Vergleichende Analyse von Datenschutzalgorithmen und -konzepten

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

vorgelegt von: Bodo John
Matrikelnummer: 213206744
geboren am: 28.06.1988 in Bergen auf Rügen
Erstgutachter: Prof. Dr.rer.nat. habil. Andreas Heuer
Zweitgutachter: Dr.-Ing. Holger Meyer
Betreuer: Hannes Grunert
Abgabedatum: 29.08.2016

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 29.08.2016

Inhaltsverzeichnis

1 Einleitung

Die Entwicklung der Technik schreitet wie gewohnt voran. Technische Geräte haben die Evolutionsstufen der intelligenten Handlung erreicht und können dem Menschen auf vielfältige Weise assistieren. Um zu assistieren statt zu behindern müssen richtige Entscheidungen für das Verhalten getroffen werden. Dies geschieht über vorliegende Daten. Je nach Assistenzsystem können bzw. müssen diese Daten permanent aktualisiert werden. Daraus ergeben sich riesige Datenbestände die meist schnell analysiert werden müssen um eine entsprechend notwendige Aktion des Systems zu erreichen. Beispielhaft kann die Messung von Bewegungsdaten von Personen in ihrer Wohnung genannt werden. Verschiedene Sensoren sammeln Daten und können in Notfallsituationen Rettungsdienste alarmieren. Ein anderes Beispiel wäre ein Assistenzsystem eines Pkw, welcher unter Umständen permanent Daten über das Verhalten des Pkw sammelt und wiederum bei einer Gefahrensituation im Straßenverkehr eingreift, z.B. durch die Steuerung der Geschwindigkeit des Personenkraftwagens oder durch die Alarmierung von Rettungsdiensten bei Unfällen anderer oder des eigenen Pkw.
Die Vorteile von Assistenzsystemen sind deutlich zu erkennen und ebenso die Gefahren ihrer Arbeitsweise. Bei der Datensammlung sind alle Informationen meist direkt einer Person zugeordnet. Das gesamte gewonnene Wissen kann für, aber auch gegen den Menschen eingesetzt werden. Assistenzsysteme werden von Unternehmen hergestellt, welche mit dem Verkauf von Rohdaten oder ausgewerteten Daten mit Erkenntnissen über das Verhalten von eindeutig bestimmten Personen Geld verdienen können. Mit den Erkenntnissen können auch gleich direkte Werbemaßnahmen angewendet werden. Ein Beispiel aus einer Autozeitschrift spezifiziert das Problem mit einem Unfallszenario und dem Notrufsystem eCall, welches automatisch Rettungskräfte alarmiert. Die Zusatzdienste sind hier das Problem, welche nach dem Willen der Versicherungswirtschaft allen Marktteilnehmern die gesammelten Daten zur Verfügung stellen sollen [Gul14]. Geschwindigkeitsüberschreitungen wird die Versicherungsgesellschaft aus den Daten erkennen und Beiträge bei den jeweiligen Fahrern erhöhen wollen.
Der Gesetzgeber in Deutschland schreibt in §3a des Bundesdatenschutzgesetzes Datensparsamkeit und Datenvermeidung vor. Daten sollen nur im nötigen Umfang und soweit wie möglich anonymisiert werden, soweit Aufwand und Schutzzweck dies zulassen [BRD]. Um den Anforderungen zu begegnen gibt es verschiedene Anonymisierungsverfahren für die Daten. Vor der Datenspeicherung und die Herausgabe an Dritte können die Daten soweit anonymisiert werden, dass sie keiner einzelnen Person mehr zugeordnet werden können. Anhand des Verhaltens von Samsung [Beu15] und LG [Hof13] in Bezug auf ihre Smart-TV Produkte und die Datensammlung ist deutlich, dass es reges Interesse an persönlichen Daten und Verhalten gibt, welche auch eindeutig zugeordnet sind. Zur Erfüllung der Aufgaben der lokalen Assistenzsysteme benötigen die Hersteller keine detaillierten Informationen die jedem einzelnen Nutzer direkt zugeordnet werden. Die Assistenzsysteme können Daten zum Teil mittels eigenen Ressourcen auswerten und vor allem anonymisieren, sodass der Transfer aller gesammelten Daten zu den Servern der Hersteller vermieden wird.

1.1 Problemstellung

Das Projekt PArADISE[1] des Lehrstuhls Datenbank- und Informationssysteme an der Universität Rostock hat den Privatheitsaspekt und die Performanz von Analysesystemen zu seinen Hauptaufgaben gemacht[HM15]. Bei der Datenanalyse soll bereits ein Großteil der Anfrage in Sensornähe erledigt werden [GH14]. Die Abbildung 1 soll dies verdeutlichen.

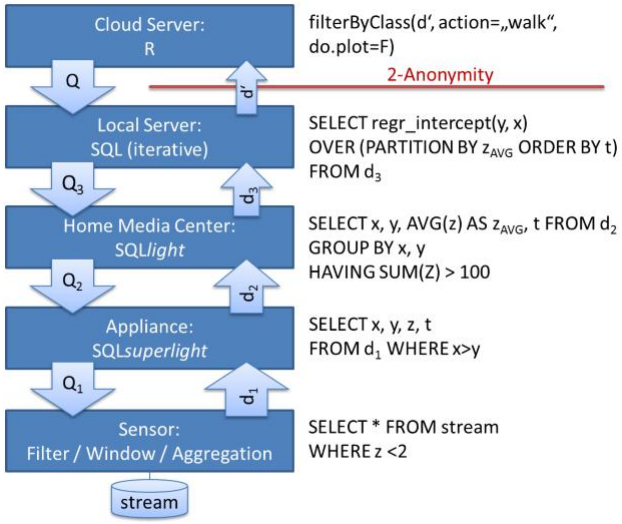


Abbildung 1: Anfrageumschreibung [GH16]

Erkennbar ist, dass die Gesamtanfrage aufgeteilt wird und die Sensoren bereits eine einfache Select-From-Anfrage mit ihren Ressourcen durchführen können. Bevor die Daten vom lokalen Server in die Cloud des Herstellers geladen werden, werden sie mittels Datenschutzalgorithmen anonymisiert. Dadurch wird die Restanfrage Q_δ auf den modifizierten Daten durchgeführt und damit die Datensparsamkeit durchgesetzt [GH16]. Es gibt diverse Anonymisierungsalgorithmen und es muss festgestellt werden, welches Verfahren bei welcher Anfrage und Datenmenge am besten geeignet ist. Durch die Kriterien Speicherbedarf, Zeitkomplexität, sowie durch den Informationsverlust der Anonymisierung können die Verfahren bewertet werden.

1.2 Gliederung

In dieser Arbeit werden die Datenschutzalgorithmen zur Anonymisierung von Daten vorgestellt 2. Diese sind k-Anonymität, l-Diversity, t-Closeness, Slicing und Differential Privacy. Es werden ihre Komplexität untersucht und die ihrer Approximationsalgorithmen 3. Es werden mögliche Anfragearten vor der Anonymisierung identifiziert 4. Danach soll ein Entscheidungsalgorithmus entwickelt werden, der bei entsprechender Eingabe ein Datenschutzverfahren empfiehlt 5. Eine Evaluation des implementierten (α, k)-Prototypen wird in Kapitel 6 getätigt. Eine Zusammenfassung 7 stellt die wesentlichen Ergebnisse dar.

[1] Privacy AwaRe Assistive Distributed Information System Environment

2 Datenschutzverfahren

Folgende grundlegende Konzepte werden näher erläutert: k-Anonymity, l-Diversity, t-Closeness, Slicing und Differential Privacy. Allen gemein ist die Unterteilung der Attribute eines Datenbestandes in Quasi-Identifikator, sensible Daten und Schlüssel. Das Schlüsselattribut bestimmt ein Tupel eindeutig. Die Schlüssel und sensiblen Daten dürfen nicht zusammen offen gelegt werden [GH14]. Dies würde in einem expliziten Beispiel bedeuten, dass Max Mustermann (Schlüssel) zusammen mit seiner Bachelorabschlussnote (sensibler Datensatz) in einem öffentlichen Aushang steht. Im Gegensatz dazu ist die Veröffentlichung der Matrikelnummer zusammen mit der Note akzeptabel, da die Matrikelnummer ein Pseudonym darstellt, welches der jeweiligen Person und dem Dienstleister, hier das Studienbüro, bekannt ist. Ein Quasi-Identifikator ist eine Teilmenge von Attributen, welche die meisten Tupel identifizieren kann [Dal86].

2.1 k-Anonymität

Die k-Anonymität ist vorhanden, wenn eine Person oder ein Datensatz mit mindestens k-1 Personen oder Datensätzen nicht zu unterscheiden ist. Um sie zu erreichen, kann ein Rauschen hinzugefügt oder Informationen unterdrückt oder generalisiert werden [SS98]. Die Generalisierung funktioniert bei numerischen Werten nach dem Prinzip des Zusammenfassens von Werten in Wertintervalle. Bei anderen Datentypen wird in feste Hierarchien eingeordnet. Die folgende Abbildung 2 verdeutlicht dies in Bezug auf das Attribut Alter.

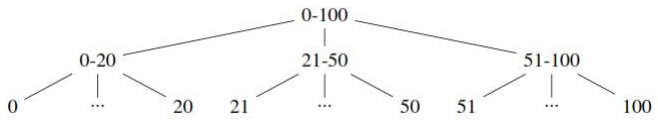


Abbildung 2: Prinzip der Generalisierung [Mü16]

Die Abbildung zeigt die Einteilung der einzelnen Attributwerte in die höheren Ebenen des Baumes, wie z.B. der Wert „0-20“. Sollte eine Generalisierungsstufe nicht ausreichen, wird die nächste genommen, in der Abbildung 2 „0-100“. Die folgende Tabelle 1 soll beispielhaft die Daten eines Krankenhauses darstellen mit Patienten, die an einem Tag eingeliefert wurden.
Wird die Generalisierung als Methode genutzt, ergibt sich eine Tabelle bei der die PLZ verallgemeinert wurde um eine eindeutige Identifikation zu verhindern. Dies war ebenfalls bei dem Alter der Personen notwendig. Eine k-Anonymität mit k=2 wird erreicht. Das heißt, dass einer Person mindestens zwei Datensätzen zugeordnet werden kann und sie damit nicht eindeutig bestimmt werden kann. Die Staatsangehörigkeit wurde vollkommen weggelassen bzw. unterdrückt. Dies entspricht der Methode der Informationsunterdrückung. Die Attribute Alter, Geschlecht und PLZ sind die Quasi-Identifikatoren in der Beispieltabelle, da mit diesen Attributwerten die Mehrzahl der Personen eindeutig bestimmbar ist.
Es gibt dennoch die Möglichkeit die k-Anonymität zu attackieren und damit dennoch Personen eindeutig zu bestimmen. Diese Attacken sind unsortiertes Verlinken, komplementäres Veröffentlichen und zeitliche Attacke [Swe02]. Weitere Möglichkeiten ergeben sich durch Homogenitätsattacken und starkes Hintergrundwissen [Hau07]. Homogenitätsattacken sind möglich durch die

Name	Alter	Geschlecht	PLZ	Staatsangehörigkeit	Krankheit
Hans Spargel	14	männlich	18528	deutsch	Hodenkrebs
Anna Schmidt	7	weiblich	18528	deutsch	Keuchhusten
Karl Einfalt	15	männlich	18528	deutsch	Bronchitis
Hilde Wilde	65	weiblich	18540	deutsch	Grippe
Susanne Müller	71	weiblich	18540	deutsch	Herzinfarkt
Maria Mauer	88	weiblich	18546	deutsch	Darmkrebs
Klaus Kur	16	männlich	18528	deutsch	Grippe
Anne Haber	9	weiblich	18528	deutsch	Magersucht
Diana Will	8	weiblich	18528	deutsch	Grippe
Johann Kraus	69	männlich	18546	deutsch	Lungenkrebs
Max Interesse	66	männlich	18551	deutsch	Gehirnerschütterung
Mustafar Jamir	68	männlich	18551	amerikanisch	Gehirnerschütterung
Sylvia Sauer	65	weiblich	18546	deutsch	Diabetes

Tabelle 1: Beispiel Patientendaten

Alter	Geschlecht	PLZ	Krankheit
<18	männlich	18528	Hodenkrebs
<18	weiblich	18528	Keuchhusten
<18	männlich	18528	Bronchitis
<18	männlich	18528	Grippe
<18	weiblich	18528	Magersucht
<18	weiblich	18528	Grippe
>=65	weiblich	18540	Grippe
>=65	weiblich	18540	Herzinfarkt
>=65	weiblich	18546	Darmkrebs
>=65	weiblich	18546	Diabetes
>=65	männlich	185**	Lungenkrebs
>=65	männlich	185**	Gehirnerschütterung
>=65	männlich	185**	Gehirnerschütterung

Tabelle 2: Patiententabelle k-anonymisiert

textbf Alter	**Geschlecht**	**PLZ**	**Krankheit**
<18	männlich	18528	Hodenkrebs
<18	weiblich	18528	Keuchhusten
<18	männlich	18528	Bronchitis
<18	männlich	18528	Grippe
<18	weiblich	18528	Magersucht
<18	weiblich	18528	Grippe
>=65	weiblich	18540	Grippe
>=65	weiblich	18540	Herzinfarkt
>=65	Person	185**	Darmkrebs
>=65	Person	185**	Diabetes
>=65	Person	185**	Lungenkrebs
>=65	Person	185**	Gehirnerschütterung
>=65	Person	185**	Gehirnerschütterung

Tabelle 3: Patiententabelle mit l-Diversity

Eingrenzung einer Person auf zwei oder mehr Tupel und erfolgt bei vorliegenden gleichen sensitiven Daten.

2.2 l-Diversity

Die Problematik des starken Hintergrundwissens löst das Konzept der l-Diversity und soll an einem Beispiel verdeutlicht werden. In unserer Tabelle existieren zwei weibliche Patienten die in Sassnitz wohnen und mindestens 65 Jahre alt sind. Nehmen wir an, dass Sylvia Sauer in der Buchhaltung im Hafen Mukran tätig ist. Ihr Arbeitgeber und ihre Kollegen wissen, dass sie am Wochenende ins Krankenhaus in Bergen eingeliefert wurde und Dienstag wieder zur Arbeit erschienen ist. Die Tochter einer Angestellten in der gleichen Abteilung arbeitet im Krankenhaus und hatte Zugriff auf die anonymisierte Tabelle 2. Folglich ist klar, dass Sylvia Diabetes oder Darmkrebs hat. Anlässlich einer Abschiedsfeier einer weiteren Kollegin, die in Rente geht, gibt es Kuchen. Sylvia lehnt den Kuchen nicht ab und isst zwei große Stücken. Daraus kann geschlussfolgert werden, dass sie Darmkrebs hat. Die Tabelle muss also weiter anonymisiert werden um die Problematik zu vermeiden. Die folgende Tabelle zeigt eine weitere anonymisierte Tabelle die l-Diversity erfüllt. Für Sylvia gibt es jetzt mehr Möglichkeiten der Krankheit.

Definition 2.1 (Entropy l-Diversity)**.** Eine Tabelle besitzt Entropy l-Diversity, wenn für jeden q*-Block gilt:

$$-\sum_{s\in S} p_{(q*,s)} \log(p_{(q*,s')}) \geq \log(l)$$

wobei

$$p_{(q*,s)} = \frac{n_{(q*,s)}}{\sum_{s' in S} n_{(q*,s')}}$$

der Bruch der n Tupel im q*-Block mit sensitiven Attributwerten wie dem sensitiven Attributwert s ist. [MKGV07].

Ein q*-Block ist eine Menge von Tupeln in einer Tabelle, deren nicht-sensitive Attributwerte zu q* generalisiert wurden. Aus der Definition des Entropy l-Diversity ergibt sich, dass jeder q*-Block mit mindestens l unterschiedlichen sensitiven Werten aufgelistet ist. Damit der Angreifer erfolgreich ist, muss er l-1 mögliche Werte entfernen [MKGV07]. In unserem Beispiel sind jetzt vier Krankheiten für Sylvia möglich und dementsprechend ist l-1=3. Vorher war l-1=1 und damit kein Schutz vorhanden.
Rekursives (c, l)-diversity dient der Sicherstellung, dass keine sensitiven Attribute zu häufig oder zu wenig vorkommen. Gerade Werte die häufig bzw. selten sind ermöglichen es wieder Verbindungen herzustellen[LLV07]. Die gleiche Verteilung der Attribute gibt c an, wobei je geringer c, desto verteilter sind sie. Des Weiteren gibt es Positive Disclosure-Recursive (c, l)-diversity und Negative/Positive Disclosure-Recursive (c, l)-diversity [Hau07].

2.3 t-Closeness

Bei dem t-Closeness-Verfahren wird der Wissensgewinn eines Angreifers bei der Betrachtung der Verteilung Q, der sensitiven Attributewerte in der gesamten Tabelle erreicht. Der Angreifer kennt die Quasi-Identifikatoren und ist in der Lage die Äquivalenzklasse zu identifizieren in der die für ihn zu identifizierende Person ist. Er lernt die Verteilung P, der sensiblen Attribute in dieser Klasse. Der Vergleich der Gesamtverteilung Q zu der Verteilung P ergibt eine Distanz, sodass wenn Q und P nahe beieinander sind, dann ist auch der Wissenszuwachs bzw. das Ausgangswissen W1 und erweiterte Wissen W2 nahe beieinander [MKGV07].

Definition 2.2 (t-Closeness)**.** Ein q*-Block besitzt t-Closenenss, wenn die Distanz zwischen den zu veröffentlichenden sensitiven Attributen dieses Blocks zur gesamten Tabelle nicht mehr als der vorher festgelegte Grenzwert t beträgt. Eine Tabelle besitzt t-Closeness, wenn alle q*-Blöcke von ihr t-Closeness besitzen [Hau07].

Zwischen zwei probabilistischen Verteilungen die Distanz zu messen ist problematisch. Bei Verteilungen P = $(p_1, ..., p_m)$ und Q = $(q_1, ..., q_m)$ ist eine mögliche Messvariante die Variational Distanz:

$$D[P, Q] = \sum_{i=1}^{m} \frac{1}{2} |p_i - q_i|.$$

Des Weiteren gibt es die Earth Mover's Distanz. Um allerdings kategorische Attributwerten muss noch eine Umrechnungsformel definiert werden, um Distanzen zu berechnen. Die Distanz zweier kategorischer Attributwerte wird meist mittels Generalisierung bewertet, sodass sie in der gleichen Domäne sind [Hau07].

2.4 Slicing

Der folgende Abschnitt nutzt das Paper „Slicing in Assistenzsystemen“ als Quelle [GH15]. Der Vorteil beim Slicing ist der Erhalt von Verbindungen von stark korrelierenden Attributen. Dies ermöglicht einen höheren Datennutzen für Analyseverfahren. Die Vorgehensweise des Slicing beginnt mit der horizontalen und vertikalen Partitionierung (Tuple Partition). Der Datenbestand wird bei dem horizontalem Split mittels Filterkriterien in mehrere Mengen von Tupeln zerlegt. Die vertikale Partitionierung unterteilt die Attributmenge in mehrere kleine Attributmengen

Alter	Geschlecht	PLZ	Krankheit
14	männlich	18528	Grippe
7	weiblich	18528	Keuchhusten
15	männlich	18528	Magersucht
16	männlich	18528	Hodenkrebs
9	weiblich	18528	Grippe
8	weiblich	18528	Bronchitis
65	weiblich	18540	Herzinfarkt
71	weiblich	18551	Gehirnerschütterung
88	weiblich	18546	Diabetes
69	männlich	18546	Lungenkrebs
66	männlich	18540	Grippe
68	männlich	18546	Darmkrebs
65	weiblich	18551	Gehirnerschütterung

Tabelle 4: Patientendatentabelle mittels Slicing

durch spaltenweisen Split des Datenbestandes. Der Datenbestand wird durch n horizontale und m vertikale Splits zu $n*m$ kleineren Relationen. Die Teilrelationen werden permutiert und danach wieder zu einer einzelnen Relation zusammengeführt. Die folgende Tabelle 4 zeigt das bisherige Beispiel nach dem Slicing.
Die horizontale Zerlegung lässt sich in SQL mittels einer Selektion nach einem Attribut für jeden einzelnen Attributwert oder einen Wertebereich durchführen. Die Selektionsbedingungen werden durch verschiedene Attributwerte festgelegt. Dafür werden alle Attribute aus der Tabelle in einem durchnummerierten Alias gespeichert. Alternativ werden die Tupel in der Relation durchnummeriert und die Selektion mittels internen Zeilennummern durchgeführt. Die folgende Abbildung zeigt den SQL-Code.

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

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

Abbildung 3: SQL Horizontaler Split [GH15]

```
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 Vertikaler Split, Permutation [GH15]

Von den erzeugten Partitionen werden die notwendigen Attribute der inneren SQL-Anweisung aus der unteren Abbildung übergeben. Mittels der Rand()-Funktion werden Attributwerte als Zufallszahlen erzeugt und durch Order-By sortiert. Den ausgewählten Attributen wird eine Ordnungszahl hinzugefügt. Abschließend werden die Teilrelationen mittels Join und Union zusammengefügt.

2.5 Differential Privacy

Definition 2.3 (Differential Privacy)**.** Eine randomisierte Funktion K ist $\epsilon-$ Differential Privacy, wenn für alle Datensätze D_1 und D_2 die sich in höchstens einem Element unterscheiden und für alle Teilmengen aus der Menge anonymisierten Werte $S \subseteq Range(K)$ gilt:

$$PR[K(D_1) \in S] \leq e^{\epsilon} * Pr[K(D_2) \in S].[\text{Dwo}]$$

Differential Privacy begrenzt den Effekt jedes einzelnen Eintrages in der Datentabelle auf das Ergebnis des genutzten Algorithmus. Selbst das Löschen eines Datensatzes erhöht nicht die Wahrscheinlichkeit der Identifikation anderer Datensätze bzw. Personen. Die Verteilung der Ausgabe bleibt gleich [Dwo]. Die Wahrscheinlichkeit des Outputs bei gegebenen Datensatz D_1 ist gleich dem wahrscheinlichen Output bei Datensatz D_2. Die Abbildung 5 soll dies verdeutlichen. Die Abbildung beinhaltet zwei Kurven. Die schwarze Kurve ist die Verteilung bei D_1. Ist die Eingabe nun D_2, also ein Datensatz mit einem zusätzlichen Wert, ergibt sich die rote Kurve. Die Kurve ist fast gleich geblieben, aber nach rechts verschoben.

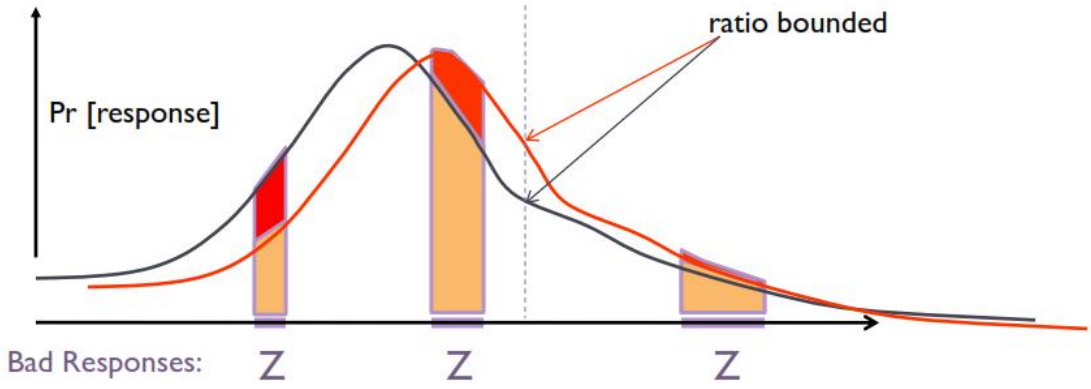


Abbildung 5: Differential Privacy

Die Verschiebung und die Veränderung der wahrscheinlichen Ausgabe sind begrenzt. Je größer die Verschiebung der Kurve, desto größer der Einfluss eines Datensatzes. Definiert ist dies durch die Sensitivität.

Definition 2.4 (L1-Sensitivität)**.** Für $f : D \to R^d$, ist die L1-Sensitivität von f

$$\Delta f = \max_{D_1, D_2} ||f(D_1) - f(D_2)||_1$$

für alle D_1, D_2 die sich höchstens in einem Element unterscheiden [Dwo].

Definition 2.5 (ϵ, δ-Differential Privacy)**.** Ein randomisierter Algorithmus $M : X^n \to Y$ ist (ϵ, δ) Differential Privacy, wenn für zwei Datensätze $S, S' \in X^n$ die sich in der Zeile unterscheiden und jeder Datensatz $T \subseteq Y$, dann gilt

$$PR[M(S) \in T] \leq e^{\epsilon} * Pr[M(S') \in T] + \delta[\text{NS15}].$$

Erreicht werden kann Differential Privacy durch das Hinzufügen von Rauschen bei den Daten bzw. Algorithmen [Dwo]. Es gibt mehrere Möglichkeiten mittels Rauschen Differential Privacy zu erreichen. Der Laplace-Algorithmus ist einer davon. Er ist eine Instanz des Exponential Mechanismus und liefert approximierte Summen durch die Anwendung von beschränkten Funktionen auf dem Datenbestand. Wenn q eine lineare Anfrage ist, führt der Laplace Mechanismus

$$Pr[L(B) = r] \propto exp(-\epsilon * |r - q(B)|)[\text{HLM12}].$$

aus.

Inputs: Data set B over a universe D; Set Q of linear queries; Number of iterations $T \in \mathbb{N}$; Privacy parameter $\varepsilon > 0$; Number of records n.

Let A_0 denote n times the uniform distribution over D.
For iteration $i = 1, ..., T$:

1. ***Exponential Mechanism:*** Select a query $q_i \in Q$ using the Exponential Mechanism parameterized with epsilon value $\varepsilon/2T$ and the score function
$$s_i(B, q) = |q(A_{i-1}) - q(B)| .$$
2. ***Laplace Mechanism:*** Let measurement $m_i = q_i(B) + Lap(2T/\varepsilon)$.
3. ***Multiplicative Weights:*** Let A_i be n times the distribution whose entries satisfy
$$A_i(x) \propto A_{i-1}(x) \times \exp(q_i(x) \times (m_i - q_i(A_{i-1}))/2n) .$$

Output: $A = \text{avg}_{i<T} A_i$.

Abbildung 6: Laplace Mechanismus [HLM12]

Die Abbildung 6 zeigt den Multiplicative Weights Exponential Mechanism[HLM12]. Bei diesem Algorithmus wird Laplace Rauschen hinzugefügt. Der Algorithmus läuft über T Iterationen und bei jeder Iteration wird der Exponential Mechanismus angewendet. Es werden entsprechende Anfragen ausgewählt, welche mit $\frac{\epsilon}{2T}$ parametrisiert sind und die Auswertungsfunktion $s_i(B, q)$ nutzen. Es folgt die Anwendung des Laplace Rauschen. Abschließend werden multiplikative Update-Regeln angewendet und es erfolgt die Ausgabe. Eine weitere Möglichkeit Rauschen beizugeben ist Gaußsches Rauschen. Des Weiteren wird bei hoher Sensitivität die Sparse Vektor Technik genutzt [DR14]. Der Exponential Mechanismus von McSherry und Talwar soll vorgestellt werden. Eine Qualitätsfunktion $q : X^* * F \to \Re$ definiert ein Optimierungsproblem und eine Lösungsmenge F für die gilt: gegeben eine Datenbasis S in X^*, wähle f in F so das $q(F, f)$

maximiert. Der Exponential Mechanismus löst ein solches Problem durch das Wählen einer randomisierten Lösung, bei der die Wahrscheinlichkeit einer ausgegebenen Lösung f exponentiell mit ihrer Qualität $q(F, f)$ zunimmt.
Als weitere Methode des Differential Privacy ist die Subsample and Aggregate Methode zu nennen. Die Abbildung 7 verdeutlicht die Idee.

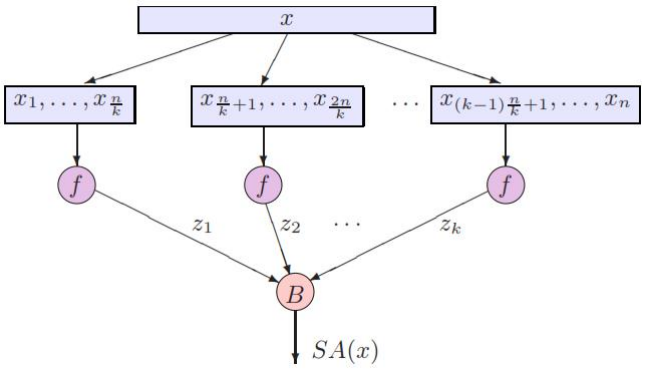


Abbildung 7: Algorithmus Subsample and Aggregate [Smi11]

Der Input x wird in zufällige k Blöcke der Größe n/k geteilt. Die Funktion f erstellt zu jedem der k Blöcke eine Schätzung $z_1 \ldots z_k$. Diese Schätzungen werden aggregiert und dabei die Differential Privacy Funktion B genutzt [Smi11]. Die Subsample and Aggregate Methode wird in dieser Arbeit nur für die Vollständigkeit genannt und nicht weiter betrachtet.

3 Speicher- und Zeitkomplexität

Als Quelle dieses und des folgenden Abschnittes dient das Buch „Algorithmik: Die Kunst des Rechnens“[HF06]. Die Effizienz von Algorithmen wird in der Informatik mit den Maßen der Komplexität gemessen. Es gibt die Platzkomplexität, welche den Speicherbedarf misst und die Zeitkomplexität, welche den Zeitbedarf misst. Die Platzkomplexität wird unter anderem durch die Anzahl der Variablen und die Größe der eingesetzten Datenstrukturen beeinflusst. Die Zeitkomplexität wird durch die Anzahl der Aktionen, die der Prozessor während des Programmdurchlaufs ausführt, ermittelt. Die theoretische Informatik bestimmt die Komplexität auf einer abstrakteren Ebene um Unabhängigkeit von Faktoren, wie der jeweiligen Prozessorleitung, zu erhalten. Bei der Bestimmung der Komplexität sind vor allem Rekursionsaufrufe und Schleifendurchläufe maßgebend. Die Anzahl der Schleifendurchläufe und die Tiefe des Rekursionsbaumes geben die Komplexität an [Hof15]. Der Bedarf an Zeit und Platz variiert mit den Eingaben und ergibt die Performanz des Algorithmus mit den unterschiedlichen Eingaben.

3.1 Komplexitätsklassen

Die Zeit- und Platzkomplexität gibt Auskunft über den Ressourcenverbrauch eines Algorithmus und damit über seine Performanz. Dementsprechend gibt es gute und nicht so gute Performanzen für die guten und nicht so guten Algorithmen. Allerdings gibt es oft keine besseren Algorithmen für gewisse Probleme. Diese Probleme zu lösen ist so komplex, dass ihre Lösung unvernünftig hohen Einsatz von Ressourcen erfordert. Grob gesagt gibt es zwei Arten von Problemen bzw. ihren Lösungen: deterministisch-polynomiell und nicht-deterministisch-polynomiell lösbar. Die Probleme, die in Polynomialzeit lösbar sind, haben polynomielle Wachstumsraten bei entsprechenden Eingaben und werden als handhabbar bezeichnet. Unhandhabbar sind die Probleme mit exponentiellem Zeitaufwand. Die Abbildung 8 verdeutlicht das Wachstum.

	Funktion \ N	20	60	100	300	1000
polynomial	$5N$	100	300	500	1500	5000
	$N \times \log_2 N$	86	354	665	2469	9966
	N^2	400	3600	10,000	90,000	1 Million (7 Stellen)
	N^3	8000	216,000	1 Million (7 Stellen)	27 Millionen (8 Stellen)	1 Milliarde (10 Stellen)
exponentiell	2^N	1 048 576	eine 19-stellige Zahl	eine 31-stellige Zahl	eine 91-stellige Zahl	eine 302-stellige Zahl
	$N!$	eine 19-stellige Zahl	eine 82-stellige Zahl	eine 161-stellige Zahl	eine 623-stellige Zahl	unvorstellbar groß
	N^N	eine 27-stellige Zahl	eine 107-stellige Zahl	eine 201-stellige Zahl	eine 744-stellige Zahl	unvorstellbar groß

Abbildung 8: Wachstumsraten polynomieller und exponentieller Funktionen[HF06]

Zu den polynomiellen Funktionen gehören auch die linearen Funktionen. Diese haben das geringste Wachstum und werden durch ihre Konstanten beeinflusst. Polynomielle Funktionen erreichen bei größer werdenden Eingaben mitunter auch schnell steigende Ausgaben. Die Funktion N^3 zeigt dies. Ist N=20 ist die Ausgabe 8000 und ist N=100 ist die Ausgabe schon 1 Million. Bei den exponentiellen Funktionen ist bereits bei N=20 eine große Ausgabe im Millionen-Bereich und wächst bis ins Unermessliche bei N^N. Die folgende Abbildung 9 verdeutlicht grafisch die Wachstumsraten einiger Funktionen.

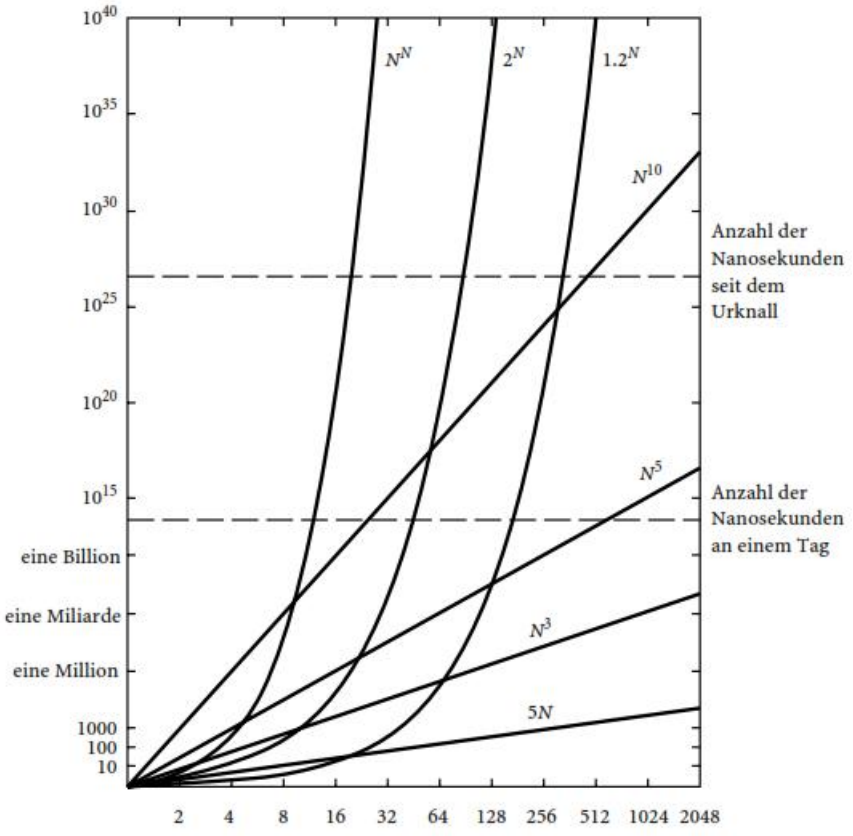


Abbildung 9: Grafik Wachstumsraten polynomieller und exponentieller Funktionen [HF06]

Es wird deutlich, dass exponentielle Funktionen bei geringen Eingaben von Vorteil sein können. Die Funktion 1.2^N ist bis zu einer Eingabe von 16 besser als die Funktion $5N$. Dennoch ist das rasante Wachstum der exponentiellen Funktionen unverkennbar.
Die Probleme, die polynomielle Algorithmen besitzen und damit handhabbar sind, gehören zu der Klasse P. Die Klasse NP beinhaltet die Probleme, die durch nichtdeterministische Algorithmen in Polynomialzeit gelöst werden können. Die schwersten Probleme in NP sind die *NP-vollständigen* (NPC). Es gibt Polynomialzeit-Reduktionen von jedem Problem in NP auf jedes der NPC. Die Abbildung 10 zeigt die Komplexitätsklassen in einem Mengendiagramm mit einigen Beispielproblemen.
Die Abbildung zeigt, dass NP die Klasse P beinhaltet. Es gibt aber auch noch Klassen über NP, deren Probleme noch schwieriger zu sein scheinen. Prinzipiell erkennbar ist:

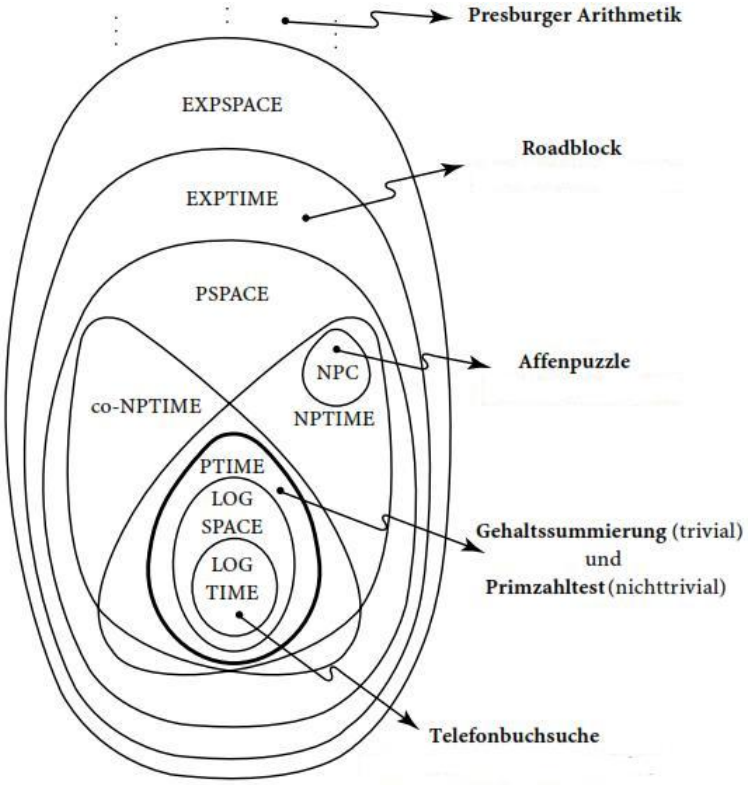


Abbildung 10: Komplexitätsklassen in Anlehnung der Abbildung aus „Algorithmik: Die Kunst des Rechnens“ [HF06]

$$LOGTIME \subseteq LOGSPACE \subseteq PTIME \subseteq NPTIME \subseteq PSPACE \subseteq EXPTIME \subseteq EXPSPACE \subseteq 2EXPTIME....$$

LOG steht für logarithmisch, P für polynomiell, EXP für exponentiell und 2EXP für doppelt exponentiell. Co-NP in der Abbildung ist eine duale Komplexitätsklasse. Aufwendiger als NP ist $PSPACE$ und die Klassen mit dem Präfix EXP. Die EXP-Klassen beinhalten Probleme die nachweisbar unhandhabbar sind. Alle Probleme die in $2EXPTIME$ und $2EXPSPACE$ sind, sind unangemessen und doppelt exponentiell 2^{2^N}. Dies geht dann so weiter mit $3EXPTIME$ usw. bis K-fach exponentiellen Funktionen. Zu den Problemen in NP mit NPC gibt es noch $NP - hard$.

Definition 3.1 (NP-schwer, NP-vollständig)**.** Eine Sprache L’ $\subseteq \Sigma^*$ (Σ^* ist das Alphabet) heißt

- NP-schwer, falls für alle L $\in$ NP die Bezeichnung L$\leq_{poly}$ L’ gilt.
- NP-vollständig, falls L’ NP-hart und ein Element von NP ist.

NPC bezeichnet die Klasse aller NP-vollständigen Probleme[Hof15].

NP-schwere Probleme sind nicht in NP, aber da jedes Problem aus NP in Polynomialzeit auf sie reduziert werden kann, sind sie mindestens so schwer wie sie bzw. mindestens so schwer wie die NP-vollständigen.
Die Einteilung in deterministisch-polynomiell und nicht-deterministisch-polynomiell lösbar ist meistens nicht ausreichend. Als Beschreibung der Komplexität wird unter anderem die Groß-O-Notation genutzt. Ein Beispiel wäre die Komplexität von $O(N)$, wobei das O für „von der Ordnung von“ steht. Das $O(N)$ steht für lineares Wachstum, da die Laufzeit linear mit N wächst. Es existiert eine Konstante K, sodass der Algorithmus im schlimmsten Fall mit einer Zeit von K*N läuft. Bei der Groß-O-Notation interessiert es nicht, ob die Anzahl der elementaren Anweisungen 3*N, N oder gar 100N ist, da nur die Art des Wachstum wichtig ist, nämlich linear.

3.2 (α, k)-Anonymität

Der folgende Abschnitt nutzt für die Ausführungen und Abbildung als inhaltliche Quelle „(α, k)-anonymous data publishing“[WLFW09]. Die (α, k)-Anonymität als erweiterte Variante der k-Anonymität beschützt nicht nur die Identifikation von Personen, sondern auch die Relation zu sensitiven Daten. Dadurch sind Homogenitätsattacken nicht mehr möglich. Bei (α, k)-Anonymität ist α ein Bruch und k gibt, wie bei der k-Anonymität, die mögliche Zuordnung von Daten zu Personen an. Nachdem Werte anonymisiert wurden, darf in keiner Äquivalenzklasse ein sensitiver Wert häufiger als der prozentuale Wert α vorkommen. Die Abbildung zeigt das Ergebnis einer (0.5,2)-Anonymität als Anwendung auf die Daten der Abbildung 11.

Job	Birth	Postcode	Illness
Cat 1	1975	4350	HIV
Cat 1	1955	4350	HIV
Cat 1	1955	5432	flu
Cat 1	1955	5432	fever
Cat 2	1975	4350	flu
Cat 2	1975	4350	fever

Abbildung 11: Beispiel medizinische Rohdaten[WLFW09]

Job	Birth	Post code	Illness
*	*	4350	HIV
*	*	4350	HIV
*	*	5432	flu
*	*	5432	fever
*	*	4350	flu
*	*	4350	fever

Abbildung 12: Anwendung (0.5,2)-Anonymität[WLFW09]

In der Abbildung sind die Daten soweit anonymisiert, dass keine Homogenitätsattacke möglich ist. Durch die Anonymisierung der Jobkategorie und das Geburtsjahr gibt es für jede PLZ mehrere Erkrankungen. Diese Möglichkeit lieferte bisher nur l-Diversity. In der Abbildung 12 ist *global recoding* angewendet worden. Dies bedeutet, dass die Werte eines Attributes auf den selben

Domainlevel der Hierarchie gehoben werden. Im Gegensatz dazu ist *local recoding* die Generalisierung einzelner Datensätze, sodass der Wert 1975 in der ersten Spalte Birth der Abbildung 11 entfernt wird, aber in den letzten beiden Zeilen bestehen bleiben.

Definition 3.2 (α-Deassocoation Bedingung)**.** Gegeben sei ein Datensatz D, ein Attributsatz Q und ein sensitiven Wert s in der Domain des Attributs S $\notin$ Q. Sei (E,s) die Tupel in der Äquivalenzklasse E die s für S enthält und α sei ein nutzerspezifischer Schwellenwert bei dem 0 $< \alpha <$1 gilt. Der Datensatz D ist α-deassociated hinsichtlich der Attribute Q und dem sensitiver Wert s, wenn die Frequenz (im Bruch) von s in jeder Äquivalenzklasse kleiner oder gleich α ist. Das heißt, $|(E, s)|/|E| \leq \alpha$ gilt für alle Äquivalenzklassen E [WLFW09].

Definition 3.3 ((α, k)-Anonymization)**.** Eine Tabellensicht ist (α, k)-anonymisiert, wenn die Tabellensicht soweit modifiziert wurde, dass die Tabellensicht die k-Anonymität und α-Deassocoation Eigenschaften hinsichtlich der Quasi-Identifikatoren erfüllt [WLFW09].

Das Erreichen von (α, k)-Anonymität ist durch *global* und *local recoding* möglich. Der Vorteil bei *local recoding* ist der Erhalt von mehr Informationen. Das Umcodieren (recoding) ist mit Aufwand verbunden. Die Kosten dafür können durch die Anzahl der Unterdrückung bzw. die Anzahl der generalisierten Werte (mit * gekennzeichnet) ermittelt werden. Die optimale k-Anonymität mittels *local recoding* zu erreichen ist NP-schwer. Mittels *local recoding* die optimale (α, k)-Anonymizatät zu erreichen ist ebenfalls NP-schwer. Beweisbar ist dies durch die Transformation des Edge Partition in 4-Clique Problem auf das (α, k)-Anonymitätsproblem in Polynomialzeit [WLFW09].
Der erweiterte Incognito Algorithmus als global recoding Algorithmus löst das Problem der optimalen (α, k)-Anonymität. Dieser ist allerding nicht skalierbar und kann zu extremen Ausreißern im Datensatz führen. Alternativ sind die progressive local recoding und Top-Down-Approach Algorithmen nutzbar. Das progressive local recoding hat eine Laufzeit von $O(p * n * m * log(m))$, wobei p die durchschnittliche Tiefe der Hierarchie der Quasi-Identifikatoren, n die Anzahl der Attribute der Quasi Identifikatoren und m die Anzahl der Tupel im Datensatz ist. Die Abbildung 13 zeigt den progressive local recoding Algorithmus.
Verglichen mit dem global recoding Algorithmus sind die beiden local recoding Algorithmen durchschnittlich viermal schneller und geben dreimal weniger Ausreißer aus den Datensatz [WLFW09]. Die folgende Abbildung 14 zeigt eine grafische Darstellung der Laufzeiten.
Der erweiterte Incognito Algorithmus weist ein exponentielles Wachstum auf. Der Top-Down Algorithmus ist wesentlich schneller, zeigt jedoch ebenfalls eine exponentielle Entwicklung. Es existieren keine Angaben über die Speicherkomplexität, diese soll über eigene Messungen ermittelt werden.

3.3 Verteilte k-Anonymität

Als Quelle der folgenden beiden Abschnitte wird der Artikel „D2Pt: Privacy-Aware Multiparty Data Publication“ genutzt [NJTF15]. Die Daten sind über verschiedene Orte verteilt, welche auch physisch voneinander getrennt und vertikal sowie horizontal partitioniert werden können. Das DPP_2GA implementiert verteilte k-Anonymität bei vertikal partitionierten Daten. Es soll ein global k-anonymisierter Datensatz von zwei lokal k-anonymisierten Datensätzen von verschiedenen Stellen erstellt werden. Während der Ausführung des Algorithmus ist die k-Anonymität des

Input: data set D, quasi-identifier Q, a sensitive attribute S or a sensitive value in S, an integer k, and a fraction α
Output: (α, k)-anonymous view V

test if D has an (α, k)-anonymous table and return FALSE if not
$V \leftarrow \emptyset$
while $D \neq \emptyset$ **do**
  let D' contain all precisely α-deassociated trunks
  $D_r \leftarrow D - D'$
  $V \leftarrow V \cup D'$
  $q_{max} \leftarrow \lfloor |D_r| - \frac{|(D_r, s)|}{\alpha} \rfloor$
  choose a set of at most q_{max} tuples in D_r satisfying (α, k)-anonymity
  let D'' be the set of chosen tuples
  $D \leftarrow D_r - D''$
  $V \leftarrow V \cup D''$
  if $D \neq \emptyset$ **then**
    choose one attribute A in Q with the highest entropy
    generalize D according to attribute A
  end if
end while
return V

Abbildung 13: progressive local recoding [WLFW09]

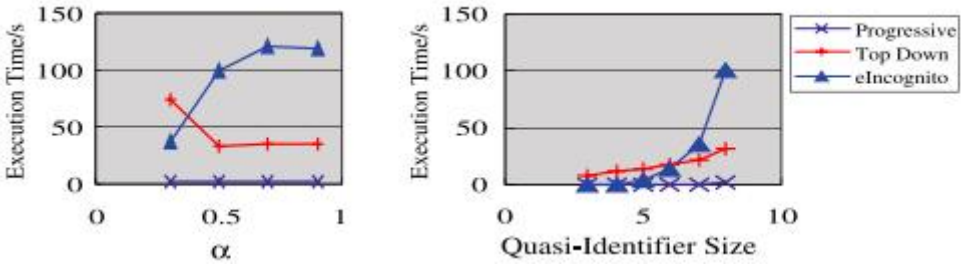
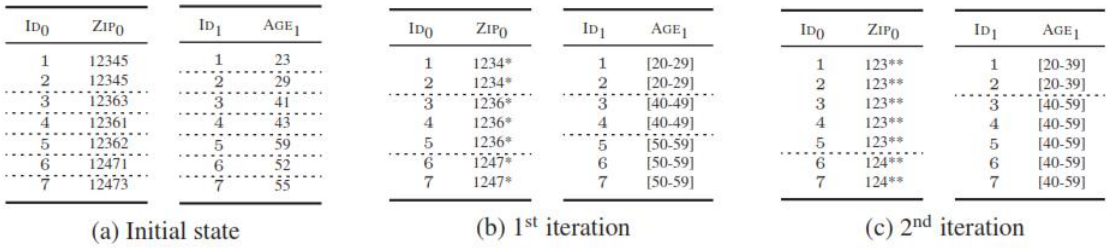


Abbildung 14: Anwendung (0.5,2)-Anonymität [WLFW09]

lokalen Datensatzes gewahrt. Die Schritte von DPP_2GA sind: lokale k-Anonymität sicherstellen, Austausch verschlüsselter Informationen über lokale Äquivalenzklassen, globale k-Anonymität sicherstellen und Zusammenfügen lokaler Datensätze durch eine globale ID (GID). Um die lokale k-Anonymität herzustellen wurde der Datafly Algorithmus verwendet. Dieser hat eine Laufzeit von $O(N * logN)$ [Swe98]. In der Abbildung 15 sind die Rohdaten eines Beispiels erkennbar, welche umgeformt werden zu lokaler k-Anonymität.
Es sind auf beiden Seiten verschiedene Äquivalenzklassen vorhanden. Die beiden Seiten tauschen verschlüsselte Informationen über ihre Äquivalenzklassen aus und vergleichen sie. Wenn die Äquivalenzklassen keine globale k-Anonymität erreicht haben, wird eine neue lokale k-Anonymität der Seiten hergestellt und wieder die Äquivalenzklassen vergleichen. Nachdem eine globale k-Anonymität erreicht wurde, werden die Partitionen über die ID vereinigt. Es existieren keine Angaben über die Speicherkomplexität.



ID_0	ZIP_0
1	12345
2	12345
3	12363
4	12361
5	12362
6	12471
7	12473

ID_1	AGE_1
1	23
2	29
3	41
4	43
5	59
6	52
7	55

(a) Initial state

ID_0	ZIP_0
1	1234*
2	1234*
3	1236*
4	1236*
5	1236*
6	1247*
7	1247*

ID_1	AGE_1
1	[20-29]
2	[20-29]
3	[40-49]
4	[40-49]
5	[50-59]
6	[50-59]
7	[50-59]

(b) 1st iteration

ID_0	ZIP_0
1	123**
2	123**
3	123**
4	123**
5	123**
6	124**
7	124**

ID_1	AGE_1
1	[20-39]
2	[20-39]
3	[40-59]
4	[40-59]
5	[40-59]
6	[40-59]
7	[40-59]

(c) 2nd iteration

Abbildung 15: Verteilte 2-Anonymität mittels DPP_2GA [NJTF15]

3.4 Verteilte t-Closeness

Die Komplexität des t-Closeness stellt bei der Berechnung der optimalen Generalisierung ein NP-schweres Problem dar, wenn $0 \leq t < 1$[LY13]. Das Distributed two-Party t-Closeness (D2Pt) ist eine Erweiterung des Distributed Privacy-Preserving two-Party Generic Anonymizer (DPP_2GA) Protokolls und erweitert das DPP_2GA in Bezug auf das t-Closeness-Konzept. Das Protokoll besteht aus Anonymizer, Synchronizer und Fragmenter. Der Anonymizer erstellt mittels einem Anonymisierungsalgorithmus einen lokalen k-anonymen Datensatz. Je nach Algorithmus wird eine Anzahl an Durchläufen benötigt bis eine generalisierte Tabelle entsteht, die t-Closeness erfüllt. Wird z.B. der Datafly Algorithmus verwendet, wird die Laufzeit wie im vorherigen Abschnitt $O(N * logN)$ betragen. Durch den Synchronizer entstehen auf beiden Seiten gleichgroße Äquivalenzklassen. Danach wird durch den Fragmenter die Zusammenführung und Freigabe der Datensätze getätigt. Die Datensätze erfüllen die Anforderungen der Privatheit des t-Closeness [NJTF15]. Die Abbildung 16 verdeutlicht das Fragmentieren.

GID	ZIP_0	AGE_1
1	1234*	[20-29]
1	1234*	[20-29]
2	*	[40-59]
2	*	[40-59]
2	*	[40-59]
2	*	[40-59]
2	*	[40-59]

GID	DISEASE
1	Gastric ulcer
1	Gastritis
2	Flu
2	Stomach cancer
2	Pneumonia
2	Bronchitis
2	Flu

GID	TREATMENT
1	Antacid
1	Acid-reducing drug
2	Antipyretic drug
2	Cytostatic drug
2	Antibiotics
2	Antibiotics
2	Antipyretic drug

Abbildung 16: Verteiltes t-Closeness mittels $D2Pt - Merge$ [NJTF15]

Die GIDs wurden erstellt und als neues Attribut eingefügt, um die Referenz zwischen Quasi-Identifikatoren und sensitiven Attribut zu erhalten. Durch die GID wird ein sensitives Attribut mit k Quasi-Identifikatoren assoziiert. Das k entspricht hier der Größe der jeweiligen Äquivalenzklasse. In der Abbildung 16 wird beispielsweise die Gastritis mit zwei Quasi-Identifikatoren verknüpft entsprechend der Größe der Äquivalenzklasse. Es gab keine Angaben über die Speicherkomplexität.

3.5 t-Closeness - SABRE

SABRE (Sensitive Attribute Bucketization Redistribution) ist ein Rahmenkonzept für t-Closeness. Es wird die Tabelle mit Daten in Buckets gleichartiger sensitiver Attribute partitioniert und danach werden die Tupel jedes Buckets in dynamisch festgelegte Äquivalenzklassen umverteilt [CKKT11]. Der Algorithmus wird in Abbildung 17 gezeigt.

```
1  Let {v_1, v_2, ..., v_m} be all the SA values in DB, and
   {p_1, p_2, ..., p_m} be their distributions;
2  Let VP be the list of (v_i, p_i), i = 1, 2, ..., m;
3  if SA is categorical then
4      Let H be the domain hierarchy of SA;
5      φ = bucketCat(H, VP);
6  else
7      Sort VP in the ascending order of SA values;
8      φ = bucketNum(VP);
9  S_a = ECSize(φ);
10 foreach array a in S_a do
11     Create an empty EC, say G;
12     foreach a_i, ith element of a do
13         ec_i = takeOut(B_i, a_i);
14         add ec_i to G;
15     output G;
```

Abbildung 17: SABRE Algorithmus [CKKT11]

Die Liste der sensitiven Attribute mit ihrer Verteilung in der gesamten Datenbank wird entweder mittels *bucketCat* oder *bucketNum* in die Buckets aufgeteilt. *bucketCat* wird bei kategorischen Attributen und *bucketNum* bei numerischen Attributen genutzt. Danach erfolgt die dynamische Festlegung der Größe jeder Äquivalenzklasse durch *ECSize* und es wird eine Liste von Arrays (S_a) zurück gegeben. Für ein Array a nimmt SABRE a_i Tupel aus Bucket $B_i \in \varphi$, wobei a_i das i-te Element aus a ist ind i=1,2,...$|\varphi|$ [CKKT11]. Es werden aus den Tupeln Äquivalenzklassen geformt und ausgegeben.
Es gibt zwei Instanziierungen von SABRE: SABRE-KNN und SABRE-AK. Sie unterscheiden sich in der Auswahl der Tupel die aus den Buckets in die Äquivlenzklassen gefügt werden. SABRE-KNN findet die exakten Nachbarn und SABRE-AK die approximierten Nachbarn. Die Zeitkomplexität von SABRE-KNN ist $O(|S_G| * |DB|)$, wobei $|S_G|$ die Anzahl der Äquivalenzklassen ist und $|DB|$ die Größe des Datensatzes. SABRE-AK ist effizienter und hat eine durchschnittliche Zeitkomplexität von $O(|S_G| * (log\frac{|DB|}{|\varphi|} + \frac{|DB|}{|S_G|*|\varphi|}) * \varphi)$[CKKT11]. $|\varphi|$ ist die Anzahl der Buckets, $\frac{|DB|}{|\varphi|}$ ist die durchschnittliche Größe der Buckets und $\frac{|DB|}{|S_G|*|\varphi|}$ ist die durchschnittliche Anzahl an Tupel, welche aus den Buckets entnommen wird um die Äquivalenzklassen zu bilden. Über die Speicherkomplexität existieren es keine Aussagen.

3.6 l-Diversity

Die Komplexität der l-Diversity stellt bei der Berechnung der optimalen Generalisierung ein NP-schweres Problem dar, wenn $l \geq 3$[XYT10]. Es ist zudem APX-hard [DMZ11]. Das heißt, es gibt keine Approximationsalgorithmen, die dieses Problem in polynomieller Zeit lösen.
Es gibt den verbesserten V-MDAV Algorithmus für l-Diversity. Die folgende Abbildung 18 zeigt den Algorithmus.

```
Algorithm: l-diversity V-MDAV algorithm
Input: dataset T, diversity parameter l
Output: l-diversity table T'
Step: 1. Compute the distances between the records and
store them in a distance matrix.
    2. Compute the centroid c of the dataset.
    3. Repeat, until diversity of remaining records < l
       (1)Let r be the most distant record to c, and form
          a group g_r={r}
       (2)Size down ungrouped vectors of the dataset
          according to their distance to vector r, and
          form distance priority queue Q.
       (3)Repeat, until equivalence class g_r satisfy l-
          diversity
       { diversity_first=D(g_r)
         pop the head vector v from queue Q
         diversity_Second=D(g_r+v)
         if (diversity_second>diversity_first)
         { g_r= g_r+v
                  Mark v with assigned sign;   }
       }
         (4)Extend group g_i
    4. Assign the remaining records to their closest
group.
```

Abbildung 18: V-MDAV Algorithmus [JmTtHq08]

Der V-MDAV Algorithmus generiert anonymisierte Datentabellen mit hoher Datenqualität. Der verbesserte V-MDAV Algorithmus ist gegen Homogenitäts- und Hintergrundwissen-Attacken sicher. Es werden Gruppen generiert die l-Diversity erfüllen. Um optimale k-Partitionierung zu erreichen, werden diese Gruppen auf die Größe zwischen l und $2l-1$ erweitert. Die Komplexität von l-Diversity V-MDAV ist $O(n^2)$ [JmTtHq08]. Es gab keine Angaben über die Speicherkomplexität.

3.7 Slicing

Das Slicing wurde in Kapitel 2.4 mit Codebeispiel erläutert. Eine horizontal und vertikale Partitionierung unterteilt die Attributmenge. Der horizontale Split benötigt linearen Aufwand mit m Partitionen und n Tupeln in der Relation ergibt das $O(n * m)$. Die vertikale Partitionierung braucht zwar auch nur linearen Aufwand, aber das anschließende Permutieren und Sortieren von m Teilrelation und n' Tupeln innerhalb einer Teilrelation verursacht einen Aufwand von

$O(m * \log(n') * n')$. Datenbankmanagementsysteme mit hashbasierten Sortierverfahren erreichen eine Komplexität von $O(m * n)$ [GH15]. Es gab keine Angaben über die Speicherkomplexität.

3.8 Differential Privacy

Der Multiplicative Weights Exponential Mechanism wurde in Abschnitt 2.5 vorgestellt. Er nutzt Laplace- und Exponential Mechanism mit einer Laufzeit von $O(n*|Q|+T*|D|*|Q|)$, wobei Q eine Menge von linearen Anfragen ist, D das Universum vom Datensatz, T die Anzahl der Iterationen und n die Anzahl der Datensätze [HLM12]. Die Komplexität von Laplace'schem- und Gausschem Rauschen ist $O(1)$ [NKK13]. Die Subsample and Aggregate Methode hat eine Komplexität von $O(n * log * n)$, wenn die Daten allerdings sortiert sind $O(n)$ [DL09]. Nach der Anwendung der Methode muss eine entsprechende Aggregatsfunktion angewendet werden. Prinzipiell gibt es keine Beschränkungen für die Wahl der Aggregatsfunktion. Oft wird der Median genommen oder das wionsorisierte Mittel (Winzorized mean) [DR14]. Eine Möglichkeit ist die Funktion nach Nissim et al. mit einer Komplexität von $O(m^2 * log * m)$ [NRS07]. Es existieren keine Angaben über die Speicherkomplexität, diese soll über eigene Messungen ermittelt werden.

4 Anfragearten

Bei der Aufnahme von Sensordaten entstehen viele Daten auf denen auf der untersten Ebene der Datenverarbeitung bereits einfache SQL-Anfragen erfolgen sollen. Auf den höheren Ebenen werden dann die Anfragen komplizierter. Dieser Abschnitt soll einen Überblick über die möglichen Anfragearten geben. Als Hauptquelle wird das Buch „Datenbanken Konzepte und Sprachen“ genutzt [SSH10]. Die einfachsten Anfragen sind Selektionen und Projektionen. Selektionen wählen über ein Selektionsprädikat Zeilen einer Tabelle aus. Projektionen wählen Spalten einer Tabelle aus. Nehmen wir als Beispiel die Tabelle 1 und nennen Sie Patienten. Als beispielhafte Projektionsanfrage an diese Tabelle wollen wir die Namen und das Alter der Patienten abfragen:

SELECT Name, Alter
FROM Patienten

Ergebnis ist eine Tabellensicht mit den Spalten Name und Alter mit ihren Attributwerten. Mit der Selektion können in unserem Beispiel die gesamten Daten von einzelnen Patienten abgefragt werden. Eine Beispiel ist:

SELECT *
FROM Patienten
WHERE Name = 'Hans Spargel'

Bei der Einteilung der Anfragearten kann in einfache und komplexe Anfragen unterschieden werden. Einfache Anfragen sind eher kurz. Dennoch kann bei der Projektion und Selektion die Anzahl der Selektionsprädikate und Attributlisten groß sein. Kompliziertere Anfragen ergeben sich mit Schachtelungen und Aggregationen. Als einfache Anfrage soll folgendes Beispiel dienen:

SELECT Name, Alter, PLZ
FROM Patienten
WHERE Alter <65 and PLZ = '18528'

Rückgabe sind alle Namen der Patienten die unter 65 sind und im Postleitzahlgebiet 18528 wohnen. Die Ausgabe sieht wie in Tabelle 5 aus.

Name	Alter	PLZ
Hans Spargel	14	18528
Anna Schmidt	7	18528
Karl Einfalt	15	18528
Klaus Kur	16	18528
Anne Haber	9	18528
Diana Will	8	18528

Tabelle 5: Ausgabe der Patientendaten nach einfacher SQL-Anfrage

```
SELECT v.Name,
       v.Vorname,
       v.Strasse,
       v.Hausnummer as HNR,
       v.PLZ,
       v.Ort,
SUM(ka.Kosten) as Aufwand,
COUNT(ka.ID) as Anzahl
FROM Versicherte v
       join Versichertenmitgliedschaft vm
       on vm.ID = v.ID
       join Krankenversicherungsleistung kvL
       on kvL.Versicherungsstatus_ID = vm.ID
       join Krankenhaus kh
       on kh.ID = kvL.kh_ID
       join Krankenhausaufenthalt ka
       on ka.ID = kvL.erhalteneLeistung_ID
       where EXTRACT(YEAR from ka.Datum) = 2008
GROUP BY v.Name, v.Vorname, v.Strasse, v.Hausnummer, v.PLZ, v.Ort
ORDER BY Aufwand, Anzahl;
```

Abbildung 19: Beispiel komplexe SQL-Anfrage

Die komplexen SQL-Anfragen beziehen meistens nicht nur eine Tabelle ein, sondern nutzen Daten aus einer bzw. mehreren anderen Tabellen. Die folgende Abbildung 19 zeigt eine komplexere Anfrage. Bei dem Beispiel werden die Daten von Versicherten gesucht, die 2008 einen Krankenhausaufenthalt hatten und die angefallenen Kosten dafür.
Die *with recursive* Funktion in SQL ermöglicht es eine Iteration in der Abfrage zu haben und damit mehrere Ergebnistupel zu erhalten. Die with recursive Anfrage besteht aus einem rekursiven und nicht rekursiven Teil. Das folgende Beispiel ist aus der Dokumentation von PostgreSQL [Pos].
Der Aufbau ist immer der gleiche. Erst ist ein *nichtrekursiver* Teil vorhanden, dann kommt das UNION und dann der *rekursive* Teil. Es wird zuerst der nicht-rekursive Teil ausgewertet und das UNION/ UNION ALL. Die Ergebnisse werden in eine temporäre Tabelle abgelegt. Die

```
WITH RECURSIVE t(n) AS (
       VALUES (1)
       UNION ALL
       SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Abbildung 20: Rekursive SQL-Anfrage

```
SELECT Name, Jahrgang
FROM Weine
WHERE Jahrgang < (
SELECT avg (Jahrgang)
From Weine)
```

Abbildung 21: Einfache Anfrage mit Aggregatfunktion [SSH10]

Iteration bricht ab, wenn keine weiteren Werte geliefert werden. Im Beispiel erfolgt eine Anfrage auf die Rekursionstabelle. In dieser Tabelle stehen die Werte von 1 bis 99. Diese werden durch die letzte Abfrage mit der Funktion *sum* aufsummiert.
Aggregatfunktionen arbeiten im Gegensatz zu den skalaren Operationen tupelübergreifend. Es werden Tupelmengen betrachtet und von ihnen entsprechende Eigenschaften berechnet. Diese Eigenschaft ist ein tupelübergreifender Wert. Diese Werte der Menge kann z.B. der Durchschnitt der Tupelwerte sein. Die wichtigsten Aggregatfunktionen sind **count, sum, avg** und **max** bzw. **min**.

Aggregatfunktion	**Berechnung**
count	Anzahl der Werte einer Spalte
count(*)	Anzahl der Tupel einer Relation
sum	Summe der Werte einer Spalte (nur numerische Wertebereiche)
avg	arithmetischen Mittelwert der Werte einer Spalte (nur numerische Wertebereiche)
max	größten Wert einer Spalte
min	kleinsten Wert einer Spalte

Tabelle 6: Übersicht Aggregatfunktionen [SSH10]

Es gibt zeilen- und spaltenbasierte Anfragen an Datenbanken. Zeilenbasierte Anfragen wählen ganze Zeilen einer Tabelle aus die ein Kriterium erfüllen. Spaltenbasierte Anfragen beziehen sich auf einzelne Spalten der Tabelle. Die Aggregatsfunktionen sind mehrheitlich spaltenbasiert. Die in der Tabelle 6 angegebenen Aggregatfunktionen dienen zum Zählen und Aufsummieren von Werten, der Durchschnittsbildung und dem Herausfiltern des größten bzw. kleinsten Wertes. Mit den dargestellten Aggregatfunktionen lassen sich Anfragen formulieren. Die folgende Anfrage 21 soll exemplarisch die mögliche Nutzung darstellen.
Als Ergebnis der Anfrage 21 werden alle Namen von Weinen, deren Jahrgang kleiner als der durchschnittliche Jahrgang aller Weine ist, ausgegeben. Der Code 22 soll ein etwas komplexeres Beispiel darstellen.
In diesem Beispiel 22 werden alle Weingüter ausgegeben, bei denen das Durchschnittsalter ihrer Weine größer als Durchschnittsalter aller Weine ist.
Es gibt noch weitere Aggregatfunktionen, die seit dem SQL:2003-Standard ergänzend eingeführt wurden. Diese dienen komplexeren statistischen Auswertungen. Es können Berechnungen der

```
SELECT Erzeuger.Weingut
FROM Erzeuger natural join Weine
GROUP BY Erzeuger.Weingut
HAVING avg (year(current_date) - Jahrgang) > (
        SELECT avg (year(current_date) - Jahrgang)
From Weine)
```

Abbildung 22: Komplexere Anfrage mit Aggregatfunktionen [SSH10]

Varianz, Kovarianz und Standardabweichung getätigt werden. Die Tabelle 7 listet die erweiterten Aggregatfunktionen seit SQL:2003 auf.

Aggregatfunktion	**Berechnung**
stddev_pop	Standardabweichung gesamte Population
stddev_samp	Standardabweichung Stichprobe
var_pop	Varianz gesamte Population
var_samp	Varianz Stichprobe
covar_pop	Kovarianz gesamte Population
covar_samp	Kovarianz Stichprobe

Tabelle 7: Übersicht über Aggregatfunktionen die mit dem SQL Standard aus dem Jahr 2003 eingeführt wurden[SSH10]

Die pop-Varianten beziehen sich auf die gesamte Population und die samp-Varianten auf Stichproben. Weitere mögliche Funktionen, die der statistischen Auswertung dienen, sind in Tabelle 8 zu sehen.

Aggregatfunktion	**Berechnung**
corr	Korrelationskoeffizienten
regr_r2	Bestimmtheitsmaß
regr_slope	Anstieg einer linearen Gleichung
regr_intercept	y-Achsenabschnitt einer linearen Gleichung

Tabelle 8: Übersicht über komplexere Aggregatfunktionen [Sch16]

5 Entscheidungsalgorithmus

Ziel dieser Arbeit ist ein Entscheidungsalgorithmus zu erarbeiten, welcher bei gegebener Anfrage eine Empfehlung für ein oder mehrere Anonymisierungsverfahren gibt. Der folgende Algorithmus 23 zeigt die empfohlene Wahl des jeweiligen Verfahrens.

```
IF Anfrage = einfach und spaltenorientiert
        THEN Slicing

ELSEIF Anfrage = einfach und zeilenorientiert
        AND IF Anzahl der Zeilen klein
                THEN (alpha,k)-Anonymitaet
        ELSE SABRE or V-MDAV

ELSEIF Anfrage = komplex und spaltenorientiert
        THEN Slicing

ELSEIF Anfrage = komplex und zeilenorientiert
                AND IF Anzahl der Zeilen klein
                THEN (alpha,k)-Anonymitaet
        ELSE THEN SABRE or V-MDAV

ELSEIF Anfrage enthaelt join ueber mehrere Datenbanken
        THEN D2Pt (Verteiltes t-closeness)

ELSEIF Anfrage enthaelt join ueber eine Datenbanken
        THEN SABRE or V-MDAV
```

Abbildung 23: Entscheidungsalgorithmus

Der Entscheidungsalgorithmus 23 sieht vor bei spaltenorientierten, einfachen Anfragen Slicing zu nutzen. Bei der Selektion mehrerer Spalten kann das Slicing die Werte in den Spalten randomisiert vertauschen. Dies gilt selbst bei geringer Spaltenzahl, da selbst bei nur einer Spalte mit Quasi-Identifikatoren die Vertauschung ihrer Werte gemacht werden kann und ausreichend ist zur Anonymisierung. Das gleiche gilt für spaltenorientierte Anfragen die komplex sind. Unabhängig von der Anfrage ist nur wichtig, dass Spalten eines Datensatzes zurückgeben werden. Problematisch wird es, wenn die Daten in den Spalten homogen sind. Dann sollte ein Anonymisierungsverfahren gewählt werden, welche Daten unterdrückt um die Identifizierung der Zusammenhänge der Daten zu verhindern. Dann werden allerdings Informationen verloren gehen.
Bei einer einfachen, zeilenorientierten Anfrage mit einer kleinen Anzahl an Tupel bei der Rückgabe sollte die (α,k)-Anonymität gewählt werden. Die (α,k)-Anonymität liefert generalisierte Daten zurück, die vor Homogenitätsattacken sicher sind. Bei Anfragen mit einer großen Anzahl an Tupeln, die zurückgegeben werden, sollte die (α,k)-Anonymität nicht gewählt werden, da die Zeitkomplexität mit $O(p * n * m * log(m))$ einen ungünstigen Zeitverlauf hat. Eine einfache,

zeilenorientierte Anfrage mit einer möglicherweise großen Anzahl an Rückgabetupeln, sollte mittels t-Closeness und seinem SABRE Algorithmus anonymisiert werden. Alternativ ist auch das l-Diversity Verfahren mit dem V-MDAV-Algorithmus zu empfehlen. Die Zeitkomplexität der Algorithmen ist auch für große Tupelmengen geeignet. Gleiches gilt für komplexe, zeilenorientierte Anfragen.
Anfragen die einen oder mehrere Joins über Relationen aus mehreren Datenbanken enthalten, sollten mittels dem verteilten t-Closeness mit dem D2Pt bearbeitet werden. Die verteilte k-Anonymität hat, wie die k-Anonymität selbst, das Problem der Homogenitätsattacken. Das verteilte t-Closeness verhindert dies und ist für Datensätze, die physisch voneinander getrennt sind geeignet.
Sind die Anfragen mit einem oder mehreren Joins über eine Datenbank ausgestattet, dann sollte bei der resultierenden Tabelle das t-Closeness oder l-Diversity verwendet werden. Da ein Join zeilenorientiert ist, wird der SABRE- bzw. alternativ der V-MDAV-Algorithmus empfohlen.
Die (α,k)-Anonymität und der Multiplicative Weights Exponential Mechanism sind aufgrund ihrer Zeitkomplexität nur bedingt einsetzbar.

6 Evaluation

6.1 Messung von Informationsverlust mittels Kullback-Leibler-Divergenz

Durch die Anonymisierung werden Attributwerte unterdrückt bzw. generalisiert. Zur Bewertung der Anonymisierungsalgorithmen kann der Informationsverlust der neu ergebenen Daten gemessen werden. Dies ist mit der Kullback-Leibler-Divergenz möglich [HS10].

Definition 6.1 (Kullback Leibler Divergence)**.** Sei $D' = (A'_1, ..., A'_n, O, C)$ der originale und $D'' = (A''_1, ..., A''_n, O, C)$ der aus D' generalisierte Datensatz. Seien A'_i und A''_i die Attribute eines Quasi-Identifikators, O sind weitere Attribute und C sind Annotationen in D' und D''.

$$D(A'_i||A''_i) = \sum_{j}^{n_i} freq(a'_{ij}) log_2 \frac{freq(a'_{ij})}{freq(a''_{ij})}$$

$$= -\sum_{j}^{n_i} freq(a'_{ij}) log_2 freq(a''_{ij}) + \sum_{j}^{n_i} freq(a'_{ij}) log_2 freq(a'_{ij}) \text{[LLBW11]}$$

.

Mittels dieser Definition lässt sich der Informationsverlust für ein Attribut messen. Durch die Summe über alle Quasi-Identifikatorattribute lässt sich feststellen wie stark sich der Datenbestand verändert hat. Da die Kullback-Leibler-Divergenzen von Attribut zu Attribut variieren, müssen diese normalisiert werden um Vergleichbarkeit zu gewährleisten.

Definition 6.2 (Kullback Leibler Divergence mit Entropie)**.** Die Summe des normalisierten Informationsverlustes über alle A_i ist definiert als:

$$\sum_{j=1}^{n} D_N(A'_i||A''_i)^2$$

$$D_N(A'_i||A''_i) = \frac{D(A'_i||A''_i)}{H(A'_i)}$$

wobei

$$H(A'_i) = \sum_{j}^{n_i} freq(a'_{ij}) log_2 freq(a''_{ij}) \text{[LLBW11]}$$

die Entropie des Attributs A'_i ist.

Die Normalisierung ist nötig, da die Anzahl der Attributwerte ungleich sein kann zwischen dem Originaldatenbestand und dem aus ihr generalisierten Datensatz. $D_N(A'_i||A''_i)$ ist eine Zahl zwischen 0 und 1. Ziel ist, dass $D_N(A'_i||A''_i) \leq \delta_D$. δ_D ist die maximal erlaubte Veränderung der Kullback-Leibler-Divergenz und wird als kleine positive Zahl fest gesetzt. Zur Verdeutlichung soll ein Beispiel aus der Literatur angeführt werden [LLBW11]. Die Abbildung 24 zeigt zwei Datenbestände. Der linke Datenbestand ist eine generalisierte Tabelle und der rechte eine Tabelle mit unterdrückten Datensatz.

A'_1	A'_2	$A'_3 \dots A'_m$	Class
[1,4]	M	...	y
[1,4]	M	...	y
[1,4]	M	...	y
[1,4]	F	...	y
[5,8]	F	...	n
[5,8]	F	...	n
[5,8]	F	...	y
[5,8]	F	...	n

A''_1	A''_2	$A''_3 \dots A''_m$	Class
[1,4]	M	...	y
[1,4]	M	...	y
[1,4]	M	...	y
*	*	...	y
[5,8]	F	...	n
[5,8]	F	...	n
[5,8]	F	...	y
[5,8]	F	...	n

Abbildung 24: Beispiel generalisierter Datensatz und Datensatz mit Unterdrückung[LLBW11]

Werte	$freq(a'_{1j})$	$freq()a''_{1j})$
[1,4]	$4/8 = 0.5$	$3/8 = 0.375$
[5,8]	$4/8 = 0.5$	$4/8 = 0.5$
*	0	$1/8 = 0.125$

Tabelle 9: Häufigkeit Attributwerte [LLBW11]

Zur Berechnung von $D(A'_1||A''_1)$ wird nun die Häufigkeit der der Attributwerte A'_1 und A''_1 ermittelt.
Daraus ergibt zur Berechnung:

$$D(A'_1||A''_1) = 0.5 * log_2(\frac{0.5}{0.375}) + 0.5 * log_2(\frac{0.5}{0.5}) = 0.21$$

$$D_N(A'_1||A''_1) = \frac{D(A'_i||A''_i)}{H(A'_i)} = 0.21$$

Die Berechnung von $D_N(A'_i||A''_i)$ auf generalisierten bzw. anonymisierten Datenbeständen, welche durch verschiedene Verfahren erstellt wurden, ermöglicht die Vergleichbarkeit der Qualität der erzeugten Daten. Je größer $D_N(A'_i||A''_i)$ ist, desto mehr Informationsgehalt ging verloren. Hierbei kann noch die Transitivität von D_N erwähnt werden. Falls ein Datensatz D' zu D'' und dann noch zu D''' generalisiert wurde, kann gleich $D_N(A'_i||A'''_i)$ berechnet werden. Es gilt:

$$D_N(A'_i||A''_i) + D_N(A''_i||A'''_i) = D_N(A'_i||A'''_i).$$

Weitere Möglichkeiten der Evaluation sind die Messung der Zeit die für das Anonymisieren gebraucht wird und das Messen des genutzten Speicherplatzes. Bei der Zeitmessung kann die einfache, in Java vordefinierte Methode $System.currentTimeMillis()$ genutzt werden.

[2]Persönliche Mitteilung Hannes Grunert vom 21.06.2016

6.2 Evaluation eines Prototypen für (α, k)-Anonymität

Mittels Generalisierung erreicht der Prototyp das Anonymisieren von Daten. Als Ausgangsdaten werden 32561 Datensätze von einem Zensus genutzt.[3] Die Attribute sind: *age, fnlwgt, education, educationNum, materialStatus, occupation, relationship, race, sex, capitalGain, capitalLoss, hoursPerWeek, nativeCountry* und *income*. Die Attribute bestehen aus folgenden Attributwerten:

age: fortlaufender Wert von 0-100;

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked;

fnlwgt: fortlaufender Gewichtungswert von 0-1500000;

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool;

education-num: fortlaufender Wert von 1-16 je nach Ausbildungsgrad;

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse;

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces;

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried;

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black;

sex: Female, Male;

capital-gain: fortlaufender Wert für Vermögenszuwachs von 0-99999;

capital-loss: fortlaufender Wert für Vermögensverringerung von 0-3770;

hours-per-week: fortlaufender Wert für Arbeitsstuden pro Woche von 0-99;

native-country: Ursprungsland der Person, United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong,

[3] http://archive.ics.uci.edu/ml/datasets/Adult

Holand-Netherlands;

income: jährliches Einkommen der Person mit >50 und <=50K als Klassifizierungsattribut.

Die Funktionsweise des Prototypen wird am Beispiel erklärt. Im ersten Testdurchlauf wurde als Quasi-Identifikator QI=(age, fnlwgt, education) gewählt. Die Menge an allen Quasi-Identifikatoren in der Datenmenge ist bereits vorgegeben[4]. Als sensitives Attribut wurde *income* gewählt, welches das mögliche Jahreseinkommen anzeigt ($\leq 50k, > 50k$). Das α wurde auf 0.5 gesetzt und k=2.
Gemäß dem progressive local recoding Algorithmus wurde die Tabelle auf (α, k)-Anonymität geprüft. Hierfür wurde *age, fnlwgt, education* als erster Quasi-Identifikator genutzt und gleichzeitig damit die Äquivalenzklassen gesetzt. Für jeden Datensatz wurde geprüft, ob das jeweilige *income* in seiner Äquivalenzklasse nicht öfter als α vorkommt. Datensätze, die dieses Kriterium erfüllen, kommen in die Ergebnistabelle in die Datenbank. Wenn es öfter als α vorkommt, werden die Datensätze in eine andere Tabelle geschrieben, in der sie bearbeitet werden. In dieser Bearbeitungstabelle erfolgt dann die Generalisierung des *fnlwgt*. Dabei wird das *fnlwgt* in eine Generalisierungshierarchie gemäß Abbildung 25 eingeordnet.

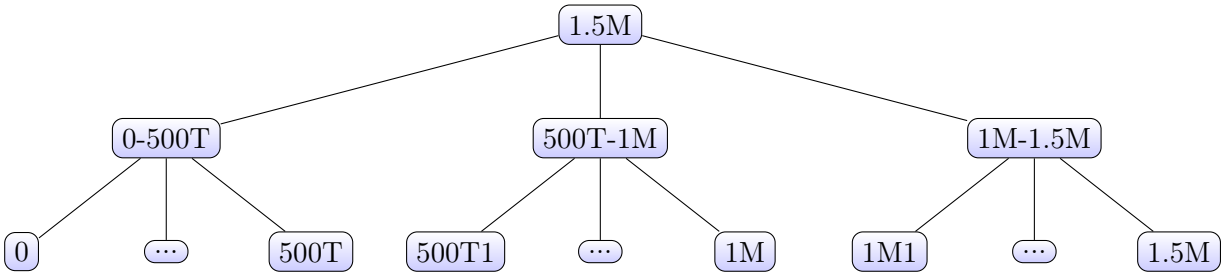


Abbildung 25: Generalisierung fnlwgt

Es werden bei den ersten Generalisierungen nur die mittlere Ebene der dargestellten Bäume erreicht. Bei dieser Generalisierung werden die Werte zu den jeweils höchsten Werten in ihrer Klasse zugeordnet und überschrieben. Nach dieser Generalisierung wird in den neuen Äquivalenzklassen geprüft, ob α als Grenzwert überschritten wurde oder nicht. Die Daten werden auf die jeweiligen Tabellen verteilt. In der Bearbeitungstabelle werden die Datensätze bezüglich des Attributes *education* generalisiert. Die Abbildung 26 zeigt die Struktur.
Es werden die einzelnen Abschlüsse als *graduate* deklariert und Schulbesuche ohne Schulabschluss mit *school* verallgemeinert Nach Prüfung der Äquivalenzklassen in Bezug auf α als Kriterium und darauffolgender Verteilung der Datensätze auf die jeweiligen Tabellen folgt die Generalisierung des Alters. Die Attributwerte von *age* werden gemäß der Struktur in Abbildung 27 generalisiert. Nachdem die Äquivalenzklassen geprüft und die Datensätze verteilt wurden, werden die vorher beschriebenen Generaliserungsschritte mit den höchsten Generaliserungsebenen wiederholt. Als letztes werden die übrigen Tupel, die (α,k) nach jeder Generaliserung immer noch nicht erfüllen,

[4]Identifiziert wurden sie durch einen Algorithmus von Hannes Grunert [GH14]

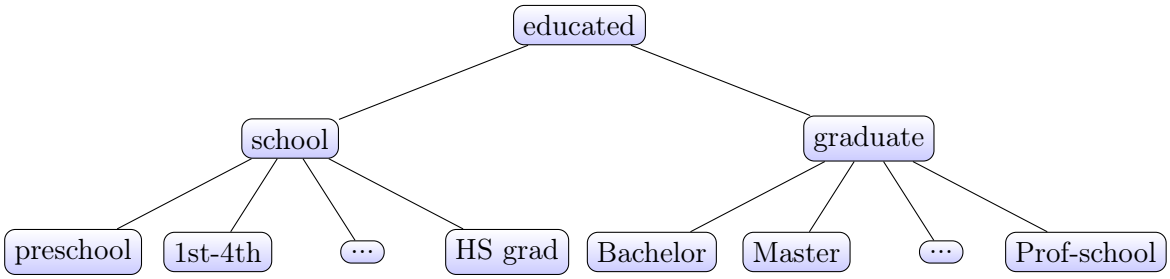


Abbildung 26: Generalisierung education

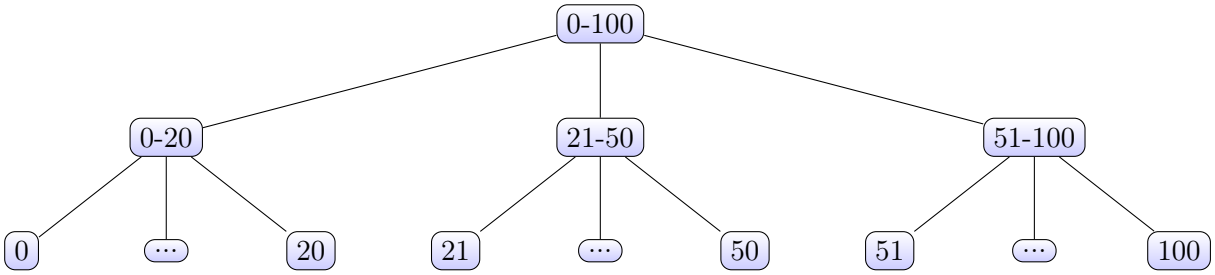


Abbildung 27: Generalisierung age

durch das Ersetzen der Attributwerte des Quasi-Identifikators mit „***“ anonymisiert und alle Werte hinzugefügt.
Die Speicherplatzkomplexität beträgt O(n), da die Gesamtzahl an Tupeln lediglich auf verschiedene Datenbanktabellen verteilt wird. Verdoppelt sich die Eingabe, dann verdoppelt sich der Speicherplatzbedarf. Die folgenden Abbildungen 28 und 29 zeigen den Pseudocode des Prototypen. Der gesamte Code wird ins Git Repository[5] hochgeladen.
Die Anfragen selektieren unterschiedliche Tupelmengen und danach wird der (α, k)-Anonymity-Prototyp ausgeführt. Die erste Anfrage 30 wählt alle Tupel aus, die beim Attribut Alter einen Wert zwischen 30 und 50 haben. Dies ergibt eine Tupelzahl von insgesamt 16390.
Die Anfrage 31 wählt alle Tupel aus, deren *age* kleiner als der Durchschnittswert von allen *age* ist. Dies ergibt eine Tupelzahl von insgesamt 17508.
Als weitere Anfrage 32 werden alle Tupel ausgewählt, deren Attributwert *age* kleiner als der Durchschnitt von *age* ist. Dies ergibt eine Tupelzahl von insgesamt 395.
Die Anfrage 33 wählt alle Daten aus, deren *educationNum* den Höchstwert haben. Dies ergibt eine Tupelzahl von insgesamt 413.
Die Anfrage 34 wählt alle Daten aus, die beim Attribut Alter einen Wert zwischen 30 und 35 haben. Dies ergibt eine Tupelzahl von insgesamt 5214.
Die obigen Anfragen wurden für das sensitive Attribut *income* und *occupation* ausgeführt. Das α wurde auf 0.5 gesetzt und k=2. Es sollte getestet werden, welche Auswirkungen die Anzahl der Werte des sensitiven Attributs auf die Anzahl der unterdrückten Tupel und den Zeitaufwand hat. Für α wurde 0.5 gewählt, k=2 und QI=age. Es wurde einerseits gemessen wie lange der Algorithmus braucht und bei wie vielen Tupel die Informationen in Bezug auf den Quasi-Identifikator

[5] https://git.informatik.uni-rostock.de/hg/PArADISE

```
public void check_alphakano1(int k){
rs = stmt.executeQuery("SELECT age,education, fnlwgt, COUNT(*) "
        +"From adult GROUP BY age, education, fnlwgt ORDER BY age");
while(rs.next()){
        rs_erg = stmt_erg.executeQuery("SELECT income, COUNT(*) "
                +"FROM table WHERE QI=rs.getString");
        int count_age = rs.getInt(4);
        int count_sens=rs_erg.getInt(2);
        if(count_age < k){
                insert_table(alles in table_work);
        }else{
                double alpha = count_sens/count_age;
                if(alpha <= 0.5){
                        sql = "Insert in table_ergebnis";
                        q++;
                }
                rs2 = stmt.executeQuery("Select distinct income"
                        +"From table WHERE QI=rs.getString"
                        +" and income != rs_erg.getString(1)+");
                        while(rs2.next()){
                                rs3 = stmt3.executeQuery("SELECT *
                                        +"FROM adult WHERE"
                                        +"QI=rs.getString"
                                        +"and race =rs2.getString");
                                int i=0;
                                while(rs3.next()){
                                  if(i<q){
                                        sql = "Insert in tmp_adult";
                                        i++;
                                  }else{
                                        sql ="Insert in tmp_adult_work";
                                        }
                                }
                        }
                }else{
                        insert_table(alles in table_work);
                }
        }
}
```

Abbildung 28: Code Prototyp

```
public void check_alphakano2(int k) throws SQLException{
        sql3 = "UPDATE tmp_adult_work SET fnlwgt= (CASE WHEN"
        +"fnlwgt<=500000 THEN '500000' WHEN "
        +"age BETWEEN 500001 AND 1000000 THEN '1000000' "
        +"ELSE  '1500000' END)";

        repeat steps from check_alphakano1() with new data
        ...
}

//repeat check_alphakano() till DGHs arrived
```

Abbildung 29: Code Prototyp

```
SELECT *
FROM adult
WHERE age between 30 and 50
```

Abbildung 30: erste Anfrage für den Prototypen

```
SELECT *
FROM adult
WHERE age <
(SELECT avg (age) FROM adult)
```

Abbildung 31: zweite Anfrage für den Prototypen

```
SELECT *
FROM adult
WHERE age =
(SELECT min (age) FROM adult)
```

Abbildung 32: dritte Anfrage für den Prototypen

```
SELECT *
FROM adult
WHERE educationNum =
(SELECT max (educationNum)
FROM adult)
```

Abbildung 33: vierte Anfrage für den Prototypen

```
SELECT *
FROM adult
WHERE age between 30 and 35
```

Abbildung 34: fünfte Anfrage für den Prototypen

unterdrückt wurden. Hierbei wurde allerdings nicht die Kullback-Leibler Divergenz verwendet, sondern lediglich die Tupel gezählt, deren Quasi-Identifikator nach der höchsten Generalisierung unterdrückt wurden.
Die Abbildung 35 zeigt die Ergebnisse der Tupelunterdrückung. Die Messung ergab, dass eine höhere Anzahl an Werten eines sensitiven Attributs nicht immer zu weniger unterdrückten Tupeln führt. Bei 395 Tupeln und der Anfrage 32 wurden bei *income* als sensitives Attribut alle Datensätze unterdrückt, jedoch bei *occupation* 358 Tupel unterdrückt. Bei allen anderen Messungen konnte mit *income* eine höhere Anzahl an Tupel generalisiert werden. Vermutlich ist nicht nur die Anzahl der Attributwerte des sensitiven Attributs ein Faktor damit möglichst viele Tupel (α,k)-Anonymität erfüllen, sondern auch die Verteilung der Attributwerte im Datensatz.
Die Abbildung 36 zeigt die Entwicklung des Zeitaufwandes. Die Entwicklung der Kurve lässt auf linearen Zeitverlauf schließen, statt auf exponentiellen. Die benötigte Zeit ist bei *occupation* als sensitives Attribut fast immer höher als bei *income*. Lediglich einmal bei 413 Tupeln war der Prototyp mit *occupation* schneller fertig. Dies könnte den Schluss zulassen, dass je höher die Anzahl der sensitiven Attributwerte, desto höher der Zeitbedarf. Allerdings muss beachtet werden, dass die weiter zu generalisierenden Tupel in die jeweilige Tabelle dafür geschrieben werden. Das heißt, je weniger Tupel in die Ergebnistabelle kommen, desto mehr Tupel werden in die Tabelle der weiter zu generalisierenden Tupel geschrieben. Dies geschieht bei jeder Generalisierung. Durch eine höhere Anzahl zu generalisierender Tupel bei *occupation* müssen mehr Tupel immer wieder in die Tabelle der weiter zu generalisierenden Tupel geschrieben werden. Dies nimmt immer wieder Zeit in Anspruch. Da in Abbildung 24 stellenweise wesentlich mehr Tupel unterdrückt wurden, liegt die Vermutung nahe, das immer wieder eine höhere Anzahl an Tupel bis zur Unterdrückung in die Tabellen geschrieben wurden. Das wiederum kann der Grund für den höheren Zeitbedarf sein.
Als Weiteres soll die Auswirkung der Größe des Quasi-Identifikators auf den Informationsverlust und den Zeitaufwand betrachtet werden. Hierfür wurde QI=(age, fnlwgt, education), QI=(age, education) und QI=(age) und für alle *income* als sensitives Attribut gewählt. Es wurden gesonderte Anfragen verwendet, welche im Anhang 8 aufgeführt sind. Die Abbildung 37 zeigt das Ergebnis. Bei allen Messungen mit allen Quasi-Identifikatoren wurde immer die gleiche Anzahl an Tupel unterdrückt. Mit diesem Ergebnis scheint die Laufzeit unabhängig von der Anzahl an Attributen des Quasi-Identifikators zu sein. Die Abbildung 38 zeigt jedoch das Ergebnis einer zweiten Messung. Bei dieser Messung wurde *race* als sensitives Attribut gewählt und die Quasi-Identifikatoren blieben die gleichen. Die zweite Messung wurde lediglich mit 176 und 2795 Tupeln gemacht. Es zeigt sich, dass QI=(age) die höchste Zahl an Tupeln unterdrückt. Bei 176 Tupeln haben QI=(age, fnlwgt, education) und QI=(age, education) die gleiche Anzahl an unterdrückten Tupeln und bei 2795 Tupeln ist QI=(age, education) sogar minimal besser beim Informationserhalt. Durch die zweite Messung wird klar, dass die Anzahl der Attribute eines Quasi-Identifikators

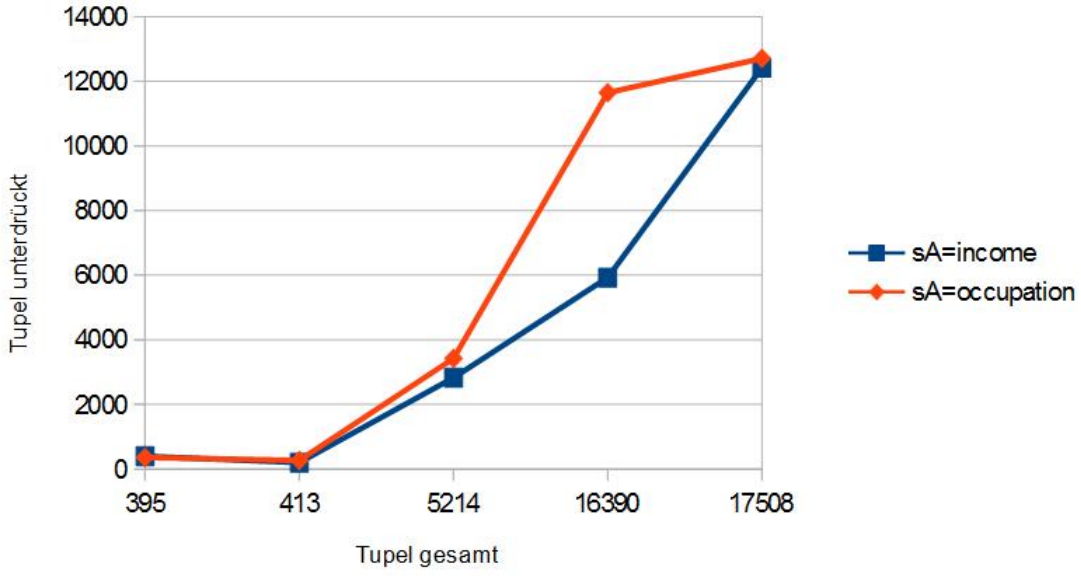


Abbildung 35: Entwicklung der Anzahl unterdrückter Tupel für *income* und *occupation*

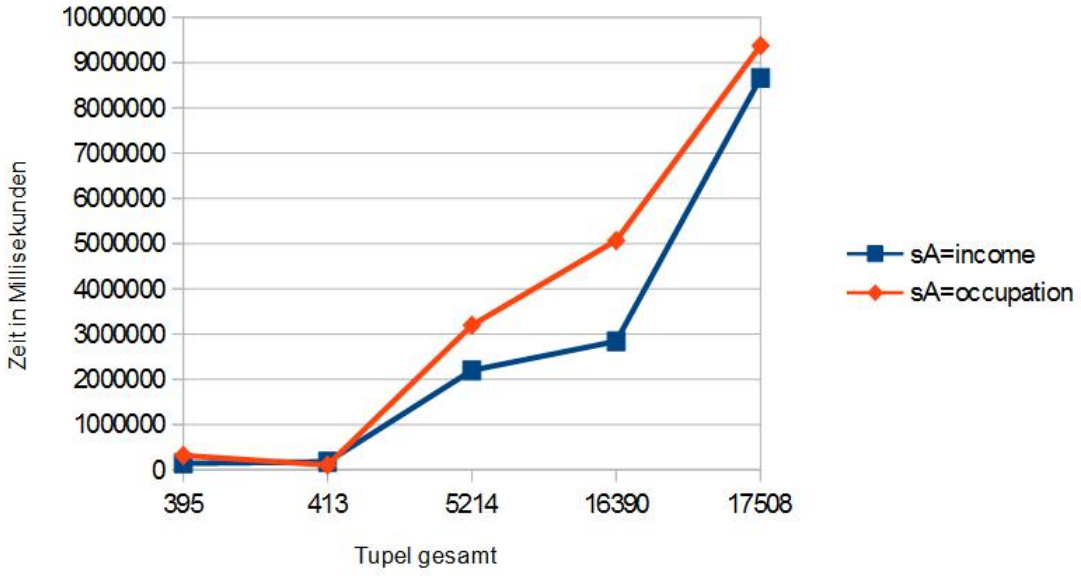


Abbildung 36: Entwicklung des Zeitaufwandes für *income* und *occupation*

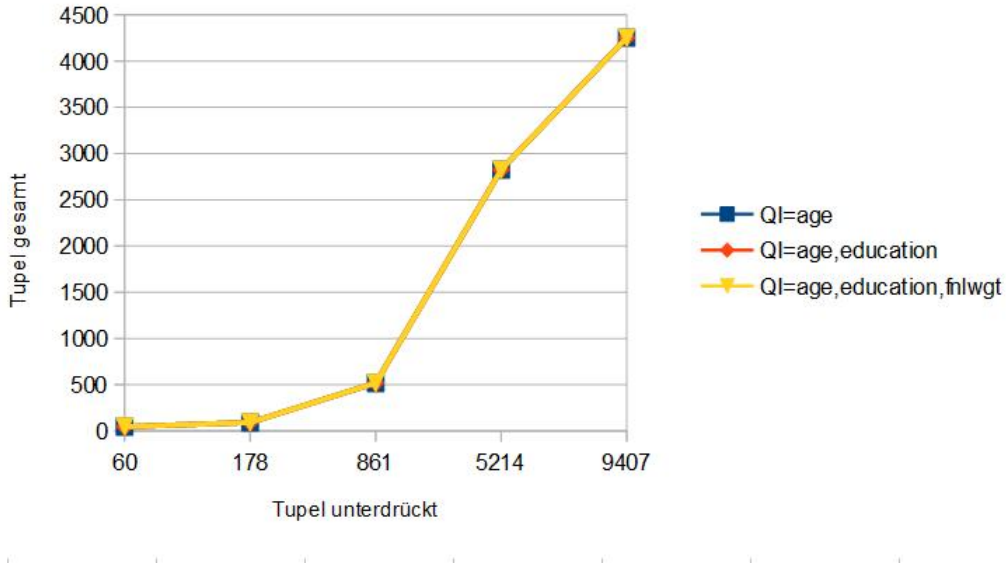


Abbildung 37: Entwicklung der Anzahl unterdrückter Tupel in Bezug auf Anzahl Attributwerte QI für sensitives Attribut *income*

durchaus Auswirkungen auf den Informationsverlust hat, aber auch die Verteilung der sensitiven Daten ausschlaggebend ist.
Bei dem Zeitaufwand in Bezug auf die verschiedenen Quasi-Identifikatoren ist eindeutig festzustellen, dass je größer die Anzahl an Attributen bei den Quasi-Identifikatoren ist, desto höher der Zeitbedarf ist. In der Abbildung 39 ist ein großer zeitlicher Sprung von QI=(age, education) zu QI=(age, fnlwgt, education) zu sehen. In der zweiten Messung mit *race* als sensitives Attribut bestätigte sich dies.

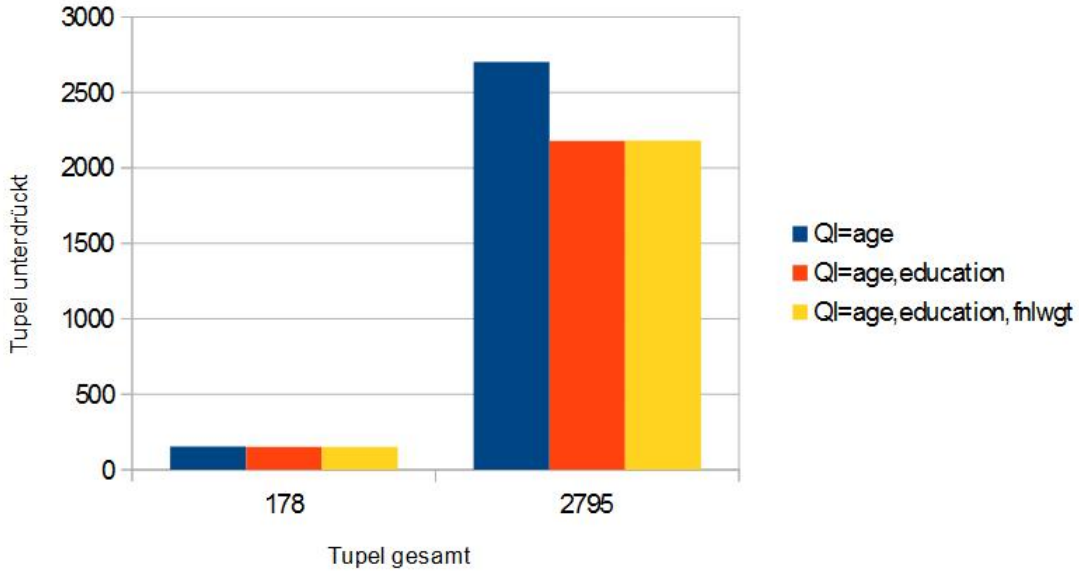


Abbildung 38: Entwicklung der Anzahl unterdrückter Tupel in Bezug auf Anzahl Attributwerte QI für sensitives Attribut *race*

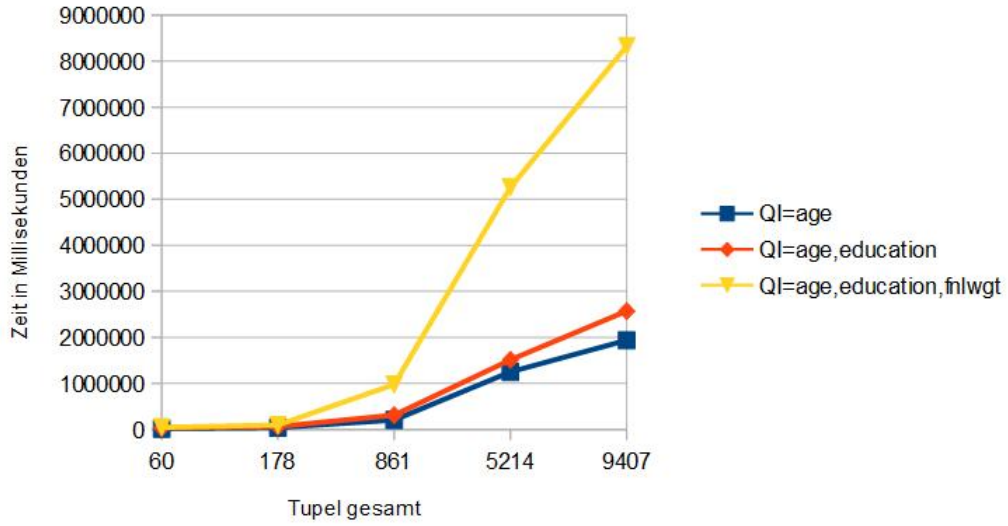


Abbildung 39: Entwicklung des Zeitaufwandes für Anzahl Attributwerte QI für sensitives Attribut *income*

7 Zusammenfassung

7.1 Zusammenfassung

Diese Arbeit beschreibt detailliert die Datenschutzalgorithmen zur Anonymisierung von Daten. Diese sind k-Anonymität, l-Diversity, t-Closeness, Slicing und Differential Privacy. Es wurde die Komplexität aller Verfahren untersucht. Da die optimale Ergebnislösung der meisten Verfahren NP-schwer ist, wurden Approximationsalgorithmen hinzugezogen und ihre Komplexität untersucht. Diese sind (α, k)-Anonymität, Distributed Privacy-Preserving two-Party Generic Anonymizer (DPP_2GA) für verteilte k-Anonymität, Distributed two-Party t-Closeness (D2Pt) für verteiltes t-Closeness, SABRE (Sensitive Attribute Bucketization Redistribution) für t-Closeness, V-MDAV Algorithmus für l-Diversity und den Multiplicative Weights Exponential Mechanism für Differential Privacy. Aus dieser Betrachtung wurde dann ein Entscheidungsalgorithmus generiert. Der Entscheidungsalgorithmus ist eine Empfehlung für jeweilige mögliche vorangehende SQL-Anfragearten auf Datenbanken bzw. die zu anonymisierenden Daten. Je nach Art der Anfrage wird ein Datenschutzalgorithmus bzw. sein Approximationsalgorithmus empfohlen.
Es wurde anschließend ein Prototyp für die (α, k)-Anonymität implementiert. Es wurden Messungen in Bezug auf Zeitaufwand und Informationsverlust durchgeführt. Der Informationsverlust wurde durch die Anzahl der unterdrückten Tupel ermittelt. Realisiert wurde dies durch die Unterdrückung bei den Attributwerten der Quasi-Identifikatoren. Als Datenbasis diente ein Zensusdatensatz mit über 32 tausend Daten.
Als Ergebnis der Messungen konnte festgestellt werden, dass je größer die Anzahl der Attributwerte der Quasi-Identifikatoren, desto mehr Zeit benötigt der Algorithmus zur Vollendung. Es konnte nicht festgestellt werden, dass die größere Anzahl der Attributwerte der Quasi-Identifikatoren auch weniger Tupel unterdrückt. Bei einer Messung kam es zu gar keiner Veränderung und bei einer weiteren mit einem anderen sensitiven Attribut konnte zum Teil eine deutlich geringere Anzahl an Tupel unterdrückt werden. Dementsprechend kann gesagt werden, dass eine höhere Anzahl der Attributwerte der Quasi-Identifikatoren einen geringeren Informationsverlust bei der Anonymisierung zur Folge haben kann. Allerdings hängt dies auch stark an der Verteilung der Attributwerte des sensitiven Attributs und auch an (α) ab.
Eine weitere Messung sollte den Einfluss von der Anzahl sensitiver Attributwerte auf den Informationsverlust und den Zeitaufwand ermitteln. Ergebnis der Messung ist, dass je höher die Anzahl der sensitiven Attributwerte, desto mehr Zeit wird benötigt. Jedoch konnte der Informationsverlust mit mehr sensitiven Attributwerten nicht durchgehend verbessert werden, sondern er wurde mit steigender Anzahl an Tupeln schlechter. Lediglich bei der ersten Teilmessung mit 395 Tupel waren mehr sensitive Attributwerte besser für einen geringen Informationsverlust.

7.2 Ausblick

Von dieser Arbeit ausgehend können weitere Algorithmen für die Verfahren implementiert und untersucht werden. Der Entscheidungsalgorithmus könnte mit diesen zusätzlichen Algorithmen verfeinert werden. In dieser Arbeit gab es Messergebnisse, bei denen die Informationsunterdrückung bei verschiedener Anzahl an Attributwerten von Quasi-Identifikatoren gleich war. Es gab auch Messergebnisse, bei denen weniger Tupel unterdrückt wurden bei höherer Anzahl Attributwerten von Quasi-Identifikatoren. Mit dem implementierten Prototypen können weitere

Messungen der Informationsunterdrückung in Bezug auf Anzahl der Quasi-Identifikatoren und verschiedene sensitive Attribute gemacht werden.

Literatur

[Beu15] Beuth, Patrick: *Nicht vor dem Fernseher, Schatz.* http://www.zeit.de/digital/datenschutz/2015-02/samsung-smart-tv-private-gespraeche, 02 2015. Zugriff 19.04.2016.

[BRD] BRD, Justizministerium: *Bundesdatenschutzgesetz (BDSG) § 3a Datenvermeidung und Datensparsamkeit.* http://www.gesetze-im-internet.de/bdsg_1990/_3a.html. Zugriff 19.04.2016.

[CKKT11] Cao, Jianneng, Panagiotis Karras, Panos Kalnis und Kian-Lee Tan: *SABRE: a Sensitive Attribute Bucketization and REdistribution framework for t-closeness.* The VLDB Journal, 20(1):59–81, 2011.

[Dal86] Daleniusl, Tore: *Finding a needle in a haystack.* Journal of official statistics, 2(3):329–336, 1986.

[DL09] Dwork, Cynthia und Jing Lei: *Differential privacy and robust statistics.* In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*, Seiten 371–380. ACM, 2009.

[DMZ11] Dondi, Riccardo, Giancarlo Mauri und Italo Zoppis: *On the Complexity of the l-diversity Problem.* In: *Mathematical Foundations of Computer Science 2011*, Seiten 266–277. Springer, 2011.

[DR14] Dwork, Cynthia und Aaron Roth: *The algorithmic foundations of differential privacy.* Foundations and Trends in Theoretical Computer Science, 9(3-4):211–407, 2014.

[Dwo] Dwork, Cynthia: *Differential Privacy.* http://www.msr-waypoint.com/pubs/64346/dwork.pdf. Microsoft Research.

[GH14] Grunert, Hannes und Andreas Heuer: *Big Data und der Fluch der Dimensionalität: Die effiziente Suche nach Quasi-Identifikatoren in hochdimensionalen Daten.* In: Klan, Friederike, Günther Specht und Hans Gamper (Herausgeber): *Proceedings of the 26th GI-Workshop Grundlagen von Datenbanken, Bozen-Bolzano, Italy, October 21st to 24th, 2014.*, Band 1313 der Reihe *CEUR Workshop Proceedings*, Seiten 29–34. CEUR-WS.org, 2014.

[GH15] Grunert, Hannes und Andreas Heuer: *Slicing in Assistenzsystemen: Wie trotz Anonymisierung von Daten wertvolle Analyseergebnisse gewonnen werden können.* In: *Grundlagen von Datenbanken*, 2015.

[GH16] Grunert, Hannes und Andreas Heuer: *Datenschutz im PArADISE.* Datenbank-Spektrum, 16(2):107–117, Juli 2016.

[Gul14] Gulde, Dirk: *Auf dem Weg zum gläsernen Autofahrer.* http://www.auto-motor-und-sport.de/fahrberichte/datensicherheit-im-auto-auf-dem-weg-zum-glaesernen-autofahrer-8314046.html, 07 2014. Zugriff 19.04.2016.

[Hau07] Hauf, Dietmar: *Allgemeine Konzepte K-Anonymity, l-Diversity and T-Closeness.* IPD Uni-Karlsruhe. http://dbis. ipd. uni-karlsruhe. de/img/content/SS07Hauf_kAnonym. pdf, 2007. Zugriff 26.05.2016.

[HF06] Harel, David und Yishai Feldman: *Algorithmik: die Kunst des Rechnens.* Springer-Verlag, 2006.

[HLM12] Hardt, Moritz, Katrina Ligett und Frank McSherry: *A simple and practical algorithm for differentially private data release.* In: *Advances in Neural Information Processing Systems*, Seiten 2339–2347, 2012.

[HM15] Heuer, Andreas und Holger Meyer: *Das PArADISE-Projekt-Big-Data-Analysen für die Entwicklung von Assistenzsystemen.* Grundlagen von Datenbanken, Seiten 102–103, 2015.

[Hof13] Hoffmann, Felix: *Smart-TV zieht Nutzerdaten – abschalten unmöglich.* http://www.welt.de/wirtschaft/webwelt/article122169667/Smart-TV-zieht-Nutzerdaten-abschalten-unmoeglich.html, 11 2013. Zugriff 19.04.2016.

[Hof15] Hoffmann, Dirk: *Theoretische Informatik.* Carl Hanser Verlag GmbH Co KG, 2015.

[HS10] Hintoglu, Ayça Azgin und Yücel Saygin: *Suppressing microdata to prevent classification based inference.* Band 19, Seiten 385–410. Springer, 2010.

[JmTtHq08] Jian-min, Han, Cen Ting-ting und Yu Hui-qun: *An improved V-MDAV algorithm for l-diversity.* In: *Information Processing (ISIP), 2008 International Symposiums on*, Seiten 733–739. IEEE, 2008.

[LLBW11] Li, Jiuyong, Jixue Liu, Muzammil Baig und Raymond Chi-Wing Wong: *Information based data anonymization for classification utility.* Data & Knowledge Engineering, 70(12):1030–1045, 2011.

[LLV07] Li, Ninghui, Tiancheng Li und Suresh Venkatasubramanian: *t-closeness: Privacy beyond k-anonymity and l-diversity.* In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, Seiten 106–115. IEEE, 2007.

[LY13] Liang, Hongyu und Hao Yuan: *On the complexity of t-closeness anonymization and related problems.* In: *Database Systems for Advanced Applications*, Seiten 331–345. Springer, 2013.

[Mü16] Müller, Martin: *Technical Report CS-02-16 Generalisierung von Attributen in Relationalen Datenbanken.* Universität Rostock, Institut für Informatik, 2016. Zugriff 15.04.2016.

[MKGV07] Machanavajjhala, Ashwin, Daniel Kifer, Johannes Gehrke und Muthuramakrishnan Venkitasubramaniam: *l-diversity: Privacy beyond k-anonymity.* ACM Transactions on Knowledge Discovery from Data (TKDD), 1(1):3, 2007.

[NJTF15] Nielsen, Jan Hendrik, Daniel Janusz, Jochen Taeschner und Johann-Christoph Freytag: *D2Pt: Privacy-Aware Multiparty Data Publication.* In: *BTW*, Seiten 105–124, 2015.

[NKK13] Nguyen, Hiep H, Jong Kim und Yoonho Kim: *Differential privacy in practice.* Journal of Computing Science and Engineering, 7(3):177–186, 2013.

[NRS07] Nissim, Kobbi, Sofya Raskhodnikova und Adam Smith: *Smooth sensitivity and sampling in private data analysis.* In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, Seiten 75–84. ACM, 2007.

[NS15] Nissim, Kobbi und Uri Stemmer: *On the generalization properties of differential privacy.* arXiv preprint arXiv:1504.05800, 2015.

[Pos] PostgreSQL: *PostgreSQL 8.4.22 Documentation.* https://www.postgresql.org/docs/8.4/static/queries-with.html. Zugriff 13.06.2016.

[Sch16] Schmude, Maik: *Systematische Untersuchung der Anfragekapazität verschiedener Datenbankmanagementsysteme.* Bachelorarbeit, Universität Rostock, 2016. erscheint im Oktober 2016.

[Smi11] Smith, Adam: *Privacy-preserving statistical estimation with optimal convergence rates.* In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*, Seiten 813–822. ACM, 2011.

[SS98] Samarati, Pierangela und Latanya Sweeney: *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression.* Technischer Bericht, Technical report, SRI International, 1998.

[SSH10] Saake, Gunter, Kai-Uwe Sattler und Andreas Heuer: *Datenbanken: Konzepte und Sprachen.* mitp Verlags GmbH & Co. KG, 2010.

[Swe98] Sweeney, Latanya: *Datafly: A system for providing anonymity in medical data.* In: *Database Security XI*, Seiten 356–381. Springer, 1998.

[Swe02] Sweeney, Latanya: *k-anonymity: A model for protecting privacy.* International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10(05):557–570, 2002.

[WLFW09] Wong, Raymond, Jiuyong Li, Ada Fu und Ke Wang: *(α, k)-anonymous data publishing.* Journal of Intelligent Information Systems, 33(2):209–234, 2009.

[XYT10] Xiao, Xiaokui, Ke Yi und Yufei Tao: *The hardness and approximation algorithms for l-diversity.* In: *Proceedings of the 13th International Conference on Extending Database Technology*, Seiten 135–146. ACM, 2010.

8 Anhang

Als Testumgebung wurde eine MySQL-Datenbank auf einem Server genutzt. Eine virtuelle Maschine diente als Server. Diese war mit einer 64-Bit-CPU (QEMU Virtual CPU Version (cpu64-rhel6)) mit vier Kernen mit jeweils 2GHz und 4MB Cache und 4GB RAM ausgestattet. Der (α, k)-Prototyp wurde auf einem Laptop ausgeführt, der mittels VPN-Verbindung auf die Datenbank Zugriff hatte. Die Hardware des Laptop (Lenovo S210 Touch) bestand aus einem Intel Pentium CPU 2117U mit zweimal 1,8 GHz und vier GB RAM.

Tupel gesamt	**sA=income**	**sA=occupation**
395	395	358
413	199	266
5214	2826	3422
16390	5926	11651
17508	12424	12710

Tabelle 10: Ergebnisse Messung 1

Tupel gesamt	**sA=income**	**sA=occupation**
395	146339	319177
413	178623	106982
5214	2198809	3195774
16390	2839714	5068900
17508	8661769	9372923

Tabelle 11: Ergebnisse Messung 2

Tupel gesamt	**QI=age**	**QI=age,education**	**QI=age,education,fnlwgt**
60	46	46	46
178	92	92	92
861	519	519	519
5214	2826	2826	2826
9407	4253	4253	4253

Tabelle 12: Ergebnisse Messung 3

Tupel gesamt	QI=age	QI=age,education	QI=age,education,fnlwgt
60	18294	34611	46212
178	39105	63465	97244
861	205068	312615	977946
5214	1249540	1513186	5258268
9407	1938087	2572717	8326154

Tabelle 13: Ergebnisse Messung 4

Tupel gesamt	QI=age	QI=age,education	QI=age,education,fnlwgt
178	155	153	153
2795	2701	2179	2182

Tabelle 14: Ergebnisse Messung 5

```
SELECT *
FROM adult
WHERE age between 80 and 83
```

Abbildung 40: 60 Tupel

```
SELECT *
FROM adult
WHERE age=65
```

Abbildung 41: 178 Tupel

```
SELECT *
FROM adult
WHERE age=30
```

Abbildung 42: 861 Tupel

```
SELECT *
FROM adult
WHERE age between 30 and 35
```

Abbildung 43: 5214 Tupel

```
SELECT *
FROM adult
WHERE age between 30 and 40
```

Abbildung 44: 9407 Tupel