Privacy-Aware Multidimensional Indexing

Alexander Grebhahn[1], Martin Schäler[2], Veit Köppen[2], Gunter Saake[2]

[1] Brandenburg University of Applied Sciences
P.O. Box 2132, 14737 Brandenburg, Germany
grebhahn@fh-brandenburg.de

[2] Department of Technical and Business Information Systems
Otto-von-Guericke University Magdeburg
P.O. Box 4120, 39016 Magdeburg, Germany
{schaeler, vkoeppen, saake}@iti.cs.uni-magdeburg.de

Abstract: Deleting data from a database system in a forensic secure environment and in a high performant way is a complex challenge. Due to redundant copies and additional information stored about data items, it is not appropriate to delete only data items themselves. Additional challenges arise when using multidimensional index structures. This is because information of data items are used to index the space. As initial result, we present different deletion levels, to overcome this challenge. Based on this classification, we analyze how data can be reconstructed from the index and modify index structures to improve privacy of data items. Second, we benchmark our index structure modifications and quantify our modifications. Our results indicate that forensic secure deletion is possible with modification of multidimensional index structures having only a small impact on computational performance, in some cases.

1 Introduction

With an increasing usage of computer-aided systems, more sensitive information is stored in electronic formats. This may cause problems with respect to privacy. Laws and guidelines are created, to improve privacy, such as the Health Insurance Portability and Accountability Act (HIPAA) [Con96] in the USA, the Hard Drive Secure Information Removal and Destruction Guidelines [Roy03] in Canada, or the Bundesdatenschutzgesetz [Bun09] in Germany. According to these laws, private information has to be deleted or encrypted in such a way, that it cannot be reconstructed after deletion (forensic secure deletion). This is a complex challenge. Due to the fact that we need to remove *every* existing copy and *every* effect caused by the data item that we want to delete.

In our research project[1], we focus on private information, such as extracted features from fingerprints or micro traces. These features are multi- (less than 20 dimensions) or highdimensional data (more than 20 dimensions) containing, for instance, three-dimensional coordinates and classification attributes[2].

[1] https://omen.cs.uni-magdeburg.de/digi-dak-plus/
[2] Classification into multi- or highdimensional data according to [GG98].

In contrast to cloud computing models, where data and queries are outsourced to the cloud [HXRC11] and a privacy preserving storage (an encrypted storage) of data is necessary, we focus on systems, storing data in an unencrypted way. Additionally, to have a secure data life cycle, it is not only necessary to encrypt data, but also to delete data in a forensic secure manner [DW10].

Due to the huge amount of data, it is necessary to store data in systems that handle it in an appropriate way. As a result, we use multi- and highdimensional index structures to speed-up query response times. Within this paper, we make two major contributions:

1. We analyze, how we can reconstruct sensitive data from well-known index structures under certain assumptions (deletion strategies) and provide, to the best of our knowledge, the first empirical study regarding this topic.
2. We recommend improvements for forensic secure deletion for some of these indexes and evaluate their benefits and drawbacks exemplarily w.r.t. reduced reconstruction ability of data and run-time overhead. Based on our results, we define different levels of forensic secure deletion w.r.t. necessary reconstruction effort.

The remainder of the paper is organized as follows: In Section 2, we give an overview of related work and motivate our index structure selection. We also present background on functionality of analyzed index structures. In Section 3, we present four different strategies for deleting data from a database and their respective implementations for our index selection. In Section 4, we analyze these index structures w.r.t. stored information and how data can be reconstructed. Additionally, we present ideas how to modify index structures to minimize possibilities to reconstruct data. In Section 5, we evaluate index structures and index structure modifications w.r.t. performance and precision for an approximative index structure. Additionally, we show, how much information about data can be reconstructed from an index after deleting data. We draw a conclusion and present future work in Section 6.

2 Background

In this section, we briefly summarize the state of the art in database forensics, introduce terms, and provide necessary background on our selected index structures.

2.1 Related Work on Forensics in Databases.

To delete a data item from a database system forensically secure, it is necessary to consider more than the tuple stored in the table space [SML07]. This is because additional information about data items are stored by a database system. Beside the database (here, we mean the files containing the tuples), there are two other storage components communicating with the database system and an additional storage component for reconstructing the database when a media or system error occurs [BHG87]. A storage component communicating with the database system is the data dictionary. This component stores, information is stored e.g. histograms on data distribution or table schemas information. Third, the database log is stored by the database system. In this log, all information needed to recreate a consistent state of the database is stored. Stahlberg et al. give an

overview on challenges that have to be considered, in case of forensic secure deletion in database systems [SML07]. In detail, they cover information stored in database, in indexes, and in database logs exemplarily for InnoDB, a MySQL storage engine. As a representative of indexes, they cover a forensic secure deletion of data items from a B-Tree [SML07].

Within other work, see for example [Lit07, Fow08, FHMW10, Gre12], database systems, like Oracle, SQL Server, MySQL (using InnoDB as storage engine), PostgreSQL, and HSQLDB, are examined according to recomputation of data items from information left in the database files.

Besides deleting data items from a system in a forensic secure way, encrypting data is another solution for privacy preserving data management in highdimensional spaces. This is a commonly used technique for supporting privacy for outsourced data management as used within the cloud [HXRC11]. Many solutions are presented within the last years, see for example [KS07, HMCK12, WCKM09]. However, encryption is out of the scope of this paper.

2.1.1 Challenges in Definition of Forensic Secure Deletion.

To define, under which circumstances a data item is forensically secure deleted, we use the following intuitive definition for total forensic secure deletion:

A data item is deleted total forensic secure*, if absolutely no conclusions on exact or approximate values of any of the attributes of the data item can be drawn by using information stored in the system.*

However, due to non-trivial interdependencies (e.g., materialized aggregates) and non-obvious remains (e.g., in swap files or backups) using this definition is problematic as it is hardly reachable in practice. We further argue that, depending on data sensitivity, total forensic secure deletion is not always necessary. In the same sense, current laws state that the hurdles to access deleted data have be in an adequate relation to the value of the data (e.g., § 20(3) [Bun09]). Consequently, we define different levels of forensic secure deletion in Section 3. Before, we present an analysis of different deletion strategies.

2.2 Index structure selection

We give a brief overview of selected index structures in the following. For further information, see for example [GG98, Sam05]. In this paper, we: (1) show differences in data reconstruction derived from information stored in the index, (2) address different classes of index structures to generalize our results and address comprehensiveness, and (3) select well-known index structures.

For classification of index structures, we refer to existing classifications (see for example [GG98] and [WSB98]). They classify in data versus space organizing and exact versus approximative indexes. Consequently, we consider at least one data and one space organizing index as well as one exact and one approximative index. In detail, we focus on the R-Tree [Gut84] and respective extensions (e.g., [BKSS90, SRF87]), the VA-File [WB97], and the Prototype Based Approach (PBA) [CGFN08]. Note, due to the overlap of the classification attributes, we only use three index structures. We summarize our index selection in Table 1.

Table 1: Index structure selection.

Name	Data/Space Org.	Exact/Approx.	Remarks
R-Tree	Data	Exact	Tree-based
VA-File	Space	Exact	Improved sequential scan
PBA	Space	Approximative	Hash based.

2.2.1 R-Tree and its variants

One of the most popular multidimensional index structures is the R-tree. This index structure is presented by Guttman [Gut84]. Many improvements like R^+-Tree [SRF87], R^*-tree [BKSS90], X-Tree [BKK96], and SS-Tree [WJ96] are based on the ideas of the R-Tree. This basic idea is to partition the data space by the use of minimal bounding rectangles (MBR).

MBRs are organized in a hierarchical way as shown in Figure 1. The root MBR, namely R1 in Figure 1, includes the minimal space needed to include all child MBRs (R2 and R3). These MBRs again include the space of their child MBRs. This organization holds up to the leaf MBRs. Within these leafs (marked in gray in Figure 1), data items are stored.

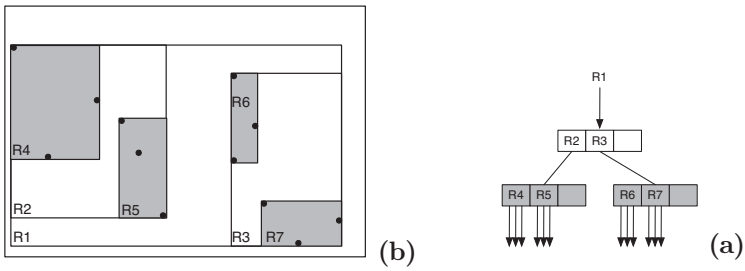


Figure 1: (a) Partitioning of a two-dimensional space by an R-Tree. (b) Hierarchical structure of the MBRs.

For each MBR of an R-Tree, two points are stored. These two points are both ends of one diagonal of the MBR. Additionally to these two points, each MBR contains pointers to all child MBRs for inner nodes and pointers to all data items indexed by the MBR for leaf nodes. According to the idea of Guttman, the storage size of a node should be correlated with the page size of the underlying system. For modeling such a correlation, the maximum and minimum number of data items per page can be defined by parameter m. The maximum number of data items is implicitly given by M, with $M = 2 \cdot m$. So, for every node of an R-Tree two points are stored together with pointers to child nodes for inner MBR or pointers to data items for leaf nodes.

2.2.2 VA-File

The VA-File is a space organizing index proposed by Weber and Blott [WB97]. The main idea of this index structure is to store a small representation of original data that fits more likely into the main memory. This representation addresses rectangular cells in form

of bit vectors used to filter and thus, to reduce the amount of points that are retrieved from hard disk.

As shown in Figure 2 (a), there are four distinct regions in each dimension when choosing a vector length of two for each dimension. This leads to 2^b regions, in the case of choosing a vector length of b. In general, in a d-dimensional space, the space is divided into 2^{bd} hyper rectangles. The formal allocation of a point to a region, by Weber and Blott [WB97], is stated in Equation 1, where $r_{i,j}$ defines the partition p_i is located in dimension j. Furthermore, m_i states the lower bound of the ith-partition. According to this definition, the bounds of partitions are defined by values of points.

$$m_i[r_{i,j}] \leq p_{i,j} < m_i[r_{i,j+1}] \tag{1}$$

For being adaptive to different data distributions, the regions width depends on the data distribution. We present an example in Figure 2 (a). Here, the width of region *10* of dimension x is larger than the width of region *00*. As a result, of this unequal distinction of the space, within the index structure a map has to be stored to describe the mapping between the original space and the resulting approximation vector. So, the VA-File stores an approximation vector for each data item and the mapping from the original space to the approximated one.

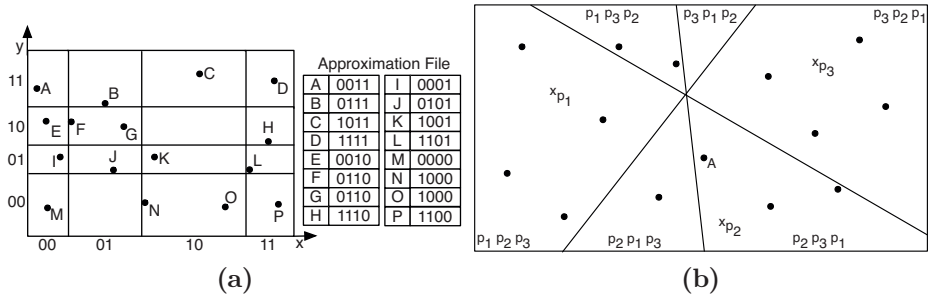


Figure 2: (a) Structure of a VA-File. (b) PBA using three prototypes.

2.2.3 Prototype Based Approach

An example for an approximation-based index structure is the Prototype Based Approach (PBA) presented by Chavez et al. as Ordering Permutations [CGFN08]. The basic idea is to use some points from a dataset to index the whole dataset. According to Chavez et al., we call these points *prototypes*. In general, this technique divides the whole space in convex regions based on distances to the prototypes. In Figure 2 (b), we show a space partition with three prototypes p_1,p_2, and p_3. When inserting a point, distances to all prototypes are computed. Next, the prototypes are ordered ascending according to their distances. Finally, this ordering is used as hash value or key of the point. For example, the value of point A in Figure 2 (b) is (p_2, p_3, p_1). As a result, the hash value of each data item and the coordinates of all prototypes are stored in the index.

3 Deletion Strategies and Respective Deletion Levels

In this section, we present and discuss different strategies for deleting data items from an index structure. Furthermore, based on the remaining possibilities to reconstruct data, we use these strategies to describe different levels of forensic secure deletion. Additional, to presented deletion levels, it is possible, to rebuild an index after every deletion.

3.1 Level 0: Delete Bits

The first strategy is using a delete bit for identifying, whether a data item is deleted or not. In other words, when data have to be deleted, not the whole item is modified, but a bit within the header of the data item for marking it as deleted. This strategy has disadvantages w.r.t. privacy of deleted data items. With the help of simple tools and knowledge of the structure of the database, it is possible to identify deleted data items and to reconstruct them completely [Lit07, Fow08]. A prototype of such a forensic tool for PostgreSQL is given in [Gre12].

In summary, using a delete bit or similar technique to mask deleted data items allows to easily reconstruct data items in total (with all attributes) with basic knowledge of the way how items (tuples) are stored. Note, this, is no deletion at all, and therefore we call it Level 0 (cf. Table 2), indicating that a data item is *not* forensically secure deleted. However, modifying the delete bit is very time efficient and requires no reorganization of index structures. Hence, this deletion strategy is very time efficient.

Integration in index structures. Integrating this strategy into known index structures is rather simple. Here, no reorganization of parts of indexes (e.g., MBRs within an R-Tree) is necessary, if an item is deleted. An additional challenge arises in frequently changing tables through the constantly increasing size of the index.

3.2 Level 1: Overwriting without reorganization

A next level strategy is deleting (and overwriting) the whole data item without modifying the index structure. Although the data item is removed and overwritten, it is possible to reconstruct (parts of) the deleted data item. This is due to remaining information (e.g., structure of the index) that can be used for an attempt to reconstruct the data. To sum up, reconstruction of data (a) is more laborious and (b) is not possible in all cases, and (c) requires more detailed knowledge on the way index structures store their data. Consequently, this deletion strategy forms forensic secure deletion Level 1, and thus, the first level that offers *basic* forensic deletion capabilities.

In contrast to Level 0 deletion strategy, we hypothesize that the amount of information that can be reconstructed, depends on the definition of the index structure and therefore, it is purpose of our analysis and experiments in the next sections. Using this deletion strategy, there are index-specific cases that still allow either (1) total reconstruction of a data-item, (2) reconstruction of some *attributes* with exact values, or (3) we can state upper and lower bounds of attribute values. First and more detailed considerations to determine probability of single

index-specific cases and an analysis for respective causes are also part of the next sections.

Integration in index structures. By using this strategy it is not necessary to consider, for example, underfull MBRs in an R-Tree. Furthermore, we do not have to recompute the partitioning of the VA-File or the PBA. However, since we have to overwrite possibly large datasets, the effort for this deletion strategy is higher than for Level 0.

3.3 Level 2: Overwriting with reorganization

To address remaining threat of reconstructable data, we introduce another level that offers *advanced* forensic secure deletion capabilities (Level 2). The goal of this level: it is *practically* impossible to reconstruct data items deleted from an index structure. The main reasons why it is possible to reconstruct data using Level 1 are remaining, index-specific traces due to missing reorganization of the index. Thus, the additional effort for reaching this level is reorganization of indexes as we describe in the next section.

Integration in index structures. The integration of this deletion strategy, within a system supporting multi-user, may cause some performance problems, because of concurrent operations on the index. Furthermore, the index reorganization strategy depends on its conceptual design.

3.4 Hypothetic Level ∞: Total forensic secure deletion in data-intensive systems

Although there are no (known) remaining traces in an index, there may be information that can be used to reconstruct data items, such as dependencies in the data (e.g., materialization of aggregates), or hidden copies (e.g., swap files, backups) that need to be considered too. To define the scope and limitation of database forensics, we therefore define a *hypothetic* deletion level that allows *no* reconstruction at all. This level is not defined for indexes only, but it is valid for data-intensive systems.

The basic idea is to have two systems. The first one is the original system (S_{org}) and the second one (S_{shadow}) a (bit-wise) copy[3] of S_{org}, which we denote by: $S_{org} \cong S_{shadow}$. Until the (initial) insertion of data item (d) that we want to delete, both systems behave the same way. That means, they store the same data, swap data from main memory to disk etc. The difference between both systems is that S_{shadow} ignores the insertion of d. After insertion of d, these systems perform again the same read and write operations. Under these circumstances, we consider a function f as total forensic secure deletion w.r.t. d iff $f(S_{org}) \cong S_{shadow}$ holds.

Since we are aware that building these shadow systems is probably practical impossible, we want to create systems that are approximations (S'_{shadow}) of S_{shadow}, where we know simplifications and thus, limit possible effects, we do not consider (e.g., swap files). This shall help to identify non trivial remains of datasets, which are part of future work.

In Table 2, we subsume our four levels of forensic secure deletion. Between this four levels, other level can be defined.

[3]This includes bit-wise copy of all HDDs, main memory, caches, and even CPU registers.

Table 2: Levels of forensic secure deletion.

Level	Technique	Application recommendation	Reconstruction effort	Runtime overhead
0	Delete Bit	No private data.	Low	Low
1	Overwrite	Private data	Medium	Medium
2	Reorganization	Sensitive data	High	High
∞	Shadow image	-	∞	∞

4 Problems with respect to privacy and improvements

In this section, we present privacy problems by information stored in index structures. Additionally, we show modifications for improving privacy of stored information. Within these modifications, we try to reach similar results w.r.t. privacy of *advanced* forensic secure deletion (Level 2) of data items even without reorganization of indexes (as in Level 1).

4.1 R-Tree

With the help of the structure of an R-Tree, conclusions on data distribution as well as single values of data items can be drawn. Firstly, the root node can be used to exclude non covered data space. This is because an R-Tree is a data partitioning method. As a result, it only indexes the space needed. Secondly, because of maximum number of points per MBR, within dense covered regions, more MBRs exist as in sparse covered regions. Thirdly, two points are stored within each MBR for defining size and location. Because an MBR covers the minimal space needed, exact values of points are used to define borders and edges of an MBR. Due to the fact that all data items are stored in leaf nodes, only these nodes have to be analyzed to reconstruct data item specific values.

For improving privacy of single data items within an R-Tree, it is possible to bounce the borders of the MBR away from the location of points dedicated to that node. This increases the overlapping of MBRs within an R-Tree. However, no exact values of single data items are used for defining the corners of the MBR. In bouncing the borders of the MBRs, one has to be aware of R-Tree properties. For example, a parent MBR covers at minimum the whole space covered by its child MBRs. As a result, when bouncing the border of a leaf MBR, all borders of all parent MBRs, sharing a border with it, have to be updated as well.

4.2 VA-File

By considering the information stored in a VA-File, three types of conclusions about the dataset or specific data items can be drawn. Firstly, because of adaptable division of the space, conclusions on the data distribution can be drawn, because all buckets have approximately the same amount of data items dedicated to them. As a result, if a bucket is larger

than a different one, it covers dense populated space. Secondly, the exact values of 2^b-1 data items are stored within the VA-File (see Equation 1). Having a d-dimensional space, this leads to $(2^b-1)^d$ exact values. Thirdly, by using the bit-vector, the approximate location of a data item can be reconstructed. Although, the approximate location of a data item may not lead to privacy problems, in some cases it is possible that the width of a bucket may not only reveal the approximate location but the exact one. For example, if $\frac{1}{2^b}$ data items have the same value in one dimension, the width of the bucket, the points are dedicated to, equals one.

For improving privacy of data items, we modify the VA-File in two different ways. Firstly, we adapt the VA-File in such a way that all buckets have the same width. This leads to some performance penalties when performing queries over none uniformly distributed data. Nevertheless, this partitioning of the data space has advantages for privacy of data items. Because, no information about the data distribution or single data items can be reconstructed from the information stored in the VA-File modification. Additionally, no data item specific information, like exact values of data items within some dimensions are stored. It may happen that a data item is located at the border of a bucket, but the location of the border is not defined by the data item and so not dependent from the data. Additionally, we extend the VA-File in a way that the length of single bit strings per dimension depends on the value domain of this dimension. In other words, we shorten the used bit string for each dimension until the number of regions per dimension is smaller than the used value domain of this dimension.

4.3 Prototype Based Approach

There are some possibilities to improve precision, performance, and privacy of the PBA. Choosing prototypes from the dataset is good for adapting the partitioning of the space to the distribution of the dataset. However, choosing prototypes in a random way leads to some negative effects because some data items may have a greater expressiveness for the distribution of the dataset than others. Additionally, choosing points from the dataset as prototypes leads to privacy problems if the prototype is used after deleting the data. Or it leads to performance problems, because permutations of all points have to be recomputed after a prototype is deleted.

Some modifications at prototype selection and respective position of the prototypes can be implemented, w.r.t. privacy. On the one hand, it is not necessary to choose points from the dataset, but points representing the distribution of the dataset in an optimal way. On the other hand, location of prototypes can be optimized w.r.t. different criteria. For example, it is possible, to choose prototypes, that all regions have the same size. This leads to some performance penalties in performing queries on non-uniform distributed datasets. However, this optimization criterion is good w.r.t. privacy, because the division of the space does not depend on data distribution.

5 Evaluation

In this section, we present a first empirical study on possibilities of reconstructing data items from information stored in multidimensional index structures. Later, we measure the performance penalties introduced by our modifications to improve privacy. In our evaluation, we use the framework QuEval[4]. With this framework, it is possible to measure performance of multidimensional index structures for specific use cases. The idea of this framework and the general structure is proposed in [GBS+12]. Due to the extensibility of the framework, it is possible to extend index structures and the framework with evaluation experiments.

5.1 Datasets

We perform all tests with three different real datasets. In Table 3, we give an overview of dataset properties. The first dataset has only a small number of dimensions (16). As a result, it is multidimensional according to [GG98]. In contrast, the remaining datasets are highdimensional, having 43 and 50 dimensions. With these two datasets, we evaluate the performance impact of the data space population, both having approximate same number of dimensions but different number of points and different value domains.

In detail, the first dataset is a freely available dataset based on extracted hand-writing features [AA96]. In the second dataset (*fingerprint features*), the spectral texture features of latent fingerprints are stored [KFV11]. The last dataset (*particle identification*) is again freely available. Within this dataset, 50 particle identification numbers are stored for 130,064 events [RYZ+05].

Table 3: Properties of datasets used for the evaluation.

domain	#dimension	#points	value domain
Hand-writing features	16	10,992	[0..100]
Fingerprint features	43	411,961	[0..255]
Particle identification	50	130,064	[0..1023]

5.2 Index structure evaluation

In Table 4, we give an overview of our evaluated index structures, modifications we implemented for improving privacy, and evaluations we performed. Due to space limitations, we are not able to present all possible evaluations. For a first insights into the problematic of privacy in multi and highdimensional index structures, we performed an evaluation of the reconstruction rate, modified index structures and evaluate performance and precision of the modifications. In detail, in Section 5.2.1, we evaluate the reconstruction rate (RR) of

[4] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/iJudge/index_en.php

deleted data items stored in an R-Tree when performing deletion Level 1. In Section 5.2.2, we present performance of two different VA-File modifications for improving privacy. Finally, in Section 5.2.3, we present precision differences when we do not choose points from the dataset as prototypes, but points in their neighborhood.

Table 4: Evaluated index structures with performed evaluations.

index structure	modifications	target
R-Tree	Original	RR
VA-File	Original according to [WB97], commensurate regions, Adaptive bit vector length	Performance
PBA	Original, blur location of the prototypes	Precision

5.2.1 Reconstruction rate of deleted data items in an R-Tree

We define the reconstruction rate (RR) of a deleted data item as given in Equation 2. $rDim$ is defined as the number of those dimensions where the exact value of data items can be reconstructed and $allDim$ is the number of all dimensions. In this evaluation, we use some ideas presented in [Lin12].

$$RR = \frac{rDim * 100\%}{allDim} \tag{2}$$

Within our evaluation, we use the Level 1 deletion strategy. In detail, we delete the whole data item, but we do not modify the borders of MBRs. In Figure 3, we show the average RR as well as the maximum RR (dotted) of 10,000 deleted data items for all three datasets used in our evaluation. Additionally, we evaluate RR with different numbers of minimal and maximal points per MBR. Here, we vary m (minimal number of points per MBR) from 2 to 12. Note, maximum number is always two times the minimal number.

In all parts of Figure 3, our results indicate that the average RR of data items decreases with increasing minimum and maximum number of points per MBR. This is, because more points within an MBR decreases the possibility that one point defines a large number of borders. In addition, by comparing the average RR lines of Figure 3 (a), (b) and (c), we draw the conclusion, that the average RR decreases for a given minimum and maximum number of points per MBR with increasing number of dimensions. Additionally, the differences of average RR from (a) to (b) is larger than from (b) to (c). This is, due to the fact the differences of dimensionality between (a) and (b) is greater than between (b) and (c). As a result, we state the hypothesis, that dimensionality has an impact on the RR of data items. Beside this, for every test case, at least one data item can be reconstructed to probability of at least 60%. In detail, within the 50 dimensional space, for every case tested, at minimum one data item can be completely reconstructed.

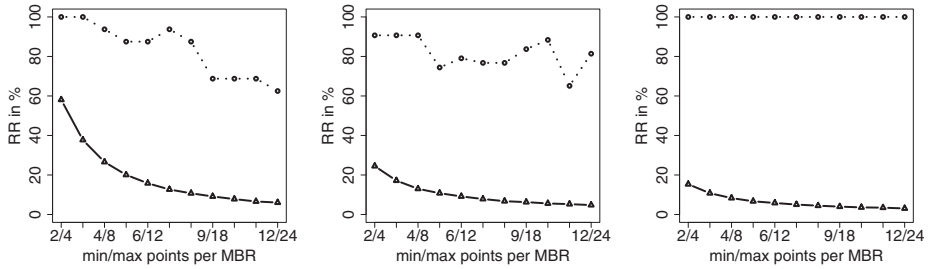


Figure 3: Reconstruction rate (RR) of deleted data items from the information stored in an R-Tree for a 16 (a), 43 (b) and 50 (c) dimensional dataset. The average RR is marked with a solid line and the maximum RR with a dotted line.

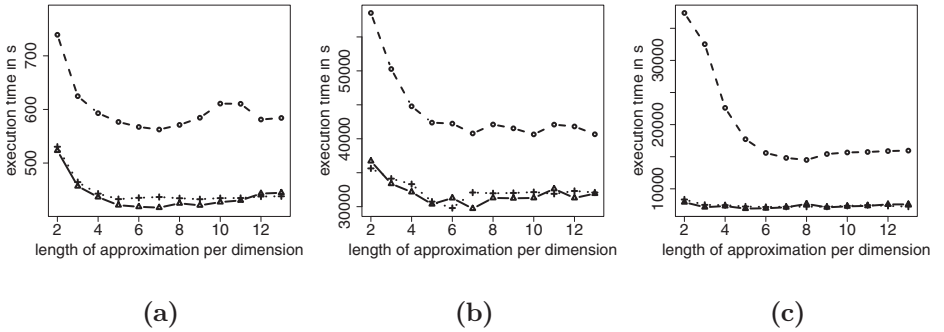
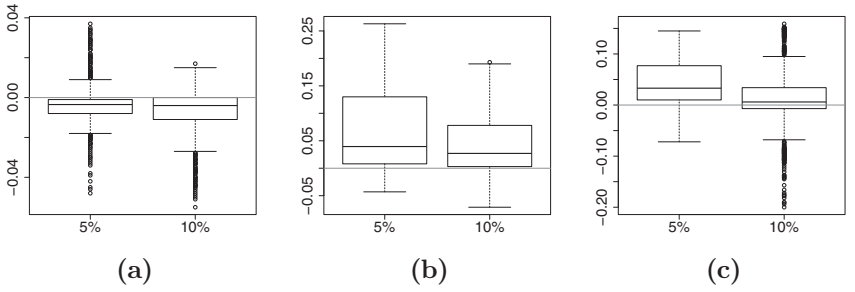


Figure 4: Performance differences of the three VA-File variants for 16 (a), 43 (b) and 50 (c) dimensional dataset. The performance of the original VA-File is marked with a solid line, the performance of the VA-File variant which is not adaptable to the distribution with a dashed line and performance of the variant with an adaptive bit-vector length with a dotted line.

5.2.2 VA-File

In Figure 4, we show the performance of the three different VA-File variants; namely the original VA-File as presented by Weber and Blott [WB97] (solid), the VA-File variant which is not adaptable to the distribution of the dataset (dashed line), and the variant with an adaptive bit-vector length within different dimensions (dotted line). In our experiments, we vary the length of the bit-vector in a range of 2 to 12.

Our results clearly show (cf. Figure 4) that the performance of the VA-File variant which is not adaptable to the distribution of the space is worse than both other VA-File variants. In our experiments, data items are stored on disk and without being adaptive to the data distribution, more points have to be accessed from it.



(a) **(b)** **(c)**

Figure 5: Precision differences between the original Prototypes Based Approach as presented in [CGFN08] to the index structure with blurred prototypes. Differences for the 16 (a), 43 (b) and 50 (c) dimensional dataset with a blur of 5% and 10%.

5.2.3 Prototype Based Approach

Choosing prototypes in a random way leads to poor results regarding to performance and precision. Additionally, in the case of deleting points from the dataset chosen as prototypes, it is necessary to choose new prototypes and to recompute the permutation of all indexed data items. Because of the permutation and the concrete values of all items from the dataset, the location of the prototypes can be recomputed. To overcome this, we modify locations of prototypes with a vector having normal distributed components between zero and given strength (in our examples 5% and 10% of the value domain). For not being affected from one parameter configuration, we performed about 1700 tests with two different blur factors and different parameter configurations for number of prototypes and considered points.

In Figure 5, we show the average difference of precision of the PBA for all three datasets. Within Figure 5, the precision differences for a blurring of 5% and 10% is given, for all three dataset. Blurring the location of prototypes has either a positive or a negative impact on the precision of the index structure depending on the dataset and index parameters. In detail, for our experiments with the 16 dimensional dataset, blurring has mainly a negative impact on precision. However, the average difference of precision is smaller than 0.02% and so, almost negligible. For our other two experiments, blurring the precision has a positive impact, but again the average difference is smaller than 0.1%. All in all, the impact on precision, when choosing random points near to dataset points instead of dataset points as prototypes is almost negligible. As a result, it is not necessary to choose points from the dataset as prototypes.

6 Conclusion & Future Work

To summarize, within this paper, we present four different deletion strategies that can be used within a database system. Additionally, we define forensic secure deletion of information from a database system and present a classification of different secure deletion levels. Furthermore, we examine three different multidimensional index structures (namely R-Tree, VA-File and PBA) in regard how information is stored and how this information

can be used for reconstruction of data items, where we performed an exemplarily evaluation for the R-Tree. Later, we exemplarily extend index structures to be privacy aware. Furthermore, we evaluate our index structure modifications with respect to performance and precision. Within this evaluation, we identify, that improving privacy may also have a positive but small effect on query performance such as improving precision of the PBA.

In future work, we want to show and evaluate a method for improving privacy of data items stored in R-Tree variants. Additionally, we want to evaluate different kinds of prototype selection methods for PBA with respect to privacy. Furthermore, we will extend index structures implementations of our QuEval framework with the presented deletion strategies and evaluate performance of index structures.

7 Acknowledgments

The work in this paper has been partially funded by the German Federal Ministry of Education and Science (BMBF) through the Research Program under Contract No. FKZ:13N10816 and FKZ:13N10817. Additionally, we want to thank Ina Lindauer for her implementation of the analysis for reconstruction of data items within an R-Tree.

References

[AA96] F. Alimoglu and E. Alpaydin. Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwriting Recognition. In *TAINN*, pages 637–640. IEEE, 1996.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[BKK96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331. ACM, 1990.

[Bun09] Bundesministerium der Justiz. Bundesdatenschutzgesetz, August 2009.

[CGFN08] E. C. Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *TPAMI*, 30(9):1647–1658, 2008.

[Con96] United States Congress. Health Insurance Portability and Accountability Act (HIPAA). http://www.hhs.gov/ocr/privacy/, 1996.

[DW10] S. M. Diesburg and A. A. Wang. A survey of confidential data storage and deletion methods. *ACM Comput. Surv.*, 43(1):2:1–2:37, 2010.

[FHMW10] P. Frühwirt, M. Huber, M. Mulazzani, and E. R. Weippl. InnoDB Database Forensics. In *AINA*, pages 1028–1036. IEEE Computer Society, 2010.

[Fow08] K. Fowler. *SQL Server Forensic Analysis*. Addison-Wesley Professional, 2008.

[GBS+12] A. Grebhahn, D. Broneske, M. Schäler, R. Schröter, V. Köppen, and G. Saake. Challenges in finding an appropriate multi-dimensional index structure with respect to specific use cases. In *GvD*, pages 77–82. CEUR-WS, 2012.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[Gre12] A. Grebhahn. Forensisch sicheres Löschen in relationalen Datenbankmanagementsystemen. Master thesis, University of Magdeburg, 2012. In German.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57. ACM, 1984.

[HMCK12] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *The VLDB Journal*, 21(3):333–358, 2012.

[HXRC11] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE*, pages 601–612, 2011.

[KFV11] T. Kiertscher, R. Fischer, and C. Vielhauer. Latent fingerprint detection using a spectral texture feature. In *MMSec*, pages 27–32. ACM, 2011.

[KS07] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *SSTD*, pages 239–257. Springer-Verlag, 2007.

[Lin12] I. Lindauer. Analyse des Rekonstruktionspotentials von multidimensionalen Indexstrukturen zum sicheren Löschen. Master thesis, University of Applied Sciences Brandenburg, Germany, 2012. In German.

[Lit07] D. Litchfield. Oracle forensics part 2: Locating dropped objects. *NGSSoftware Insight Security Research (NISR) Publication, Next Generation Security Software*, 2007.

[Roy03] Royal Canadian Mounted Police. G2-003. Hard Drive Secure Information Removal and Destruction Guidelines, October 2003.

[RYZ+05] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification. *NIMPA*, 543(2-3):577–584, 2005.

[Sam05] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[SML07] P. Stahlberg, G. Miklau, and B. N. Levine. Threats to privacy in the forensic analysis of database systems. In *SIGMOD*, pages 91–102, 2007.

[SRF87] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *VLDB*, pages 507–518, 1987.

[WB97] R. Weber and S. Blott. An Approximation-Based Data Structure for Similarity Search. Technical Report 24, Zurich, Switzerland, 1997.

[WCKM09] W. K. Wong, D. Wai-Lok Cheung, B. Kao, and N. Mamoulis. Secure kNN computation on encrypted databases. In *SIGMOD*, pages 139–152, 2009.

[WJ96] D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *ICDE*, pages 516–523. IEEE Computer Society, 1996.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*, pages 194–205. Morgan Kaufmann Publishers Inc., 1998.