requires mechanisms to execute in a reasonable amount of time, despite the size of the data.

Examples of anonymization mechanisms include microaggregation, suppression, generalization and noise addition. Among them, k-anonymous microaggregation is a high-utility approach that obfuscates demographic attributes. By carefully aggregating these attributes, a minimum level of distortion must be applied to data. Unfortunately, current microaggregation algorithms entail a very high computational cost when anonymizing big data (Rebollo-Monedero et al., 2013a). Thus, since utility extraction from big data is a priority, and already time consuming, privacy protection might be easily neglected. That is why some works are starting to propose strategies to reduce the running time of privacy enhancing mechanisms while preserving the utility of data.

1.1. Contributions of this work

In this work, we propose and evaluate various improvements to a widely-known microaggregation algorithm for k-anonymity protection, the maximum distance to average vector (MDAV). The fundamental aim of such improvements is to effectively reduce the computational complexity of MDAV. Without yielding any additional loss in data utility, we obtain remarkable reductions in running time by diminishing the number of operations necessary to aggregate data, while protecting the privacy of the individuals involved. The proposed computational improvements aim at adapting high-utility SDC to large-scale data sets with numerical demographic attributes. This approach is interesting since the reduction of the computational cost of privacy protection algorithms may encourage its implementation, especially when computation usually entails important economic costs for companies exploiting big data. More concretely, our contributions are the following:

- We propose five strategies for efficient computation that significantly accelerate the running time of k-anonymous microaggregation for multivariate data, which is ideal for big data scenarios.
- Our improvement strategies are based on algebraic and algorithmic operations. First, we rely on algebraically optimized computation of Euclidean distances. Then, algorithmically, we resort to the precomputation and reuse of distances. Also, we use partial selection of a few $(k-1)$ candidates among a list of nearest and furthest data points without requiring total sorting. When calculating centroids, we eliminate redundant operations of sum, thus considerably reducing the resulting complexity. Finally, we have numerical data represented with single-precision floating numbers in lieu of double-precision for faster computation, with utterly negligible impact in accuracy for the application at hand. Said strategies do not alter the functionality of MDAV but the resulting improvements of execution time do make this algorithm more usable in big-data contexts.
- We conduct an extensive, thorough performance evaluation of our modified MDAV. Through systematic experimentation on three standardized data sets, we find the magnitude of the computational improvements, quantified as the resulting speedup.
- The overall speed of our privacy algorithm is increased by a factor of 400 percent in relation to its original version. In terms of operativity, this would have a significant impact on the economic costs of a company dedicated to exploit big data privately.

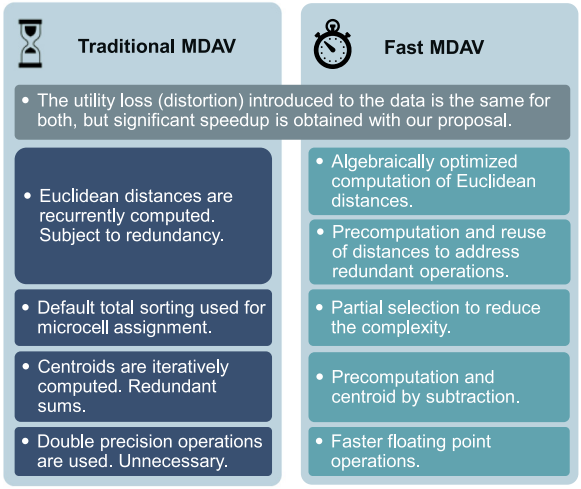


Fig. 1. Summary of the five computational improvements of F-MDAV with respect to traditional MDAV.

- The proposed enhancements do not incur additional information loss. Namely, there is no increase in data distortion. We modify the most complex parts of MDAV so that less computations are needed to obtain the same microcells. As a result, our algorithm generates the same anonymized data that the original MDAV would produce.

Our contributions are summarized in Fig. 1.

1.2. Related work and impact of this approach

The novelty of our contribution lies mainly in our effort to analyze the microaggregation algorithm, i.e., finding the components subject to be accelerated, and devise the mechanisms and algebraic properties that could implement such improvements. Some of said operations could have different applications and part of our contribution is, in fact, tailoring them to the specific case of k-anonymous microaggregation.

Recent works have shown to follow similar approaches, harnessing known mechanisms from different domains (e.g., machine learning), aiming to increase the efficiency of privacy protection algorithms, not only in terms of runtime (Pallarès et al., 2019; Rebollo-Monedero et al., 2019a; Calviño, 2017; Mortazavi and Jalili, 2017), but also in terms of resulting data utility. For instance, Laszlo and Mukherjee (2005) developed an efficient clustering mechanism to deal with large databases while preserving the data utility through a partitioning method of a modified minimum spanning tree. In the same line, Sun et al. (2012) designed an efficient and effective microaggregation mechanism based on calculating the distance among records as the mutual information (entropy) among them. Finally, Mortazavi et al. (2014) introduced fast data-oriented microaggregation (FDM), a method capable of getting multiple protected versions of a large data set (for different values of k) in a single load.

This may suggest that this is a topic of interest. Although utility in this context might have received more attention from researchers, the computational time of privacy algorithms has to do with its *usability* and it is, from our perspective, at least as important.

Although data is the new oil in the big data era, applications of big data are currently possible just because the algorithms that process it can be executed much faster than in the past. However, the execution time is still a bottleneck for some highly demanding applications, which does not favor the implementation of further processing privacy routines. Consequently, we could say that accelerating the execution of privacy protection mechanisms is a fundamental approach to encourage its adoption in the era of big data.

Certainly, besides the runtime acceleration by our method, the null degradation of data utility is part of the added value of our approach. On the other hand, the works previously cited offer reduction of runtime at a cost in data distortion. Ours does not imply any distortion to microaggregated data.

Another not negligible detail about our approach is that it has a multiplicative effect that can be combined with that of other proposals. Namely, if ours (speedup factor, 4×) is applied along with another strategy that multiplies the speed of MDAV by two, in practice, the resulting speedup would be 8×. Naturally, said effect enables an interesting opportunity for capitalizing on several of the efficiency approaches proposed for microaggregation and, particularly for MDAV. Interestingly, these computational improvements can also be combined with the functional (data utility) improvements introduced by other works where microaggregation is involved (Rebollo-Monedero et al., 2017, 2019b; Parra-Arnau et al., 2020)

Another rationale regarding the novelty of our proposal follows. Due to the massification of Internet access, the vast amounts of data containing personal information may grow and change very dynamically, commonly feeding online services. Then, microaggregation, in this context, is likely to be implemented as an ongoing process, running as fast as possible, rather than as a static one-time job. In fact, if data is sufficiently vast, microaggregating it once could be unfeasible in practice for some, e.g., real-time, applications due to the quadratic complexity of MDAV, so optimizing its running time in the big data era seems mandatory.

Inevitably, all these details derived from our contribution have important economic implications. Despite the obstacles behind implementing privacy, some big tech companies are turning their privacy stance into a huge competitive advantage. Thus, the companies that best adapt their operation to privacy requirements (preserving data utility and algorithm usability) will be in better position to exploit such advantage. Interestingly, in this context, increasing the efficiency of privacy protection mechanisms (e.g., reducing their runtime) could become a powerful value generator.

The remainder of this paper is organized as follows. Section 2 reviews the background on k-anonymous microaggregation, particularly focusing on MDAV, and reviews the state of the art in microaggregation algorithms for SDC. Next, Section 3 describes the adaptations we apply to MDAV in order to reduce its running time. Section 4 shows and analyzes the experimental results of our improvements for a variety of data sets. Lastly, conclusions are drawn in Section 5.

2. Background and state of the art

Currently, data is more prone to be shared with external parties or even openly, e.g., for research purposes, in order to better exploit its utility. Thus, malicious observers may illegitimately take advantage, since sensitive information might still be encoded within released data, although previously some information had been suppressed. Unfortunately, conventional privacy services against unintended observers and based on cryptography, fail to address the practical dilemma when the intended recipient of the information is not fully trusted.

As a first approach to protect the anonymity of individuals, it is common to just eliminate their identifiers (e.g., full names or social security numbers) before a data set is published or shared. However, this practice was proved to be insufficient in Sweeney (2000), where it was shown that 87% of the population in the United States could be unequivocally identified solely on the basis of the triple consisting of their date of birth, gender and 5-digit ZIP code, according to 1990 census data. Due to the discriminative potential of a few combined demographic attributes, more sophisticated approaches have been proposed to obscure the identity of the respondents appearing in the released data set.

2.1. Background on k-anonymous microaggregation

Privacy protection techniques usually focus on databases carrying information concerning individual respondents (from a survey or a census). Said databases, also known as *microdata sets*, contain a set of attributes that may be classified into identifiers, quasi-identifiers and confidential attributes. Firstly, *identifiers*, such as full names or medical record numbers, can single out individuals from a data set and are commonly removed to preserve the anonymity of respondents. Secondly, *quasi-identifiers* or key attributes may include age, gender, address, or physical features, which combined and linked with other external information can be used to reidentify respondents. Finally, a data set may contain *confidential attributes* with sensitive information on the respondents, such as salary, health condition, and religion.

In Fig. 2, we illustrate how a perturbed, and thus more private, version of a data set is obtained to be published instead of the original one. In the figure, the original data set combines attributes common in census and medical surveys. It has three quasi-identifiers; age, marital status and ZIP code; and two confidential attributes: annual salary and sexual orientation. The figure at hand shows how, in order to preserve the privacy of respondents, perturbation is applied to quasi-identifiers. This technique, called microaggregation, is applied to enforce k-anonymity (Sweeney, 2000), a privacy model that guarantees that each tuple of key-attribute values is identically shared by at least k records in a data set. Rather than making the original table available, a perturbed version is published where aggregated records of quasi-identifying values are replaced by a common representative tuple. The result is a microaggregated data set that may prevent reidentification attacks.

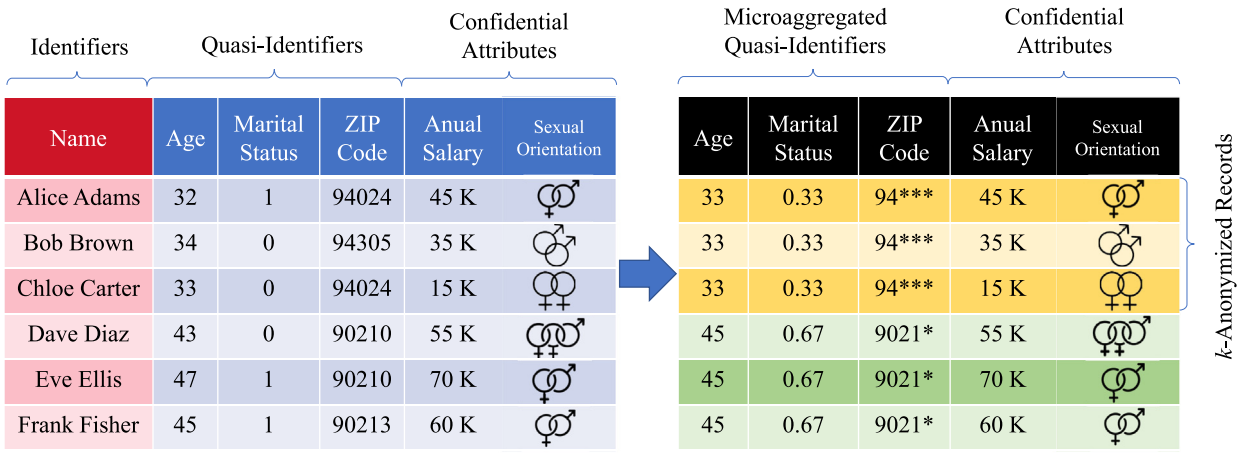
As illustrated in Fig. 3, if tuples of key attributes in a data set could be represented as points in the Euclidean space, k-anonymous microaggregation would consist in partitioning these points in cells of size k, and quantizing each cell and its elements with a representative point. Perturbed key attributes would be characterized by the set of representative points. The de facto standard for numerical microaggregation is the MDAV algorithm. It was proposed in Hundepool et al. (2003) as a practical evolution of a multivariate fixed-size microaggregation method and conceived in Domingo-Ferrer and Mateo-Sanz (2002). MDAV is one of the best-known and most widely used fixed-size microaggregation algorithms for numerical data due to its high performance both in terms of running time and the resulting (even empirical) utility of anonymized data (Rodríguez-Hoyos et al., 2018; Templ, 2014).

We provide, in Algorithm 1, a simplified version of that given in Domingo-Ferrer and Torra (2005) and termed “MDAV generic”.

From the pseudocode of Algorithm 1, we can infer that the main operations of MDAV involve the calculation of centroids (averages), the calculation of distances, and numerical sorting. Mainly due to sorting operations, MDAV shows a quadratic computational complexity; more specifically, its running time grows as $\frac{n^2}{k}$, where n is the number of records of the data set. Thus, in general, if the number of records is very large, the microaggregation process would take too long and might not be feasible.

2.2. State of the art of k-anonymous microaggregation and sorting operations

With microaggregation, as with generalization and suppression, distortion is applied to key attributes to satisfy the k-anonymity privacy model (Samarati, 2001; Sweeney, 2000). This model guarantees that each individual’s information contained in a released data set cannot be distinguished from that of at least other $k-1$ individuals whose information also appears in the data set. The original formulation of k-anonymity as a privacy criterion was modified into the microaggregation-based approach in Defays and Nanopoulos (1993),



Name	Age	Marital Status	ZIP Code	Anual Salary	Sexual Orientation
Alice Adams	32	1	94024	45 K	⚥
Bob Brown	34	0	94305	35 K	⚥
Chloe Carter	33	0	94024	15 K	⚥
Dave Diaz	43	0	90210	55 K	⚥
Eve Ellis	47	1	90210	70 K	⚥
Frank Fisher	45	1	90213	60 K	⚥

Age	Marital Status	ZIP Code	Anual Salary	Sexual Orientation
33	0.33	94***	45 K	⚥
33	0.33	94***	35 K	⚥
33	0.33	94***	15 K	⚥
45	0.67	9021*	55 K	⚥
45	0.67	9021*	70 K	⚥
45	0.67	9021*	60 K	⚥

Fig. 2. Example of k-anonymous microaggregation of published data with $k = 3$, showing probably sensitive information (annual salary and sexual orientation) as confidential attributes, in relation to demographic variables (age, marital status and ZIP code) as quasi-identifiers. The original table codifies the marital-status attribute with 1 and 0 to refer to "married" and "single" respectively. When microaggregation is applied, the marital-status attribute is anonymized by replacing the values in a cell by the proportion of married people in that cell, as MDAV does. For example, in the first cell (first three instances) of the microaggregated table, the marital-status attribute corresponds to 1/3, since, in the original table, one of the three people in that cell is married.

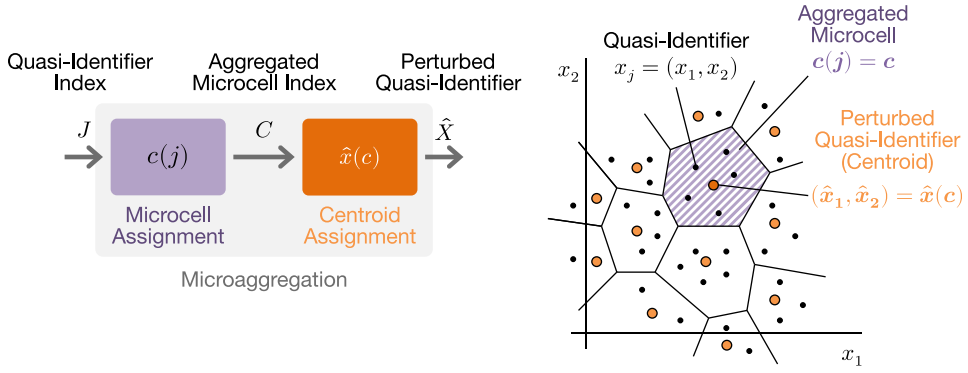


Fig. 3. k-Anonymous microaggregation as a minimum-distortion quantizer design problem with a constraint on the size of the quantizer cells (Rebollo-Monedero et al., 2011).

Algorithm 1 MDAV "generic", functionally equivalent to Algorithm 5.1 in [26].

function MDAV
input k, $(x_j)_{j=1}^n$ ▷ *Anonymity parameter k, quasi-ID portion $x_1, \ldots, x_n \in \mathbb{R}^m$ of a dataset of n records*
output q ▷ *Assignment function from records to microcells $j \mapsto q(j)$*

while $2k$ points or more in the dataset remain to be assigned to microcells **do**
  find the centroid (average) C of those remaining points
  find the furthest point P from the centroid C, and the furthest point Q from P
  select and group the $k-1$ nearest points to P, along with P itself, into a microcell, and do the same with the $k-1$ nearest points to Q
  remove the two microcells just formed from the dataset
if there are k to $2k-1$ points left **then**
  form a microcell with those and finish
else ▷ *At most $k-1$ points left, not enough for a new microcell*
  adjoin any remaining points to the last microcell ▷ *Typically nearest microcell*

Domingo-Ferrer and Mateo-Sanz (2002), Domingo-Ferrer and Torra (2005) and Domingo-Ferrer et al. (2008).

Although k-anonymity is a very popular privacy criterion, it is not flawless. Since the criterion strictly operates with key attributes, the statistical properties of confidential attributes are neglected . In general, k-anonymity overlooks the knowledge a potential attacker may already have or obtain about the data set, giving rise to similarity, skewness or background-knowledge attacks (Domingo-Ferrer and Torra, 2008; Rebollo-Monedero et al., 2010, 2013b). Stricter criteria addressing similarity attacks are p-sensitive (Truta and Vinay, 2006; Sun et al., 2008) and l-diversity. Other criteria such as t-closeness (Li et al., 2007), δ-disclosure (Brickell and Shmatikov, 2008), and average privacy risk (Rebollo-Monedero et al., 2008, 2010) deal with similarity and skewness attacks by imposing further requirements on the semantics and distribution of confidential attributes.

Evidently, the k-anonymous microaggregation process entails distortion or information loss on the data since the original data is modified. The usual criterion to quantify the distortion of microaggregated data is the mean-squared error (MSE) for numerical key attributes representable as points in the Euclidean space. MSE can be computed as

$$\mathrm{MSE} = \sum_{j=1}^{n} \left\| x_j - \hat{x}_j \right\|^2,$$

where n is the number of records of the data set, m is the number of attributes of each record, $x_j \in \mathbb{R}^m$ is the jth record, and $\hat{x}_j$ is the tuple representative of the jth record. While MSE is the general metric of the degradation of data utility after microaggregation, other more empirical utility metrics exist (e.g., accuracy of learning models) but their application may be limited to certain types of data sets. In general, the implementations of microaggregation have been oriented to significantly reduce the inherent distortion due to perturbation (Lin et al., 2010; Matatov et al., 2010; Domingo-Ferrer and González-Nicolás, 2010); as a result, this privacy mechanism can be considered part of the high-utility SDC spectrum. However, such benefit in utility commonly derives in more sophisticated and significantly costlier implementations in terms of computational time (Rebollo-Monedero et al., 2013a).

At a time like the present in which data is generated and processed at a large scale, these new issues in computational complexity must be considered in order to guarantee that privacy protection algorithms are usable in real practice. Notwithstanding, this new dimension of usability has not been addressed in the realm of SDC. To the best of our knowledge, the only approach in this respect is presented in Mohamad Mezher et al. (2017), and here we offer a substantial extension of such work.

From the pseudocode in Algorithm 1, it is straightforward to see that the running time t of the algorithm grows as $t(n) = \frac{n^2}{k}$ (in time units relative to the case $\frac{n^2}{k}$) for $n \gg k$, i.e., it has a quadratic complexity, which implies a superadditive running time. On the one hand, said complexity could make MDAV difficult to apply on big data contexts since its running time would grows significantly as the square of a large number of records. On the other hand, superadditivity is a very useful characteristic of high-utility microaggregation that opens the door to further strategies to reduce the running time of MDAV, e.g., the "divide and conquer" algorithmic approach. This superadditivity can be particularly exploited in sorting operations.

Sorting is repetitively used by MDAV to build k-anonymous groups, based on the "distance" of a reference tuple to the rest of tuples. Consequently, sorting may represent an important part of the execution time of the microaggregation algorithm.

Interestingly, since MDAV only requires finding the $k-1$ nearest tuples (see Algorithm 1), strict sorting (total sorting) is not necessary, but a more relaxed, less complex, operation called partial sorting that finds closest or furthest points faster.

Sorting problems are implemented through two main efficient algorithms: quicksort (Hoare, 1961a), for total sorting, and quickselect (Hoare, 1961b), for partial sorting. While quicksort has, in average, a complexity $\mathcal{O}(n \log n)$ when sorting n items, quickselect presents a complexity $\mathcal{O}(n)$. Thus, choosing quickselect or any of its improved versions could significantly reduce the running time of MDAV. Other versions of quicksort and quickselect (Musser, 1999; Williams, 1964) have been introduced to improve their performance both on the average and worst cases, giving rise to very efficient sorting applications.

The use of these algorithms and their adapted versions is pretty wide in computer science. For instance, to implement the C++ `sort` and `nth_element` functions, the C standard library uses, respectively, a refinement of quicksort called introsort (Musser, 1999) and a refinement of quickselect called introselect (Musser, 1999). The algorithm introsort involves executing quicksort for the average case and switching to heapsort when the worst case is detected. The same combination is done for introselect where, for its worst case, quickselect switches to the Floyd-Rivest algorithm (Floyd and Rivest, 1975a,b; Kiwiel, 2005).

Although not documented since it is a proprietary application, the `sort` command in Matlab R2017b surely derives from the quicksort algorithm or any of its variants such as introsort. Similarly, it is reasonable to think that the Matlab `mink` command, which selects the ordered k smallest values from a list, is internally executing introselect. In Section 4 we give more details on the impact of a more efficient sorting algorithm on the running time of MDAV.

As a final note, the strategies developed in this study could be synergically combined with the proposals presented in Rebollo-Monedero et al. (2017) and Rebollo-Monedero et al. (2019b) where both functional and even further computational improvements are introduced.

3. Computational improvements on MDAV

As described in Section 2, MDAV (see Algorithm 1) creates partitions or microcells from a data set by aggregating neighboring records. Since MDAV operates with numerical attributes, each record is seen as an m-dimensional point ($x_s \in \mathbb{R}^m$) in the Euclidean space, being m the number of attributes of the data set. Note that the microaggregation process iteratively extracts pairs of cells while $2k$ points or more in the data set remain to be assigned. First, a centroid C is calculated as the average of the remaining points. Then, from C, two points P and Q are found from the data set, which serve as references to obtain the neighboring points of each of the two new microcells: one formed by P with its $k-1$ nearest points and another by Q with its $k-1$ nearest points. P is obtained as the furthest point from C (the maximum distance to average vector) and Q as the furthest point from P.

The previous description reveals a set of mathematical operations over the records of the data set. These operations mainly involve centroids calculation, distances calculation, and sorting. Since these operations are used repetitively and executed over a vast number of tuples, there is an interesting chance for improvement in the overall performance of MDAV. Next, we describe in detail the improvements we propose on these operations; Table 1 summarizes the MDAV tasks improved in this work, the respective strategy followed, and gives a brief description of each one. In Section 4, we show the benefits of these strategies through extensive experimentation.

3.1. Algebraic improvement

From line 3 of Algorithm 1, we can devise that much of the MDAV runtime is intended to calculate distances in three moments: when finding the furthest point P from centroid C, when finding the furthest point Q from P, and when obtaining the $k-1$ nearest points from P and from Q to build two microcells.

The fact that distances between the (in some cases the same) points of a data set are continuously calculated turns each iteration very redundant. The computation complexity of MDAV, then, derives from such redundancy, which we tackle through this improvement.

Since MDAV considers each record of the data set as an m-dimensional point in the Euclidean space, a distance D_j between a reference point x_0 (which is C, P and Q, depending on the moment) and a collection of points x_j for $j \in \{1, \ldots, n\}$ is calculated as a quadratic Euclidean distance, i.e.,

$$D_j = \left\| x_j - x_0 \right\|^2.$$

Then, to get D_j, for each m-dimensional point x_j, an element by element subtraction (m operations) and a square norm ($2m-1$ operations) must be calculated. That is, a total of $3m-1$ operations for each point of the data set.

To reduce the resulting runtime, we consider finding an analogous expression to calculate D_j such that less operations are performed. In this case, we harness the polarization identity of the inner product to

Table 1
Summary of computational improvements for MDAV.

MDAV task	Improvement strategy	Description
Distance calculation	Algebraic modification, precomputing	We propose using a property of the inner product to calculate and compare distances such that less operations are needed for microaggregation. Being these algebraic operations, their implementation is usually even optimized in multiple computing libraries.
Sorting	Use partial sorting (or selection)	Since MDAV does not strictly require total order when finding the $k-1$ shortest points to a reference, a more relaxed version called partial sorting can help MDAV save computing resources.
Assignment of microcell	Reuse of distance calculations	Given that much of the running time of MDAV is devoted to calculating distances among the tuples of a data set and a reference point (to obtain its shortest and furthest points), such distances could be precomputed and then reused to prevent redundant operations.
Centroid calculation	Precomputation and reuse of calculations	We propose modifying the calculation of centroids in MDAV such that the redundant addition operations are eliminated.
All calculations in general	Use less precision in the calculations	Since MDAV operations may not require much precision to build microcells, we propose changing the numerical representation to single precision such that less bits be processed, thus implying a reduction in MDAV's running time.

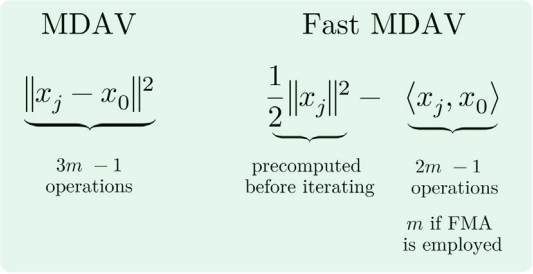


Fig. 4. Representation of the distance calculation performed for each m-dimensional point x_j when k-anonymous microcells are built. We can see that our approach Fast MDAV is able to reduce the number of operations from $3m-1$ to $2m-1$ for each of these n records. Furthermore, since the inner product $\langle x_j, x_0\rangle$ is subject to optimization if FMA is used, the number of operations could be even reduced to m.

put the expression of D_j in terms of the inner product of x_j and x_0. So we expand the last expression such that

$$\left\|x_j - x_0\right\|^2 = \|x_j\|^2 + \|x_0\|^2 - 2\langle x_j, x_0\rangle .$$

If both sides of the last equation are subtracted $\|x_0\|^2$ and multiplied by $1/2$, we obtain

$$\frac{1}{2}\left(\left\|x_j - x_0\right\|^2 - \|x_0\|^2\right) = \frac{1}{2}\|x_j\|^2 - \langle x_j, x_0\rangle .$$

Although the expressions on both sides of the equation no longer represent the real value of D_j, they are a metric still useful to compare distances since they were summed and multiplied by a constant. Thus, when the calculation of distances in MDAV is used to determine the furthest point from a fixed point x_0, we can safely use the right part of the last expression for comparison issues.

Conveniently, for each compared point x_s, the value of $\frac{1}{2}\|x_j\|^2$ can be precomputed once before MDAV is initiated, out of the redundant process, and avoiding significant recalculation in every iteration of MDAV. Thus, in this case, the distance comparison is reduced to the calculation of the inner product $\langle x_j, x_0\rangle$, which consists in an element by element multiplication, i.e., m operations, and a sum of the resulting m terms, i.e., $m-1$ operations, for each point x_j. For this representation of distances, we have a grand total of $2m-1$ operations.

By analytically operating on an expression, we get less operations than the original expression of distance D_s. More precisely, the number of operations is reduced from $3m-1$ to $2m-1$ for each distance calculation where m is the number of quasi-identifiers of each record in the data set.

Not only the number of operations is reduced, but they are algebraic in nature, and that is something for which much of the current code is optimized (e.g., in Matlab or the C standard library). Consequently, it is reasonable that Matlab uses vectorized code or more efficient CPU instructions, that is, advanced vector instructions (AVX) through the Intel math kernel library (MKL). In fact, there are instructions that compute an accumulated sum and a product, designed for the efficient computation of vector and matrix products, called multiply-accumulate operations and fused multiply–add (FMA), which are included in certain Intel processors (e.g., Haskell). Interestingly, if FMA were implemented, our proposal would lead to reduce the number of operations here analyzed to m.

In Fig. 4, we summarize the analysis carried out in the last paragraphs. Finally, this improvement lends itself to the implementation of MDAV in graphics processing units (GPU).

3.2. Distance reuse

The high utility offered by MDAV comes from smartly grouping the closest points into microcells. As already pointed out, this process certainly involves several steps where distances need to be calculated. In this section we concentrate on steps 3 and 4 of Algorithm 1 with the aim of reducing the runtime of MDAV.

We can see that, when aggregating a microcell, given a reference point P, two distance-calculation operations are performed. First, to find the $k-1$ nearest points to P necessary to form a microcell, the distances from all points to P have to be calculated previously. Afterwards, the furthest point Q from P is needed to serve as a reference for building a new microcell; this also entails calculating distances to P.

Evidently, for both steps, the calculations of distances to P can be calculated once and reused. Here, our proposal is using the distances from every point x_s to P both to find the points nearest to P and to find the furthest point from P.

3.3. Partial sorting

Sorting is another time-consuming operation within the original version of MDAV. It is recurrently implemented, e.g., to find the points closest to a given centroid in order to establish the most appropriate members for each microaggregated cell, as posed in line 4 of Algorithm 1.

Finding the points closest to a centroid C implies sorting all the distances from those points to C upwards (total sorting) and then getting the first ones, i.e., the shortest ones. Interestingly, in the case of MDAV, such total sorting is not necessary because only the $k-1$ nearest points are required. In fact, their corresponding distances to the centroid do not even need to be ordered. Finding the k smallest elements implies a more relaxed sorting approach called *partial sorting* or partial selection.

In computer science, total sorting is extensively implemented through the quicksort algorithm (Hoare, 1961a) while partial sorting through the quickselect algorithm (Hoare, 1961b). The computational

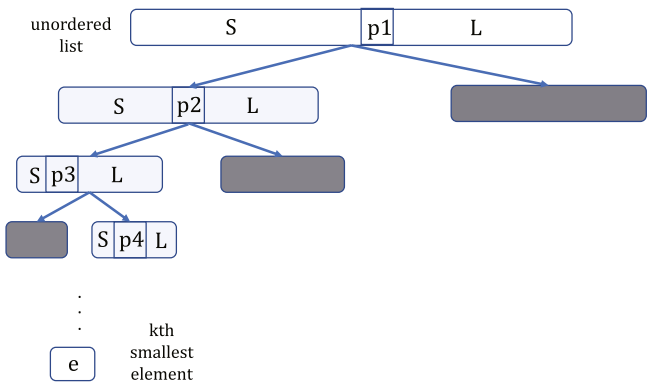


Fig. 5. Brief depiction of the recursive steps carried out for the quickselect algorithm. To find the kth element from an unordered list, quickselect starts by randomly choosing a pivot that partitions the list into two parts: the left one with the elements smaller than the pivot and the right one with the elements larger than the pivot. This process is applied again only on the part where the searched element lies. Finally, all this operation is recursively executed up until the kth smallest element is found. The gray blocks represent the part of the data where the algorithm is not executed (unlike quicksort), thus significantly reducing redundancy.

complexity of the sorting task when using quickselect may be significantly reduced, since it finds the kth smallest number in an unordered list, which does not require total order. This relaxation makes quickselect's problem a much easier one.

As expected, our proposal in this paper is to resort to the use of an implementation of partial sorting (e.g., quickselect) in MDAV when the phase of microcell assignment is performed.

Quickselect is a selection algorithm by which a single element, the kth smallest, is found from a list. Its approach starts by randomly selecting a pivot element that will partition the elements in two; the elements smaller than the pivot on the left and the larger ones on the right. Then, this same approach is recursively implemented only into the side where the element being searched lies up until a single element is obtained. On the other hand, total sorting implemented through quicksort applies the aforementioned approach on both branches, which significantly increases the computational cost. Fig. 5 offers a brief scheme of the extensive reduction of computation complexity when using quickselect instead of quicksort.

By using quickselect, the average complexity of the operations in question is reduced from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$ on the average case. This is very convenient for a context such as big data where millions of records may have to be processed.

3.4. Centroid by subtraction

MDAV builds on another critical operation, centroid calculation (line 2 of Algorithm 1). At each iteration of MDAV, a centroid C is obtained to then serve as a reference in the construction of two microcells (line 3 of Algorithm 1). Every time a couple of microcells are created, their corresponding microaggregated tuples are removed from a stack (line 5 of Algorithm 1) and a new centroid is calculated using the remaining tuples, to build other two microcells. A centroid is calculated, at the beginning of every iteration, by averaging the remaining tuples, which simply consists in adding up all these tuples and dividing by the number of tuples. However, in this iterative process, several tuples of the data set get added again and again multiple times, which definitely results in redundant work and thus running time that can be saved.

In order to accelerate the execution of MDAV, we can modify the calculation of centroids so that the redundant operations of sum are eliminated. To this end, we propose to calculate centroids "by subtraction". Accordingly, we first calculate the sum $S = \sum_{j=1}^{n} x_j$ of all the tuples of the data set $(x_1, \ldots, x_n)$. Moreover, we keep track of the tuples being aggregated by adding them up in S' as soon as they are assigned a microcell. Conveniently, subtracting S' from S has the same effect as obtaining the sum of all the tuples not already aggregated for each iteration so said subtraction can be used to calculate centroids as $C_s = \frac{S - S'}{n_s}$. The benefit evidently lays in that unnecessary adding operations are not done. Finally, note that initially precomputing the sum of all the tuples of the data set does not represent significant complexity since it is only done once.

3.5. Single precision

In computer hardware, numerical data is represented with a number of bits that define the precision of calculations. These options include single precision, where 32 bits are used, and double precision, which uses 64 bits.

Due to the higher computing capabilities of modern hardware, most of the algorithms are implemented using double precision as a standard. However, if single precision could be implemented, we could speedup the execution of such algorithms since less bits would have to be processed. Consequently, given that the standard version of MDAV performs a series of mathematical operations over numerical values, we propose to use single precision for them in order to accelerate the microaggregation process.

This is the only modification that might imply a change in the results of the calculations performed by MDAV. Notwithstanding, since this algorithm might not require extremely precise operations (in terms of the number of decimal points considered), we expect no significant changes in the structure of microcells obtained with respect to the original version of MDAV, but a faster k-anonymous microaggregation.

4. Experimental evaluation

In this section, we evaluate the efficiency of the proposed computational enhancements to MDAV. The objective of evaluation through experimentation of our approach is mainly to determine its impact on microdata. As mentioned in previous sections, such impact can be measured in terms of the algorithm's speedup and the resulting data distortion (although in our case it is unlikely to occur) spawned by F-MDAV. Another objective is finding out whether such effect is independent of the data set employed.

4.1. Evaluation criteria and data sets

We conduct this evaluation across two dimensions: *speedup* and *performance*. Speedup is measured as time gain, while performance is measured as the additional distortion incurred by the adapted versions of MDAV. However, since most of the proposed modifications do not change the internal operations of the algorithm, there is no additional distortion in the data and thus we mainly focus on measuring speedup. Below, we describe the experimental setup and our results from systematic tests over a variety of data sets. Such results are depicted for each enhancement and data set in Figs. 6, 7, and 8, while the overall speedup is illustrated in 11.

The evaluation of the computational performance of our methods has been conducted with three *standardized* data sets. These real data sets include "Large Census", "Quant Forest" and "USA House", which were previously used in Solé et al. (2012) and Pallarès et al. (2019). The "Large Census" data set has 149,642 records and includes 13 numerical attributes; "Quant Forest" has 581,012 records, from which we use a random sample of 150,000 records, and 10 numerical attributes. The "USA House" data set has 5,967,303 records and 13 numerical attributes. We use the "Large Census" data set since it is extensively employed in SDC, whereas "Quant Forest" and "USA House" data sets are used to validate the results obtained in "Large Census". For our study, all attributes are considered to be quasi-identifiers.

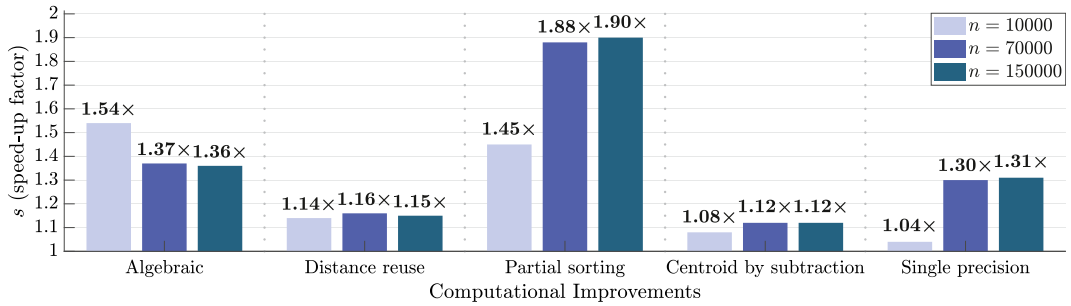


Fig. 6. Speedup factor (s) of each of the five proposed improvements, i.e., when applied individually on the Large Census data set.

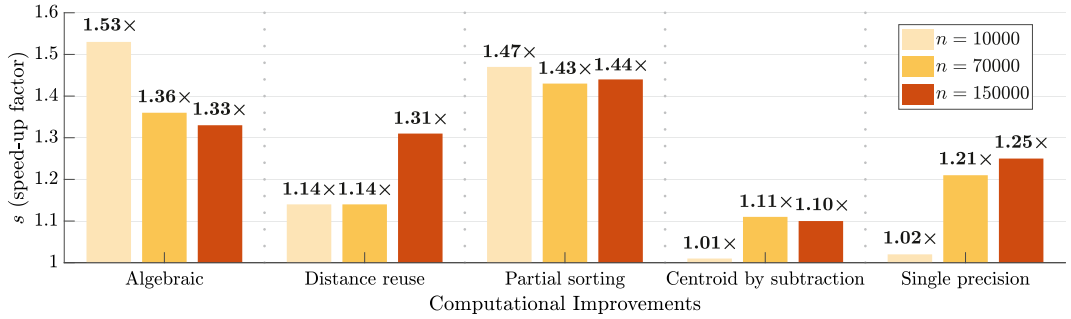


Fig. 7. Speedup factor (s) of each of the five proposed improvements, i.e., when applied individually on the Quant Forest data set.

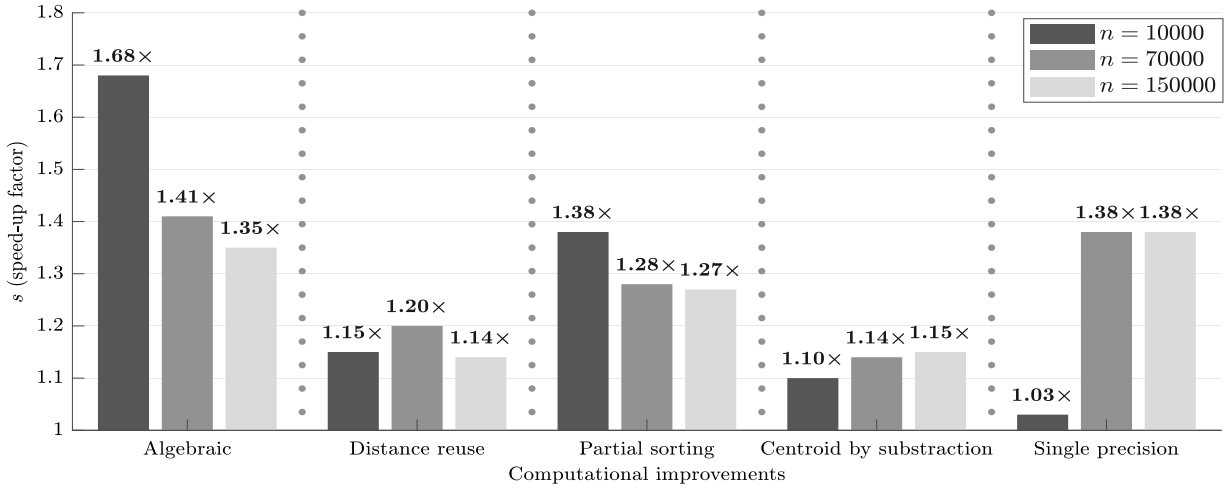


Fig. 8. Speedup factor (s) of each of the five proposed improvements, i.e., when applied individually on the USA House data set.

4.2. Experimental methodology

The experiments described in this section have been run in an Intel Core i7 CPU 3.4 GHz with 32 GB RAM. The microaggregation algorithm MDAV, its adapted versions, and the measurement tests are implemented entirely in Matlab R2017b, where, for the sake of fairness, we disable any form of parallelization.

In general, all versions of MDAV are parameterized with $k = 10$, which implies a reasonable level of privacy without incurring a significant distortion of data. Moreover, before microaggregation is applied, we follow the common practice of normalizing each column of the data set to have zero mean and unit variance.

To find the speedup obtained by our improvements, we measured the running time of MDAV before (Domingo-Ferrer and Torra, 2005) and after implementing our refinements. Our reference MDAV is the algorithm specified in Algorithm 1. We shall refer to it as *traditional MDAV*. Furthermore, the modifications proposed in this work are applied to MDAV *individually*, with the aim of measuring their separate contribution to the speed of the microaggregation algorithm. Also, to show the combined effect of the five improvements, we implement them in a version of the algorithm we call *fast MDAV*.

Our experiments rely on a speedup to show how faster MDAV may become due to the proposed computational improvements. Let t_0 be the running time of traditional MDAV and t the running time of any improved version of MDAV (including fast MDAV). Essentially, the speedup factor $s = \frac{t_0}{t}$ tells us how fast this version is with respect to traditional MDAV. For instance, consider $t_0 = 15$; if, after adapting the algorithm, its running time were reduced to $t = 5$, we would have gotten a speedup factor of $s = 3\times$, i.e., an MDAV that is 3 times faster.

Regarding our experimental *methodology*, we have a few final remarks. First, we assess MDAV over a varying number of records n with the aim of verifying the impact of our methods when the size of the data is increased. Thus, from each data set, we extract portions of data of varied sizes (different values of n). For each value of n, the running time we measure is the averaged time that it takes MDAV to microaggregate

n records. To that end, for every data set, we systematically obtain 3 random samples of n records each and then average the corresponding running times of MDAV. The running times for every improvement are registered and then compared with the time of traditional MDAV through the aforementioned speedup factor s.

The results of our computational strategies are presented in 4 bar charts; the first 3 illustrate, for each tested data set, the speedup factor obtained by every method. Although we experimented with several lengths, for the sake of visibility, the results are shown only for 3 representative values of n (10,000, 70,0000 and 150,000). In the same line, the last bar chart exhibits the speedup factor reached by the fast MDAV, i.e., when all the improvements are consolidated within the same MDAV implementation. Although in essence the modifications we propose to MDAV do not imply a change in the numerical results of its internal operations, we verify whether or not each improvement leads to a variation of the built microcells or an increased distortion with respect to traditional MDAV.

4.3. Experimental results

In the following subsections, we depict and explain the results of our experiments.

4.3.1. Algebraic improvement

As already explained in Section 3.1, this method reduces the number of operations needed to calculate distances by taking advantage of a property of the inner product. Remarkably, numerical libraries are usually optimized for these algebraic operations.

Consequently, the results of our experiments show a significant speedup of MDAV that could reach a factor of 1.54×. This is depicted in Figs. 6, 7, and 8, for the three data sets we use, over which a homogeneous computational improvement is revealed.

Along with the partial sorting improvement, this algebraic strategy presents the best performance in terms of running time. In addition, the results of the cell assignment function $c(j)$ and the resulting distortion of this new version of MDAV remain unchanged with respect to traditional MDAV.

4.3.2. Distance reuse

Given the recurrence of distance calculations in MDAV, its running time can be reduced by precomputing some of such distances as theoretically explained in Section 3.2. Accordingly, after testing this improvement, when using the experimental setup described in Section 4.2, we observe a speedup factor between 1.14 and 1.31×. This execution performance is illustrated in Figs. 6, 7, 8 for the three data sets previously mentioned.

Once again, implementing this distance reuse does not reflect any variation neither in the structure of the microcells obtained nor in the distortion imposed to the data sets.

4.3.3. Partial sorting

4.3.3.1. Preliminary test. As described in Section 3.3, the impact of sorting operations on the running time of MDAV could be reduced by using partial sorting, given its lower complexity with respect to total sorting. To illustrate the potential improvements due to partial sorting, we first perform an experiment in Matlab comparing two applications of both problems. Although not explicitly stated in the documentation, it is reasonable to assume that the functions `sort` and `mink` of Matlab R2017b implement variants of quicksort (total sorting) and quickselect (partial sorting) algorithms, respectively, as explained in Section 2. In fact, this experiment confirms that these functions follow the behavior of quicksort and quickselect in terms of computational complexity.

For this initial test, we do not only measure how long `sort` and `mink` take to execute over a list of random generated numerical values, but we try to mimic the sorting operations performed by MDAV through a few simple steps: sorting real-valued numbers and allocating and returning the indices of sorted values. It turns that returning indices noticeably slows down the execution of sorting functions for short lengths. Moreover, when necessary, we preallocate outputs to exclude the time for memory allocation from our measurements. Particularly for the function sort, we also consider the time of trimming off the shortest values from the list.

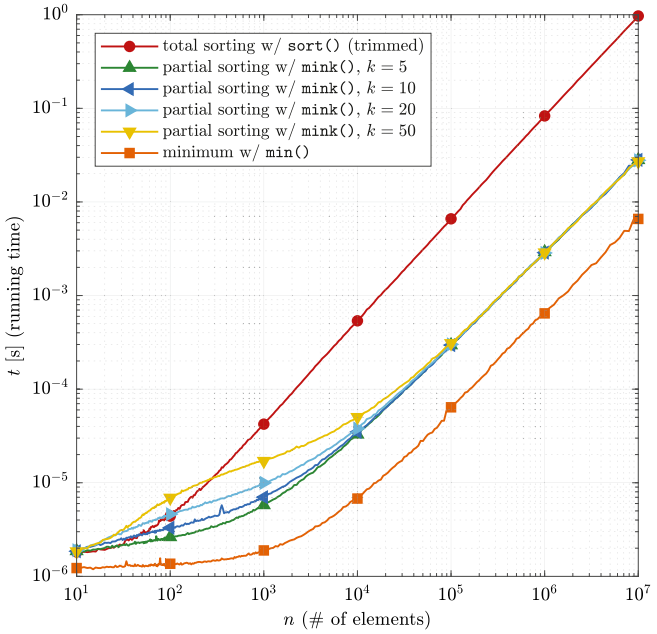


Fig. 9. Running time of different variants of sorting implemented in Matlab R2017b. Extensive testing is performed for several values of n (number of elements in the sorted list) and k (here representing the number of elements to be selected and sorted from the list, when partial sorting is tested). For the sake of clarity, double logarithmic scales are used.

The test involves more than 300 experiments, each of which consists in measuring the time it takes to obtain the k shortest values from a randomly generated list of length n. This process, along with the considerations of the last paragraph, emulate the role of sorting within MDAV. To evaluate the benefits of partial sorting over total sorting, we obtain the running times when using `mink` and `sort` functions to find the shortest values; we test their performance for several values of n, which ranges from 10 to 10 million, and for $k = \{5, 10, 20, 50\}$.

For the sake of reliability, we measure the running times for several repetitions in each experiment. Then, we compute the mean, as an estimate of average performance. Also, while the length of the data used is the same for every experiment, the values of the list are randomly sampled for every repetition. After a one-hour experiment, we found that our measurements were extremely reliable: the worst coefficient of variation, calculated as the standard deviation divided by the mean, was observed to be 1.63%.

Fig. 9 shows how our experiments take longer (the running time t gets higher) as the length n of the list increases. We use double logarithmic scales since we have very wide ranges of values for n and t, and thus extremely low and high values may appear. We can see that the running time for `mink` grows lineally with n, regardless of the value of k used, in line with the complexity $\mathcal{O}(n)$ of the quickselect algorithm; this is important evidence that `mink` would be implementing a variant of this algorithm. For `sort`, the corresponding running times are certainly higher. However, the magnitude of the difference with respect to `mink` is not very clear. For that reason, we depict in Fig. 10 the running time per element $\frac{t}{n}$ for every experiment. Using a semilogarithmic scale, this figure does show that `mink` (partial sorting) is much more efficient than sort (total sorting) since while for `mink` the running time per element is constant, said time grows logarithmically with n for sort.

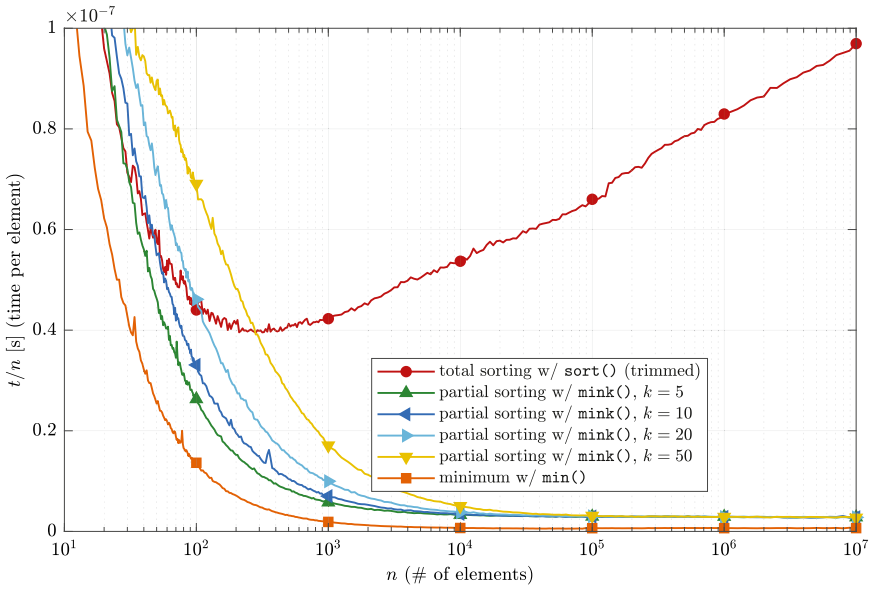


Fig. 10. Running time of different variants of sorting implemented in Matlab R2017b. Here, we depict the time taken per element $\frac{t}{n}$ (t is the time taken to sort a list of n elements) to have a clearer illustration of the remarkable performance of partial sorting implementations compared to those of total sorting. Again k represents the number of elements to be selected and sorted from the list in the case of partial sorting. Briefly, the running time of partial sorting remains constant for large values of n, while for total sorting time grows logarithmically. Also, for clarity, a semilogarithmic scale is used.

As a reference, we also plot the running time of the function *min* that retrieves only the lowest value from each list.

For the sake of verification, we also perform this experiment by using actual quicksort and quickselect algorithms implemented in C++. The tests for this experiment are also run in Matlab, where the corresponding C++ code is called through a gateway function in a MEX file. The results coincide with the ones described in the last paragraph.

4.3.3.2. MDAV with partial sorting. To estimate the speedup of microaggregation due partial sorting, we run the experiments according to the setup proposed in Section 4.2, now using a version of MDAV that relies on the function `mink` for microcell assignment. We then compare the resulting running times with those of traditional MDAV that uses `sort` by default.

As expected, partial sorting introduces interesting computational improvements. In fact, a speedup factor of almost 2× is reached for the Large Census data set when $n = 150,000$, as depicted in Fig. 6. However, the degree of improvement is not uniform for the three data sets, as it is shown in Figs. 7 and 8 for the partial sorting method where the maximum speedup factor does not attain 1.5×. This seems reasonable since the complexity of the variants of sorting may depend on the intrinsic structure of the data.

4.3.4. Centroid by subtraction

It is clear from Section 3.4 that the operations for the calculation of centroids in MDAV are subject to redundancy since the tuples of a microdata set have to be added recurrently to find a representative mean. Our centroid by subtraction strategy, which uses precomputing and subtraction, obtains a speedup factor of up to 1.15× as shown in Figs. 6–8. Although performance gain is more moderate for MDAV, it is still important, considering that its implementation does not represent any additional cost in terms of distortion.

4.3.5. Single precision

Being MDAV an algorithm whose calculations may not require extreme precision, our last method is based on using single precision for the corresponding mathematical operations. Strikingly, this modification allows a computational improvement that is even better than that of centroid by subtraction, i.e., a speedup factor of up to 1.35×, as depicted in Figs. 6, 7, 8, for our three data sets.

As anticipated in Section 3.5, the results from using this strategy show a slight variation in the structure of the microcells built by MDAV. However, the resulting distortion remains unchanged.

4.3.6. Fast MDAV

Our last series of experiments analyze the case when all proposed modifications are combined in a single version of the microaggregation algorithm, fast MDAV. The tests, following the setup in Section 4.2, show remarkable results, as reflected in Fig. 11. We confirm that fast MDAV is up to 4 times faster than the original version and that, as expected, no additional distortion is introduced in the three microaggregated data sets.

4.3.7. Information loss with F-MDAV

As mentioned when describing each computational improvement in Section 3, the rationale behind our work is simplifying redundant operations when implementing MDAV. Such simplification is based on finding alternate algebraic expressions, reusing and precomputing (repetitive) calculations, adapting a more relaxed sorting strategy, and even using less precise calculations.

When single precision is used for calculations, there is a risk of obtaining a structure of microcells different from that of MDAV. However, microaggregation does not require extremely precise calculations since those are used only for comparing distances between points. Namely, in practice, our proposal does not imply any modification of the resulting k-anonymous groups built by MDAV, so there might not be a price in distortion. As a matter of fact, we have verified that microcell allocation remains invariable after F-MDAV is implemented. Consequently, our strategies do not incur in additional distortion or information loss with respect to that provoked by original MDAV.

In times when the world revolves around big data, processing time quickly becomes a bottleneck with respect to the potential applications of large-scale databases. Moreover, domains as critical as health, vehicular traffic, or network intrusion detection are currently using tons of data to help computational systems make real-time, and even life-or-death decisions. Due to such demanding requirements, privacy issues related to data processing are commonly overshadowed. Thus, from the perspective of privacy, we feel that any improvement in (computing) efficiency is not negligible, particularly when the strategy does not entail additional data distortion, and even more when its multiplicative effect may turn a privacy mechanism feasible for a critical application.

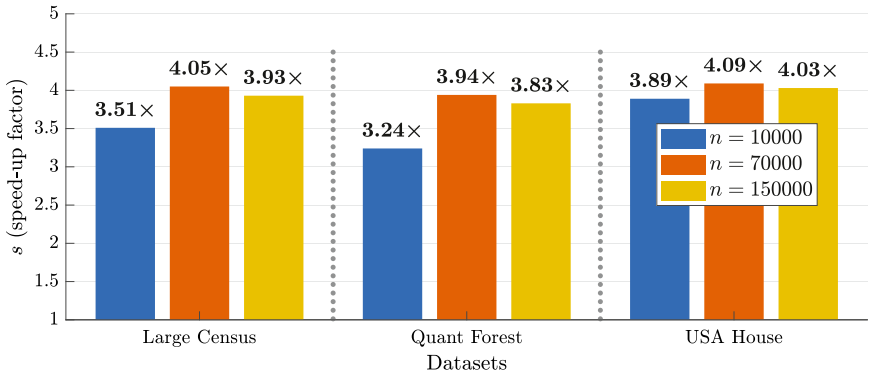


Fig. 11. Overall speedup factor s of our fast MDAV obtained over the three data sets. The five strategies are consolidated in a single version and it is tested for several values of n. Due to space considerations, only the results of tests for $n = 10\,000, 70\,000, 150\,000$ are depicted in this figure.

5. Conclusions and future work

This work addresses the problem of computational complexity of k-anonymous microaggregation for large datasets with a substantial amount of numerical records. Our approach strives to obtain a more usable privacy mechanism in contexts brought by the current big data era. We have proposed five algorithmic and algebraic strategies that significantly reduce the running time of maximum distance to average vector, a widely-known microaggregation algorithm, while leaving untouched the resulting utility of the anonymized data.

Our improvement strategies mainly span the reuse of calculations, precomputing, algebraic modifications, and a relaxed approach of sorting; all of them are implemented in the main tasks performed by the maximum distance to average vector algorithm, e.g., distance calculation, sorting, assignment of microcell, and centroid calculation. As expected, the strategies focused on the most repetitive operations, e.g., distance calculation and sorting, lead to the highest performance, making this algorithm up to twice faster than the original one. However, the specific improvement may vary with the data set used since the performance of some of these operations (particularly sorting) depends on the internal structure of the numerical data.

The results of systematic experimentation in three standard data sets show that the combined effect of our strategies produce a version of maximum distance to average vector whose running time is four times shorter than its original version. The remarkable overall effect in our fast implementation seems to be uniform for the data sets tested and is conveniently attained without incurring in additional distortion on the data. A similar result is observed when testing a synthetic data set. Our concluding message aims at calling attention to put more emphasis on the multidimensional trade off among privacy, utility and usability brought by the emerging context of big data when dealing with old and new approaches of privacy protection. Future directions to further improve the performance of microaggregation may involve, e.g., reducing its inherent complexity by resorting to eliminate its potential statistical redundancy, or applying other lossy techniques that give up some data utility in exchange for usability of the privacy mechanism.

CRediT authorship contribution statement

Ana Rodríguez-Hoyos: Methodology, Validation, Writing - original draft, Writing - review & editing. **José Estrada-Jiménez:** Methodology, Validation, Writing - original draft, Writing - review & editing. **David Rebollo-Monedero:** Conceptualization, Formal Analysis, Writing - review & editing, Validation, Supervision. **Ahmad Mohamad Mezher:** Supervision, Writing - review & editing. **Javier Parra-Arnau:** Supervision, Writing - review & editing. **Jordi Forné:** Supervision, Writing - review & editing.

Acknowledgments

This work was supported by the Escuela Politécnica Nacional, Ecuador through the project “Privacidad Sintáctica Funcional: Análisis y adaptación de mecanismos de anonimato con enfoque en la preservación de utilidad de los datos”, ref. PII-DETRI-2019-01. J. Parra-Arnau is the recipient of a Juan de la Cierva postdoctoral fellowship, IJCI-2016-28239, from the Spanish Ministry of Economy and Competitiveness (MINECO). This work was partly supported by the MINECO through the project “MAGOS”, ref. TEC2017-84197-C4-3-R and through the “Anonymized Demographic Surveys (ADS)” project, ref. TIN2014-58259-JIN, under the funding program “Proyectos de I+D+i para Jóvenes Investigadores”.

References

AOL search data scandal, 2006. URL http://en.wikipedia.org/wiki/AOL_search_data_scandal.

Assoc. Press, 2017. Divided by citizen wealth tables. NY Times URL www.nytimes.com/2009/10/24/business/global/24tax.html?_r=2&ref=global.

Brickell, J., Shmatikov, V., 2008. The cost of privacy: Destruction of data-mining utility in anonymized data publishing. In: Proc. ACM SIGKDD Int. Conf. Knowl. Discov., Data Min. (KDD). Las Vegas, NV.

Calviño, A., 2017. A simple method for limiting disclosure in continuous microdata based on principal component analysis. J. Off. Stat. 33 (1), 15–41.

Defays, D., Nanopoulos, P., 1993. Panels of enterprises and confidentiality: The small aggregates method. In: Proc. Symp. Design, Anal. Longit. Surv., Stat. Can. Ottawa, Canada, pp. 195–204.

Domingo-Ferrer, J., González-Nicolás, Ú., 2010. Hybrid microdata using microaggregation. Inform. Sci. 180 (15), 2834–2844.

Domingo-Ferrer, J., Mateo-Sanz, J.M., 2002. Practical data-oriented microaggregation for statistical disclosure control. IEEE Trans. Knowl. Data Eng. 14 (1), 189–201.

Domingo-Ferrer, J., Sebé, F., Solanas, A., 2008. A polynomial-time approximation to optimal multivariate microaggregation. Comput. Math. Appl. 55 (4), 714–732.

Domingo-Ferrer, J., Torra, V., 2005. Ordinal, continuous and heterogeneous k-anonymity through microaggregation. Data Min. Knowl. Discov. 11 (2), 195–212.

Domingo-Ferrer, J., Torra, V., 2008. A critique of k-anonymity and some of its enhancements. In: Proc. Workshop Priv., Secur., Artif. Intell. (PSAI). Barcelona, Spain, pp. 990–993.

Dwork, C., 2006. Differential privacy. In: Proc. Int. Colloq. Automata, Lang., Program. (ICALP). In: Lect. Notes Comput. Sci. (LNCS). vol. 4052, Venice, Italy, pp. 1–12.

Floyd, R.W., Rivest, R.L., 1975a. Algorithm 489: The algorithm SELECT —For finding the ith smallest of n elements. Commun. ACM 18 (3), 173.

Floyd, R.W., Rivest, R.L., 1975b. Expected time bounds for selection. Commun. ACM 18 (3), 165–172.

Hoare, C.A.R., 1961a. Algorithm 64: Quicksort. Commun. ACM 4 (7), 321.

Hoare, C.A.R., 1961b. Algorithm 65: Find. Commun. ACM 4 (7), 321–322.

Hundepool, A., de Wetering, A.V., Ramaswamy, R., Franconi, L., Capobianchi, A., de Wolf, P.-P., Domingo-Ferrer, J., Torra, V., Brand, R., Giessing, S., 2003. μ-ARGUS version 3.2 software and user’s manual. Stat. Neth., Voorburg, Netherlands, URL http://neon.vb.cbs.nl/casc.

Kiwiel, K.C., 2005. On floyd and rivest’s SELECT algorithm. Theoret. Comput. Sci. 347 (1–2), 214–238.

Laszlo, M., Mukherjee, S., 2005. Minimum spanning tree partitioning algorithm for microaggregation. IEEE Trans. Knowl. Data Eng. 17 (7), 902–911.

Li, N., Li, T., Venkatasubramanian, S., 2007. *t*-Closeness: Privacy beyond *k*-anonymity and *l*-diversity. In: Proc. IEEE Int. Conf. Data Eng. (ICDE). Istanbul, Turkey, pp. 106–115.

Li, D., Shen, X., Wang, L., 2017. Connected geomatics in the big data era. Int. J. Digit. Earth 1–15.

Lin, J.L., Wen, T.H., Hsieh, J.C., Chang, P.C., 2010. Density-based microaggregation for statistical disclosure control. Expert Syst. Appl. 37 (4), 3256–3263.

Matatov, N., Rokach, L., Maimon, O., 2010. Privacy-preserving data mining: A feature set partitioning approach. Inform. Sci. 180 (14), 2696–2720.

Mohamad Mezher, A., García-Álvarez, A., Rebollo-Monedero, D., Forné, J., 2017. Computational improvements in parallelized *k*-anonymous microaggregation of large databases. In: Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS), Workshop Priv., Secur. Big Data (PSBD). Atlanta, GA, pp. 258–264.

Monaco, A., 2016. Big data: The measure of humankind. Times High Educ. (THE) URL www.timeshighereducation.com/comment/big-data-measure-humankind#survey-answer.

Mortazavi, R., Jalili, S., 2017. Fine granular proximity breach prevention during numerical data anonymization. Trans. Data Priv. 117–144.

Mortazavi, R., Jalili, S., Gohargazi, H., 2014. Fast data-oriented microaggregation algorithm for large numerical datasets. Knowl.-Based Syst. 67 (3), 195–.205.

Musser, D., 1999. Introspective sorting and selection algorithms. J. Softw.: Pract. Exp. 27 (8), 983–993.

Narayanan, A., Shmatikov, V., 2008. Robust de-anonymization of large sparse datasets. In Proc. IEEE Symp. Secur., Priv. (S&P). Oakland, CA, pp. 111–125.

Pallarès, E., Rebollo-Monedero, D., Rodríguez-Hoyos, A., Estrada-Jiménez, J., Mezher, A.M., Forné, J., 2019. Mathematically optimized, recursive prepartitioning strategies for k-anonymous microaggregation of large-scale datasets. Expert Syst. Appl. 113086.

Parra-Arnau, J., Domingo-Ferrer, J., Soria-Comas, J., 2020. Differentially private data publishing via cross-moment microaggregation, 53 (2), 269–288.

Raghupathi, W., Raghupathi, V., 2014. Big data analytics in healthcare: Promise and potential. Health Inform. Sci. Syst. 2.

Rebollo-Monedero, D., Forné, J., Domingo-Ferrer, J., 2008. From *t*-closeness to PRAM and noise addition via information theory. In: Proc. Int. Conf. Priv. Stat. Databases (PSD). In: Lect. Notes Comput. Sci. (LNCS). Istambul, Turkey, pp. 100–112.

Rebollo-Monedero, D., Forné, J., Domingo-Ferrer, J., 2010. From *t*-closeness-like privacy to postrandomization via information theory. IEEE Trans. Knowl. Data Eng. 22 (11), 1623–1636, URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2009.190.

Rebollo-Monedero, D., Forné, J., Pallarès, E., Parra-Arnau, J., 2013a. A modification of the Lloyd algorithm for *k*-anonymous quantization. Inform. Sci. 222, 185–202. http://dx.doi.org/10.1016/j.ins.2012.08.022.

Rebollo-Monedero, D., Forné, J., Soriano, M., 2011. An algorithm for *k*-anonymous microaggregation and clustering inspired by the design of distortion-optimized quantizers. Data, Knowl. Eng. 70 (10), 892–921. http://dx.doi.org/10.1016/j.datak.2011.06.005.

Rebollo-Monedero, D., Forné, J., Soriano, M., Puiggalí Allepuz, J., 2017. *p*-PRobabilistic *k*-anonymous microaggregation for the anonymization of surveys with uncertain participation. Inform. Sci. 382–383, 388–414. http://dx.doi.org/10.1016/j.ins.2016.12.002.

Rebollo-Monedero, D., Mezher, A.M., Casanova Colomé, X., Forné, J., Soriano, M., 2019a. Efficient k-anonymous microaggregation of multivariate numerical data via principal component analysis. Inform. Sci. 417–443.

Rebollo-Monedero, D., Mohamad Mezher, A., Casanova Colomé, X., Forné, J., Soriano, M., 2019b. Efficient *k*-anonymous microaggregation of multivariate numerical data via principal component analysis. Inform. Sci. 503, 417–443.

Rebollo-Monedero, D., Parra-Arnau, J., Díaz, C., Forné, J., 2013b. On the measurement of privacy as an attacker's estimation error. Int. J. Inform. Secur. 12 (2), 129–149. http://dx.doi.org/10.1007/s10207-012-0182-5.

Rodríguez-Hoyos, A., Estrada-Jiménez, J., Rebollo-Monedero, D., Parra-Arnau, J., Forné, J., 2018. Does *k*-anonymous microaggregation affect machine-learned macrotrends?. IEEE Access 6, 28258–28277. http://dx.doi.org/10.1109/ACCESS.2018.2834858.

Samarati, P., 2001. Protecting respondents' identities in microdata release. IEEE Trans. Knowl. Data Eng. 13 (6), 1010–1027.

Sankar, L., Rajagopalan, S.R., Poor, H.V., 2013. Utility-privacy tradeoffs in databases: An information-theoretic approach. IEEE Trans. Inform. Forensics Secur. 8 (6), 838–852.

Solé, M., Muntés-Mulero, V., Nin, J., 2012. Efficient microaggregation techniques for large numerical data volumes data volumes. Int. J. Inform. Secur. 11, 253–267.

Sun, X., Wang, H., Li, J., Truta, T.M., 2008. Enhanced *p*-sensitive *k*-anonymity models for privacy preserving data publishing. Trans. Data Priv. 1 (2), 53–66.

Sun, X., Wang, H., Li, J., Zhang, Y., 2012. An approximate microaggregation approach for microdata protection, 39 (2), 2211–2219.

Sweeney, L., 2000. Uniqueness of simple demographics in the U.S. population. Tech. Rep. LIDAP-WP4, (LIDAP-WP4), Carnegie Mellon Univ., Sch. Comput. Sci., Data Priv. Lab., Pittsburgh, PA.

Templ, M., 2014. Introduction to Statistical Disclosure Control (SDC). Res. rep., URL https://pdfs.semanticscholar.org/311d/73e8e4b5b1dca49901d929f383bbd0817a7c.pdf.

Truta, T.M., Vinay, B., 2006. Privacy protection: *p*-Sensitive *k*-anonymity property. In: Proc. Int. Workshop Priv. Data Mgmt. (PDM). Atlanta, GA, p. 94.

Williams, J., 1964. Algorithm 232: Heapsort. Commun. ACM 7 (6), 347–348.