Slicing in Assistenzsystemen

Wie trotz Anonymisierung von Daten wertvolle Analyseergebnisse gewonnen werden können

Hannes Grunert
Lehrstuhl für Datenbank- und Informationssysteme
Universität Rostock
18051 Rostock
hg(at)informatik.uni-rostock.de

Andreas Heuer
Lehrstuhl für Datenbank- und Informationssysteme
Universität Rostock
18051 Rostock
ah(at)informatik.uni-rostock.de

Kurzfassung

Datenschutz stellt eine große Herausforderung für die Entwicklung von Informationssystemen dar. Obwohl viele Konzepte zur Wahrung des Datenschutzes bestehen, werden Datenschutztechniken in der Regel nicht in Datenbankmanagement- und Assistenzsysteme integriert.

In diesem Artikel stellen wir eine SQL-Implementierung des Slicing-Konzeptes von Li et al. [10] vor. Es erfolgt eine detaillierte Betrachtung hinsichtlich der Parametrisierung des Algorithmus und dessen Auswirkung auf Informationsverlust und Grad des Datenschutzes.

ACM Klassifikation

H.2.4 [**Database Management**]: Systems—*Query Processing*; K.4.1 [**Computer and Society**]: Public Policy Issues—*Privacy*

Stichworte

Datenbanken, Datenschutz, Datensparsamkeit

1. EINLEITUNG

Zwei Kernaspekte des Datenschutzes sind Datensparsamkeit und Datenvermeidung. §3a des Bundesdatenschutzgesetzes [1] definiert diese als Forderung, dass Informationssysteme möglichst wenig personenbezogene Daten erheben, verarbeiten oder nutzen sollen. Dies betrifft neben der Konzeption von Informationssystemen auch den Prozess der Datenverarbeitung.

Assistenzsysteme sollen den Nutzer bei der Bewältigung seines Alltags, sei es im Beruf oder zu Hause, unterstützen. Über verschiedene Sensoren, wie Bewegungssensoren und Wärmemessgeräte, werden Informationen über die momentane Situation und die Aktivitäten des Anwenders gesammelt. Diese Daten werden im System gespeichert und mit weiteren Informationen, beispielsweise dem Nutzerprofil eines sozialen Netzwerkes verknüpft. Aus den so gewonnenen Informationen lassen sich Verhaltensmuster, Präferenzen und zukünftige Ereignisse ermitteln. Auf Basis dieser Daten reagiert das Assistenzsystem eigenständig auf die Bedürfnisse des Nutzers und regelt Raumtemperatur, Lüftung und weitere vernetzte Geräte.

Assistenzsysteme sammeln häufig wesentlich mehr Informationen als sie für die Ausführung ihrer Funktionen benötigen. Der Nutzer hat dabei meist keinen oder nur einen unwesentlichen Einfluss auf die Speicherung und Verarbeitung seiner personenbezogenen Daten. Dadurch ist sein Recht auf informationelle Selbstbestimmung verletzt.

Die Einführung von Datenschutzmechanismen wird seitens der Entwickler von Assistenzsystemen skeptisch angesehen. Es wird befürchtet, dass durch die Anonymisierung der Daten die Entwicklung des Systems zurückgeworfen wird. Durch die Anonymisierung bzw. Pseudonymisierung der Nutzerdaten gehen Detailinformationen verloren, wodurch Analysefunktionen ungenauere Ergebnisse zurückliefern und im Extremfall unbrauchbar werden.

Im Rahmen des Graduiertenkollegs MuSAMA[1] werden Datenschutzkonzepte für die Anfrageverarbeitung in Assistenzsystemen entworfen. Diese werden allerdings nicht *ontop* auf die bestehenden Analysefunktionen aufgesetzt, sondern in enger Zusammenarbeit während der Entwicklung integriert. Ein Beispiel für dieses Zusammenwirken ist die Umsetzung des Slicing-Konzeptes von Li et al. [10] zusammen mit der Regressionsanalyse [6] auf hochdimensionalen Sensordaten.

1.1 PArADISE

Für die Umsetzung von Datenschutzrichtlinien in smarten Umgebungen wird derzeit das PArADISE[2]-Framework [7] entwickelt, welches insbesondere die Aspekte der Datensparsamkeit und Datenvermeidung in heterogenen P2P-Systemen realisieren soll.

PArADISE ist Werkzeug, welches den Entwicklern und Nutzern von Assistenzsystemen helfen soll effizient und datenschutzkonform Anfragen auf Sensordaten zu formulieren. Der Kern des Frameworks ist ein datenschutzfreundlicher

[1] Multimodal Smart Appliance Ensembles for Mobile Applications
[2] Privacy-Aware Assistive Distributed Information System Environment

27^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 26.05.2015 - 29.05.2015, Magdeburg, Germany.

Anfrageprozessor (siehe Abbildung 1). Dieser Prozessor erhält als Eingabe zum einen die Datenschutzeinstellungen der Nutzer, zum anderen den Informationsbedarf des Systems anhand von Anfragen an das Datenbanksystem. Als Ausgabe wird ein anonymisierter, annotierter Teil des Datenbestandes zurückgeliefert. In den Annotationen wird angegeben, wie der Datenbestand anonymisiert wurde und ob bereits die erforderlichen Analysen der Daten auf dem ausführenden Knoten umgesetzt wurden. Falls keine Analyse durchgeführt wurde, muss der Knoten, welcher das Ergebnis erhalten hat, die weitere Verarbeitung der Daten übernehmen.

Der Anfrageprozessor arbeitet in zwei Stufen. Ziel der ersten Stufe, der Vorverarbeitungsphase, ist die Modifikation der eingehenden Anfragen, sodass möglichst wenige Daten aus der Datenbank ausgelesen werden. Dazu wird die Anfrage analysiert. Durch die Analyse lässt sich erkennen, welche Attribute angefragt werden, wie diese hinsichtlich ihres Wertebereichs gefiltert und ggf. aggregiert und gruppiert werden.

Die so erhaltenen Informationen über die Anfrage werden mit den vorformulierten Privatheitsansprüchen [4] des Nutzers verglichen. Treten an dieser Stelle Widersprüche auf, wie beispielsweise der Zugriff auf ein Attribut, welches der Nutzer nicht von sich preisgeben möchte, wird die Anfrage angepasst. Dabei werden verschiedene Techniken, wie Anfrageumschreibungen und Abbildungen auf Sichten, angewandt.

In der Nachverarbeitungsphase erfolgt, sofern erforderlich, die Anonymisierung der Daten. Dabei wird überprüft, ob spezielle Kriterien wie k-Anonymität [11] zutreffen. Anschließend erfolgt die Anonymisierung der Daten unter Berücksichtigung des Grades der benötigten Anonymität und des Informationsverlustes [8] zur Wahrung von Funktionalität und Datenschutz.

Durch die Vorverarbeitung der Anfrage müssen weniger Daten in der Nachverarbeitungsphase anonymisiert werden. Dadurch erfolgt nur eine geringe Erhöhung der Antwortzeit auf die Anfrage.

1.2 Laufendes Beispiel

Die Umsetzung des Slicing-Konzeptes wird in diesen Artikel anhand eines laufenden Beispiels erläutert. Für die Implementation und Evaluation des Algorithmus wurde die Adult-Relation aus dem UCI Machine Learning Repository [9] verwendet. Die Relation besteht aus personenbezogenen Daten, bei denen Schlüssel sowie Vor- und Nachname der betroffenen Personen entfernt wurden. Die verbleibenden 15 Attribute enthalten weitere personenbezogene Angaben, wie beispielsweise Alter, Ehestand, Staatsangehörigkeit und Schulabschluss.

Tabelle 1 zeigt einen Ausschnitt der gesamten Relation. In diesem Artikel wird gezeigt, wie diese Relation schrittweise in eine anonymisierte Relation (siehe Tabelle 3) überführt wird.

Der Rest des Artikels ist wie folgt strukturiert: Kapitel 2 gibt einen Überblick über das Anonymisierungskonzept von Li et al. Im folgenden Kapitel gehen wir detailliert darauf ein, wie das Konzept durch Operationen aus der Relationenalgebra realisiert werden kann. In Kapitel 4 wird beschrieben wie die Implementation des Konzeptes in SQL erfolgt. Kapitel 5 evaluiert den Ansatz anhand des laufenden Beispiels. Das letzte Kapitel fasst den Beitrag zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

2. SLICING

In [10] stellen Li et al. ein neues Anonymisierungskonzept vor, welches auf der Permutation von Daten beruht. Im Gegensatz zu den bisher gängigen Anonymisierungsverfahren wie k-Anonymität [11] verzichtet dieses Verfahren auf Generalisierungstechniken.

Eine Relation ist k-anonym, wenn, auf Basis der identifizierenden Attributmengen der Relation, ein Tupel von k-1 anderen Tupeln nicht unterscheidbar ist. In der Regel wird eine Person durch ein Tupel in der Relation dargestellt. In Assistenzsystemen werden, je nach Anwendungsgebiet, nur Informationen über eine Person gesammelt. Dies gilt z. B. bei der Überwachung von Demenzpatienten. Dort lassen sich mittels der aufgenommenen Daten einzelne Handlungen einer Person identifizieren. Diese gilt es ebenfalls zu anonymisieren.

Bei der Generalisierung von Daten kommen Detailinformationen abhanden, die bei vielen Analysefunktionen benötigt werden. Das Slicing-Verfahren verspricht einen höheren Datennutzen, indem es die Verbindung von stark korrelierenden Attributen bewahrt. Auf dieser Grundlage ist es trotz Anonymisierung weiterhin möglich Data-Mining- und Analyse-Techniken, wie Regressionsanalysen, anzuwenden.

Das Konzept permutiert nicht den kompletten Datenbestand, sondern partitioniert die Daten sowohl horizontal als auch vertikal. Bei der horizontalen Partitionierung (*Tuple Partition*) wird der Datenbestand nach festgelegten Filterkriterien (Selektion nach einem bestimmten Attribut, Anzahl an Partitionen, ...) in mehrere Mengen von Tupeln zerlegt.

Die vertikale Partitionierung (*Attribute Partition*) unterteilt den Datenbestand spaltenweise, indem aus der vorhandenen Attributmenge mehrere kleine Attributmengen gebildet werden. Eine mögliche vertikale Zerlegung besteht im Aufteilen der Attribute eines Quasi-Identifikators [2] auf mehrere Partitionen.

Aus einem Datenbestand der durch n horizontale und m vertikale Partitionsvorgaben zerteilt wird, entsteht ein Raster aus $n*m$ kleineren Relationen. Jede dieser Teilrelationen wird anschließend permutiert, indem die Reihenfolge der Tupel vertauscht wird. Die permutierten Relationen werden anschließend wieder zu einer einzelnen Relation verknüpft.

3. UMSETZUNG IN DER RELATIONALEN ALGEBRA

Für das Slicing-Konzept wird von Li et al. eine grobe Methodik für die Implementierung vorgestellt. In diesem Abschnitt demonstrieren wir die Adaption des Verfahrens auf die relationale Algebra. Konkrete Implementierungsdetails werden im nachfolgenden Kapitel vorgestellt.

3.1 Schritt 1: Horizontaler Split

Im ersten Schritt wird die Relation R in n disjunkte Partitionen $R_1, ..., R_n$ zerlegt. Für jede Partition R_i wird eine Selektionsbedingung c_i angegeben, welche die Tupel aus R den Partitionen zuordnet:

$$R_1 := \sigma_{c_1}(R), ..., R_n := \sigma_{c_n}(R). \quad (1)$$

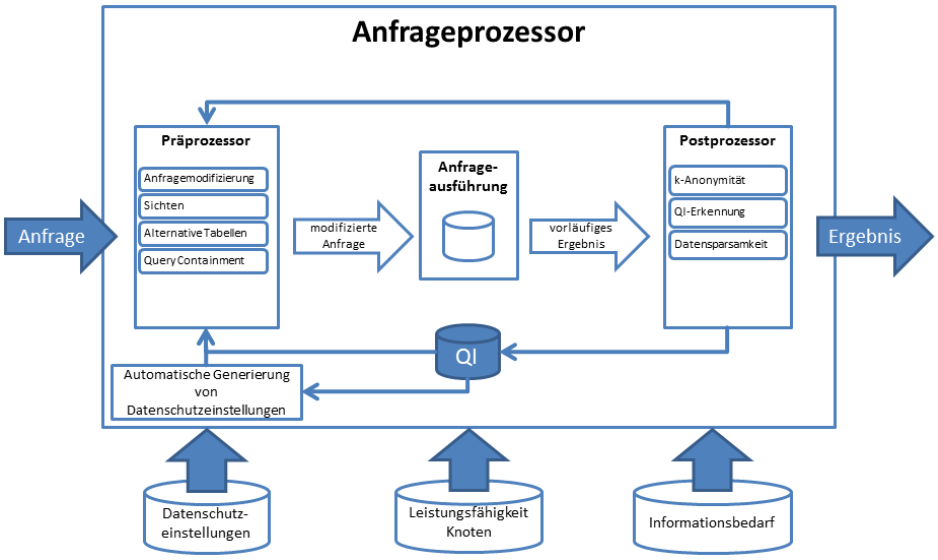

Abbildung 1: Das Konzept des datenschutzfreundlichen Anfrageprozessors

Age	Workclass	Occupation	Relationship	Race	Sex
39	State-gov	Adm-clerical	Not-in-family	White	Male
50	Self-emp	Exec-managerial	Husband	White	Male
38	Private	Handlers-cleaners	Not-in-family	White	Male
53	Private	Handlers-cleaners	Husband	Black	Male
28	Private	Prof-speciality	Wife	Black	Female
34	Private	Sales	Husband	White	Female

Tabelle 1: Die Beispielrelation vor der Anonymisierung

Die Auswahl der Selektionsbedingungen muss dabei geschickt gewählt werden. Eine Möglichkeit besteht aus der Selektion nach einem bestimmten Attribut, wobei für jeden einzelnen auftretenden Attributwert (oder eine Menge von Attributwerten) eine Selektionsbedingung der Form *attribut = 'Wert'* aufgestellt wird. Eine weitere Variante besteht darin, die Tupel in R zu nummerieren (neues Attribut *rank*) und die Selektionen in der Form '*rank between* v_j *and* v_k'[3] zu beschreiben.

3.2 Schritt 2: Vertikaler Split

Nachdem die Relation R in mehrere kleinere Relationen R_i zerlegt wurde, wird jede R_i durch Projektionen weiter unterteilt. Jede Projektion $\pi_{attributmenge_j}(R_i)$ wählt dabei ein oder mehrere Attribute aus der Relation R aus. Für jede Partitionen R_i werden dabei die gleichen m Projektionen angewandt um die m Teilrelationen R_{ij} aus R_i zu bilden:

$$R_{i1} := \pi_{attributmenge_1}(R_i), ..., R_{im} := \pi_{attributmenge_m}(R_i). \quad (2)$$

Die Attributmengen in den Projektionen müssen dabei nicht zwangsweise disjunkt sein[4]. Sie dürfen keine identifizierende Attributmenge enthalten, da ansonsten Rückschlüsse auf die Identität einer Person gezogen oder Handlungen eindeutig bestimmt werden können. Die Auswahl der Attributmengen sollte dabei in Abstimmung mit den Analysefunktionen getroffen werden, um z. B. stark korrelierende Attribute nicht auf unterschiedliche Partitionen zu verteilen. Die Ermittlung von identifizierenden Attributmengen, sogenannten Quasi-Identifikatoren [2], lässt sich mit geringem Aufwand [5] vor der Anonymisierung realisieren.

[3] v_j, v_k nummerische Werte

[4] Im grundlegenden Algorithmus von Li et al. [10] wird Disjunktheit vorausgesetzt. Erweiterungen setzen Disjunktheit nicht voraus.

3.3 Schritt 3: Permutation

Nach der Bildung der Teilrelationen R_{ij} erfolgt die eigentliche Anonymisierung. Die Permutation erfolgt durch den Ordnungsoperator τ, welcher die Tupel jeder Teilrelation sortiert. In Abschnitt 4.2 wird genauer erklärt wie eine zufällige Sortierung erfolgen kann. Das Resultat der Permutation wird in der Liste L_{ij} hinterlegt:

$$L_{ij} := \tau_{<Attribute>}(R_{ij}). \quad (3)$$

An dieser Stelle geschieht ein kleiner Bruch mit der relationalen Algebra. Der Ordnungsoperator τ darf zwar auf Relationen angewandt werden, liefert allerdings keine Relation, sondern eine Liste zurück. τ darf somit nur als letzter Operator angewandt werden [3]. Dies stellt ein Problem dar, weil für den Slicing-Algorithmus die permutierten Listen im letzten Schritt wieder zusammengefügt werden müssen.

Die Listen müssen entsprechend wieder zurück in Relationen überführt werden. Um die Ordnung zu bewahren, wird jedes Tupel mit einer Ordnungszahl *Ord* (siehe Tabelle 2) versehen, welche ihre Position in der Liste widerspiegelt. Die Ordnungszahl wird für die folgenden Verbundoperationen benötigt. Die permutierten Relationen werden mit R'_{ij} bezeichnet.

3.4 Schritt 4: Zusammenfügen

Im letzten Schritt erfolgt das Zusammenfügen der permu-

Ord	Age	Workclass	Ord	Occupation	Relationship	Ord	Race	Ord	Sex
1	39	State-gov	1	Adm-clerical	Not-in-family	1	White	1	Male
2	50	Self-emp	2	Exec-managerial	Husband	2	White	2	Male
3	38	Private	3	Handlers-cleaners	Not-in-family	3	White	3	Male
Ord	**Age**	**Workclass**	**Ord**	**Occupation**	**Relationship**	**Ord**	**Race**	**Ord**	**Sex**
1	53	Private	1	Handlers-cleaners	Husband	1	Black	1	Male
2	28	Private	2	Prof-speciality	Wife	2	Black	2	Female
3	34	Private	3	Sales	Husband	3	White	3	Female

Tabelle 2: Beispielrelation nach horizontalem und vertikalem Split. Der horizontale Split erfolgte auf dem internen Attribut *ROW_ID*. Für den vertikalen Split wurden die Attribute *Age* und *Workclass* sowie *Occupation* und *Workclass* zusammengefasst. *Race* und *Sex* wurden in jeweils einzelne Partitionen zusammengefasst.

Age	Workclass	Occupation	Relationship	Race	Sex
39	Private	Exec-managerial	Husband	White	Male
38	State-gov	Adm-clerical	Not-in-family	White	Male
50	Self-emp	Handlers-cleaners	Not-in-family	White	Male
28	Private	Handlers-cleaners	Husband	Black	Female
34	Private	Sales	Husband	Black	Female
53	Private	Prof-speciality	Wife	White	Male

Tabelle 3: Beispielrelation nach Anwendung des Slicing-Konzeptes

tierten Teilrelationen R'_{ij} zur permutierten Gesamtrelation R'. Das Zusammenfügen geschieht dabei in zwei Teilschritten.

Im ersten Teilschritt werden die permutierten Teilrelationen R'_{ij} über den Verbundoperator zu den permutierten Partitionen R'_i miteinander verbunden. Der Verbund erfolgt über die zuvor angelegte Ordnungszahl:

$$R'_i := R_{i1} \bowtie_{i1.Ord=i2.Ord} R_{i2} ... \bowtie_{i1.Ord=im.Ord} R_{im}. \quad (4)$$

Abschließend wird die Vereinigung der permutierten Partitionen R'_i gebildet:

$$R' := \bigcup_{i=1}^{n} R'_i. \quad (5)$$

R' ist das Ergebnis des Slicing-Ansatzes. Die permutierte Relation kann im weiteren Verlauf für Analysefunktion unter Einhaltung des Datenschutzes verwendet werden.

4. IMPLEMENTIERUNG

Die Umsetzung des Slicing-Konzeptes erfolgte mittels SQL-92. Einige Datenbanksysteme, wie die verwendete MySQL-DB, benötigen für die Ausführung temporäre Tabellen, die mittels *CREATE TEMPORARY TABLE* erzeugt werden. Zwecks Übersichtlichkeit wird in den Quellcodes auf diese Anweisung verzichtet. Parameter und Tabellen-Aliase werden nachfolgend in *<spitzen Klammern>* angegeben.

4.1 Horizontaler Split

Für den horizontalen Split werden alle Attribute aus der Relation *<table>* übernommen und unter einem durchnummerierten Alias *hSplitTable<id>* abgespeichert. Die Selektionsbedingungen werden für das Attribut *<attr>* mittels verschiedener Attributwerte (*<x>, <y>*) festgelegt und ggf. mit weiteren Bedingungen verknüpft (siehe Abbildung 2).

Sofern kein Selektionsattribut vorhanden ist oder explizit kein solches Attribut verwendet werden soll, so lässt sich ein künstliches Attribut einfügen. Dafür lassen sich die Zeilen der Tabelle *<table>* entweder manuell durchnummerieren

```
SELECT *
FROM <table> AS hSplitTable<i>
WHERE <attr> = <x>
--AND <attr> BETWEEN <x> AND <y>
```

Abbildung 2: SQL-Anweisung für die horizontale Zerlegung von Relationen

```
SET @rank = 0;
SELECT *,
@rank:=@rank+1 AS 'ID'
FROM <table> AS dummyTable;
```

Abbildung 3: Künstliche Erzeugung von Gruppierungsattributen basierend auf Ordnungszahlen

(siehe Abbildung 3) oder, sofern vom Datenbanksystem unterstützt, mittels der Funktion *RANK()* die Nummerierung automatisch erzeugen lassen.

Sowohl das Erzeugen als auch der horizontale Split verursachen einen linearen Aufwand. Die Komplexität dieses Schrittes beträgt $O(n*m)$, wobei n die Anzahl der Tupel in der Relation und m die Anzahl der Partitionen darstellt.

4.2 Vertikaler Split, Permutation, Ordnung

Der vertikale Split, die Permutation der Tupel und das Erzeugen der Ordnungszahl lassen sich in einer einzelnen SQL-Anweisung zusammenfassen (siehe Abbildung 4). Ausgehend von den zuvor erzeugten Partitionen *hSplitTable<i>* werden die für die Projektion benötigten Attribute *<attr-list>* in der inneren SQL-Anweisung übergeben und über zufällig erzeugte Werte sortiert (*ORDER BY RAND()*). Die äußere SQL-Anweisung übernimmt die projizierten Attribute (ohne den Zufallswert) und fügt eine Ordnungszahl *@rank* hinzu. Das Ergebnis wird unter den Alias p<i, j> abgelegt, wobei i der Index des horizontalen Splits und j der Index des vertikalen Splits darstellt.

Während die vertikale Aufteilung der Daten und das Hinzufügen der Ordnungszahlen einen linearen Aufwand (bzgl.

```
SET @rank = 0;
SELECT @rank:=@rank+1 AS Ord, <attrlist>
FROM (SELECT <attrlist> FROM hSplitTable<i>
AS dummyTbl ORDER BY RAND()) AS p<i,j>
```

Abbildung 4: SQL-Anweisung zur vertikalen Zerlegung und Permutation von Relationen

```
SELECT <attr-list>
FROM p<i,1> JOIN (p<i,1>, ..., p<i,n>)
ON (p<i,1>.Ord=p<i,2>.Ord,
    ...
    p<i,1>.Ord=p<i,n>.Ord)
AS Join<i>
```

Abbildung 5: SQL-Anweisung für die Verbundoperation

der Anzahl der Partitionen) erzeugen, benötigt die Permutation durch das anschließende Sortieren einen höheren Aufwand. Die Komplexität beträgt dabei $O(m * log(n') * n')$, wobei m die Anzahl der Teilrelationen und n' die Anzahl der Tupel innerhalb einer Teilrelation ist. Der Anteil $log(n') * n'$ stellt dabei den Aufwand für gängige Sortierverfahren. Unterstützt das verwendete Datenbankmanagementsystem hashbasierte Sortierverfahren, so beträgt die Komplexität lediglich $O(m * n')$.

4.3 Zusammenfügen

Das Zusammenfügen der Teilrelationen erfolgt in zwei Schritten. Zunächst erfolgt der Verbund der Attribute über einen Equi-Join auf den Ordnungszahlen, wobei die erste Teilrelation *p<i,1>* mit allen anderen Teilrelationen (p<i,2>, ..., p<i,n>) verknüpft wird (siehe Abbildung 5). Es erfolgt zudem eine Projektion auf die Attribute der Originalrelation *<attrlist>*, damit die Ordnungszahl im Ergebnis nicht erscheint. Das Ergebnis des Verbundes wird in der temporären Relation *Join<i>* hinterlegt.

Anschließend werden die so erzeugten Relationen Join1 bis JoinN mittels des UNION-Operators vereinigt (Siehe Abbildung 6). Das Ergebnis des Slicing-Konzeptes wird in der Relation *pRelation* ausgegeben. Diese stellt eine permutierte Version der Ursprungsrelation *<table>* dar.

5. AUSWERTUNG

Die Evaluation erfolgte in einer Client-Server-Umgebung. Als Server dient eine virtuelle Maschine, die mit einer 64-Bit-CPU (QEMU Virtual CPU version (cpu64-rhel6), vier Kerne mit jeweils @2 GHz und 4 MB Cache) und 4 GB Arbeitsspeicher ausgestattet ist. Auf dieser wurde eine

```
SELECT * FROM Join1
UNION ALL
SELECT * FROM Join2
...
UNION ALL
SELECT * FROM JoinN
AS pRelation
```

Abbildung 6: SQL Anweisung zur horizontalen Vereinigung der permutierten Partitionen

MySQL-Datenbank mit InnoDB als Speichersystem verwendet.

Der Client wurde mit einem i7-3630QM als CPU betrieben. Dieser bestand ebenfalls aus vier Kernen, die jeweils über 2,3 GHz und 6 MB Cache verfügten. Als Arbeitsspeicher standen 8 GB zur Verfügung. Als Laufzeitumgebung zur parametrisierten Generierung der SQL-Anfragen wurde Java SE 8u20 eingesetzt.

Die verwendete Adult-Relation [9] besteht aus insgesamt 32561 Tupeln, die zunächst im CSV-Format vorlagen und in die Datenbank geparst wurden.

5.1 Laufzeit

Abbildung 7 zeigt die Laufzeit der Implementation. Es wurde eine feste vertikale Partitionierung gewählt (Age und Workclass, Occupation und Relationship sowie Race und Sex als einzelne Attribute). Die 32561 Tupel wurden auf eine variable Anzahl von horizontalen Partitionen aufgeteilt (1, 2, 5, 10, 25, 50, 75, 100, 200, 300, 400, 500), sodass insgesamt zwölf Parametrisierungen getestet wurden. Für jede Variante wurden zehn Tests ausgeführt und der Mittelwert der Laufzeit gebildet.

Die Zeitmessung erfolgte zu mehreren Zeitpunkten während des Algorithmus. Die erste Messung erfolgt für die horizontale (*hSplit*), die zweite für die vertikale Partitionierung (*vSplit*). Für die Permutation und dem anschließenden Join erfolgte die dritte Zeitmessung (*Permutate/Join*). Die letzte Messung ermittelt die Zeit für die abschießenden *Union*-Operationen.

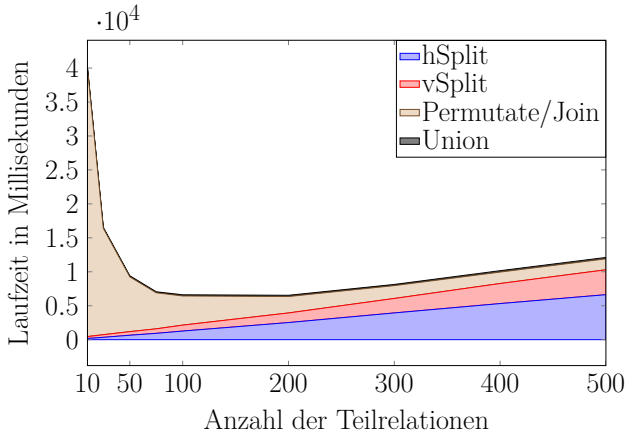

Abbildung 7: Betrachtung der Laufzeit des Slicing-Algorithmus für unterschiedlich große Partitionen. Bei großen Partitionen nimmt der Aufwand für die Permutation der einzelnen Teilrelationen drastisch zu. Kleinere Partitionen benötigen eine geringere Laufzeit für die Permutation; das Erstellen der Teilrelationen erfordert aber mehr Zeit.

Der Aufwand für die Partitionierung steigt linear mit der Anzahl der Partitionen. Dies betrifft sowohl die horizontale (51 ms bis 6626 ms), als auch die vertikale (255 ms bis 3664 ms) Partitionierung. Gleiches gilt auch für die abschließende Vereinigung, hier sind die Laufzeiten von 68 ms bis 204 ms jedoch nicht ausschlaggebend.

Die meiste Rechenzeit wird, zumindest bis zu einer Anzahl von ca. 250 horizontalen Partitionen, für die Permutation benötigt. Bei einer höheren Anzahl an Partitionen sinkt die Zeit für die Permutation. Durch die geringe Anzahl an

Anzahl Buckets	hSplit in ms	vSplit in ms	vJoin/ Permutate in ms	Union in ms
1	51	255	388846	68
2	89	252	189844	155
5	126	253	78324	142
10	177	278	39503	143
25	345	389	15664	143
50	652	531	8075	156
75	931	663	5315	145
100	1267	875	4297	172
200	2518	1415	2422	181
300	3938	2130	1896	171
400	5315	2955	1687	202
500	6626	3664	1615	204

Tabelle 4: Messwerte für die einzelnen SQL-Anfragen

Tupeln pro Partition ist der Aufwand für das Sortieren/Permutieren innerhalb einer Partition geringer.

Die Gesamtlaufzeit nimmt bis ca. 100 Partitionen stark ab (389744 ms bis 7117 ms) und stagniert bis ca. 200 Partitionen (7054 ms). Danach übersteigt der lineare Aufwand zur Partitionierung den Einsparungen für die Permutation (bis zu 12621 ms).

5.2 Anwendung mit Analysefunktionen

Die Auswirkungen der Anonymisierung auf die Analysefunktionen in Assistenzsystemen wurden am Beispiel linearer Regression getestet. Bei der linearen Regression wird eine Regressionsgerade der Form

$$y_i = \alpha + \beta * x_i + \epsilon \tag{6}$$

bestimmt. Dabei wird der Zusammenhang zwischen einer abhängigen Variablen y und einer unabhängigen Variablen x ermittelt. Hierbei werden die Parameter α und β, unter Berücksichtigung eines maximalen Fehlers ϵ, bestimmt. Für die Variablen x und y werden jeweils zwei Attribute der Datenbank getestet.

Bei der Forschung zur Modellbildung für Assistenzsysteme werden viele Regressionsanalysen mit unterschiedlichen Attributen überprüft. Die Parametrisierung der vertikalen Partitionierung des Slicing-Algorithmus erfolgt für die zu testenden Attributpaare. Dabei werden diese Kombinationen in eine Partition aufgenommen. Um den Aufwand der Anonymisierung gering zu halten, werden möglichst große Attributkombinationen gebildet. Bei der Partition der Attribute wird dabei im Vorfeld berücksichtigt, dass diese keinen Quasi-Identifikator enthalten [5]. Durch dieses Vorgehen ist es möglich, die Regressionsanalysen ohne Informationsverlust auszuführen. Durch bisher gängige Anonymisierungskonzepte, wie k-Anonymität [11], ist dies nicht gewährleistet.

6. AUSBLICK

In dieser Arbeit stellten wir die Umsetzung des Slicing-Ansatzes von Li et al. auf Basis der relationalen Algebra vor. Dieses Anonymisierungsverfahren bietet einen guten Kompromiss zwischen Datenschutz und Analysefunktionalitäten durch Minimierung des Informationsverlustes.

Anhand von linearen Regressionsanalysen wurde gezeigt, dass das Slicing-Verfahren mit den Funktionalitäten von Assistenzsystemen harmoniert. Um festzustellen, für welche Analysefunktionen ein geeignetes Anonymisierungsverfahren existiert, sind weiterführende Arbeiten notwendig. Im Rahmen des PArADISE-Projektes werden dazu weitere Datenschutztechniken und Analysefunktionen in den vorgestellten Anfrageprozessor integriert.

7. DANKSAGUNG

Hannes Grunert wird durch die Deutsche Forschungsgemeinschaft (DFG) im Rahmen des Graduiertenkollegs 1424 (Multimodal Smart Appliance Ensembles for Mobile Applications - MuSAMA) gefördert. Wir danken den anonymen Gutachtern für ihre Anregungen und Kommentare.

8. LITERATUR

[1] Bundesrepublik Deutschland. Bundesdatenschutzgesetz, 2010.

[2] Tore Dalenius. Finding a Needle In a Haystack or Identifying Anonymous Census Records. *Journal of Official Statistics*, 2(3):329–336, 1986.

[3] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (international edition)*. Pearson Education, 2002.

[4] Hannes Grunert. Privacy Policy for Smart Environments. `http://www.ls-dbis.de/pp4se`, 2014. zuletzt aufgerufen am 13.05.2015.

[5] Hannes Grunert and Andreas Heuer. Big Data und der Fluch der Dimensionalität: Die effiziente Suche nach Quasi-Identifikatoren in hochdimensionalen Daten. In *Proceedings of the 26th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken)*. `http://ceur-ws.org`, 2014.

[6] Albert Hein, Frank Feldhege, Anett Mau-Möller, Rainer Bader, Uwe Zettl, Oliver Burmeister, and Thomas Kirste. NASFIT - Intelligente Assistenzsysteme zur Funktionsunterstützung und Therapieüberwachung bei neuromuskulären Störungen. In *Ambient Assisted Living 7. AAL-Kongress 2014 Berlin, Germany, January 21-22., 2014*, Tagungsbände, Berlin, Germany, January 2014. VDE Verlag.

[7] Andreas Heuer. METIS in PArADISE: Provenance Management bei der Auswertung von Sensordatenmengen für die Entwicklung von Assistenzsystemen. In *Datenbanken für Business, Technologie und Web - Workshopband*, volume 242 of *Lecture Notes in Informatics*, pages 131–135. Springer, 2015.

[8] Ayça Azgin Hintoglu and Yücel Saygın. Suppressing microdata to prevent classification based inference. *The VLDB Journal*, 19(3):385–410, 2010.

[9] Ronny Kohavi and Barry Becker. Adult Data Set. `http://archive.ics.uci.edu/ml/datasets/Adult`, 1996. zuletzt aufgerufen am 13.05.2015.

[10] Tiancheng Li, Ninghui Li, Jian Zhang, and Ian Molloy. Slicing: A new approach for privacy preserving data publishing. *Knowledge and Data Engineering, IEEE Transactions on*, 24(3):561–574, 2012.

[11] Pierangela Samarati. Protecting Respondents' Identities in Microdata Release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.