

**Gesellschaft
für Informatik**



**Proceedings of the 27th GI-Workshop
Grundlagen von Datenbanken**

Gommern, Deutschland, 26.-29. Mai 2015



©2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. Re-publication of material from this volume requires permission by the copyright owners.

Herausgeber:

Gunter Saake,
David Broneske,
Sebastian Dorok,
Andreas Meister

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme
Universitätsplatz 2
DE-39106 Magdeburg
E-Mail: vorname.nachname@ovgu.de

Vorwort

Liebe Teilnehmerinnen und Teilnehmer,

bereits zum 27. Mal fand vom 26.05.2015 bis 29.05.2015 der Workshop „Grundlagen von Datenbanken“ (GvDB) statt. Nachdem der Workshop im letzten Jahr in Südtirol zu Gast war, kehrte er in diesem Jahr wieder nach Deutschland zurück, genauer nach Gommern in Sachsen-Anhalt, wo er bereits das zweite Mal nach 2001 stattfand.

Der viertägige Workshop wurde vom GI-Arbeitskreis Grundlagen von Informationssystemen im Fachbereich Datenbanken und Informationssysteme (DBIS) veranstaltet und hat die theoretischen, konzeptionellen und methodischen Grundlagen von Datenbanken und Informationssystemen zum Thema, ist aber auch für neue Anwendungsgebiete mit Datenmanagementbezug offen. Organisiert wurde der Workshop durch die Arbeitsgruppe Datenbanken und Software Engineering der Otto-von-Guericke-Universität Magdeburg.

Der Workshop soll die Kommunikation zwischen Wissenschaftlern/-innen im deutschsprachigen Raum fördern, die sich grundlagenorientiert mit Datenbanken und Informationssystemen beschäftigen. Er ist insbesondere als Forum für Nachwuchswissenschaftler/-innen gedacht, die ihre aktuellen Arbeiten in einem größeren Forum vorstellen wollen. Der Workshop fand im idyllischen „Hotel am See“, dem Robinien-Hof in Gommern, statt. Das Hotel befindet sich gleich gegenüber der letzten Wanderdüne in Sachsen-Anhalt. Durch die ruhige Lage am „Kulk“ bietet der Tagungsort einen idealen Rahmen für offene und inspirierende Diskussionen zu Datenbanken und Informationssystemen.

Aus den Einsendungen wurden 15 Arbeiten nach einem Review-Prozess ausgewählt und vorgestellt. Die Themenvielfalt erstreckte sich dabei von Crowdsourcing für Entity Resolution über klassischere Themen wie Datenkompression bis hin zu Datenstrommanagementsystemen. Die Beiträge wurden durch drei Keynotes ergänzt. Kai-Uwe Sattler, Professor an der TU Ilmenau, ging in seinem Vortrag *Optimierung von Datenflussprogrammen - ein Fall klassischer Anfrageoptimierung?* auf die Optimierung von Ausführungsplänen zur Datenstromverarbeitung ein. Erhard Rahm, Professor an der Uni Leipzig, präsentierte in seinem Vortrag *Scalable Graph Analytics with GRADOOP* die Potentiale von MapReduce-Frameworks wie Hadoop für Graphanalysen. Außerdem beleuchtete Wolfgang Lehner von der TU Dresden in seinem Vortrag *Next-Generation Hardware for Data Management - More a Blessing than a Curse?* offene Fragen bei der Entwicklung hardware-sensitiver Datenbanksysteme und gab einen Ausblick auf die Potentiale, die sich aus dem konsequentem HW/SW-Datenbank-CoDesign ergeben.

Das ausgewogene Workshop-Programm wurde von zwei Ausflügen abgerundet. Zunächst konnten die Teilnehmer bei einer Führung durch den Gommeraner Gesteinsgarten eine der größten und umfassendsten Sammlungen von Natursteinen aus Deutschland und Europa erkunden und bei der anschließenden Waldwande-

rung die Ruhe der Natur genießen. Aber auch für die kulinarische Unterhaltung war gesorgt. So stand neben einem Grillbüffett zur Auffrischung der Kräfte nach den Wanderungen der Besuch von Deutschlands einziger Gasthausbrauerei auf einer über 1000 Jahre alten Burg an, natürlich inklusive Verkostung.

An dieser Stelle danke ich allen Beteiligten für die erfolgreiche Durchführung des GvDB 2015: den Autoren, dem Programmkomitee und den Mitarbeitern des Tagungshotels. Besonderer Dank gilt allen Mitgleidern des Organisations-Komitee ohne deren Hilfe die Durchführung des GvDB 2015 nicht möglich gewesen wäre: David Broneske, Sebastian Dorok, Andreas Meister, Siba Mohammad und Veit Köppen. Vielen Dank!

Gunter Saake

Magdeburg am 26.05.2015

Komitee

Organisationskomitee

David Broneske
Sebastian Dorok
Andreas Meister
Siba Mohammad
Veit Köppen
Gunter Saake

Programm-Komitee

Stefan Brass	Universität Halle
Erik Buchmann	Universität Karlsruhe
Stefan Conrad	Universität Düsseldorf
Rainer Gemulla	Universität Mannheim
Friederike Klan	Friedrich-Schiller Universität Jena
Holger Meyer	Universität Rostock
Klaus Meyer-Wegener	Universität Erlangen
Gunter Saake	Universität Magdeburg
Kai-Uwe Sattler	TU Ilmenau
Eike Schallehn	Universität Magdeburg
Ingo Schmitt	TU Cottbus
Holger Schwarz	Universität Stuttgart
Günther Specht	Universität Innsbruck
Jens Teubner	TU Dortmund

Inhaltsverzeichnis

Key Notes:	9
Optimierung von Datenflussprogrammen - ein Fall klassischer Anfrageoptimierung? <i>Kai-Uwe Sattler</i>	9
Scalable graph analytics with GRADOOP <i>Erhard Rahm</i>	10
Next-Generation Hardware for Data Management – more a Blessing than a Curse? <i>Wolfgang Lehner</i>	11
Workshopbeiträge:	12
Ontologie-basierte Fragmentierungs- und Replikationsverfahren für verteilte Datenbanksysteme <i>Lena Wiese</i>	12
Automated Silhouette Extraction for Mountain Recognition <i>Daniel Braun, Michael Singhof</i>	18
Slicing in Assistenzsystemen - Wie trotz Anonymisierung von Daten wertvolle Analyseergebnisse gewonnen werden können <i>Hannes Grunert, Andreas Heuer</i>	24
Towards Visualization Recommendation – A Semi- Automated Domain-Specific Learning Approach <i>Pawandeep Kaur, Michael Owonibi, Birgitta Koenig-Ries</i>	30
Transparente Datenbankunterstützung für Analysen auf Big Data <i>Dennis Marten, Andreas Heuer</i>	36
Ansätze zur Erkennung von Kommunikationsmodi in Online-Diskussionen <i>Matthias Liebeck</i>	42

Ausführungspläne und -planoperatoren relationaler Datenbankmanagementsysteme <i>Christoph Koch, Katharina Büchse</i>	48
Modularisierung leichtgewichtiger Kompressionsalgorithmen <i>Juliana Hildebrandt, Dirk Habich, Patrick Damme, Wolfgang Lehner</i>	54
Annotation und Management heterogener medizinischer Studienformulare <i>Victor Christen</i>	60
Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data <i>Muhammad Saqib Niaz, Gunter Saake</i>	66
Crowdsourcing Entity Resolution: a Short Overview and Open Issues <i>Xiao Chen</i>	72
Toward GPU Accelerated Data Stream Processing <i>Marcus Pinnecke, David Broneske, Gunter Saake</i>	78
Where- und Why-Provenance für syntaktisch reiches SQL durch Kombination von Programmanalysetechniken <i>Tobias Müller</i>	84
Large-scale Analysis of Event Data <i>Stefan Hagedorn, Kai-Uwe Sattler, Michael Gertz</i>	90
Flexible Online-Recommender-Systeme durch die Integration in ein Datenstrommanagementsystem <i>Cornelius A. Ludmann, Marco Grawunder, Hans-Jürgen Appelrath</i>	96
Extended Abstracts:	102
Das PArADISE-Projekt - Big-Data-Analysen für die Entwicklung von Assistenzsystemen <i>Andreas Heuer, Holger Meyer</i>	102

Optimierung von Datenflussprogrammen - ein Fall klassischer Anfrageoptimierung?

[Abstract]

Kai-Uwe Sattler
Technische Universität Ilmenau
Helmholtzplatz 5
Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

Datenflusssprachen haben in den vergangenen Jahren speziell im Kontext von Big-Data-Plattformen - etwa in Form von Pig oder Jaql - große Aufmerksamkeit gewonnen. Sie bieten sich jedoch auch für die Verarbeitung und Analyse dynamischer Daten bzw. Datenströme an. Ähnlich wie bei klassischen Anfragesprachen besteht bei Datenflusssprachen die Aufgabe, aus (mehr oder weniger) deklarativen Spezifikationen effiziente Ausführungspläne abzuleiten. Der Vortrag behandelt die Herausforderungen derartiger Optimierungen, geht auf Besonderheiten im Vergleich zur klassischen Anfrageoptimierung ein und diskutiert anhand von Beispielen aus den Bereichen Datenstromverarbeitung und MapReduce konkrete Problemstellungen.

About the Author

Kai-Uwe Sattler ist Professor für Datenbanken und Informationssysteme an der TU Ilmenau. Zu seinen Arbeitsgebieten zählen Datenbanksystemaspekte, Datenbankintegration sowie Anfrageverarbeitung für, heterogenen, massiv verteilte und dynamische Datenbestände. Er ist Koautor mehrerer Lehrbücher, u.a. zu verschiedenen Datenbankkonzepten wie Grundlagen, Implementierungstechniken, Data Warehousing und Cloud Data Management sowie zu Algorithmen und Datenstrukturen.

Scalable graph analytics with GRADOOP

[Abstract]

Erhard Rahm
University of Leipzig
Augustusplatz 10
Leipzig, Germany
rahm@informatik.uni-leipzig.de

ABSTRACT

Many Big Data applications in business and science require the management and analysis of huge amounts of graph data. Previous approaches for graph analytics such as graph databases and parallel graph processing systems (e.g., Pregel) either lack sufficient scalability or flexibility and expressiveness. We are therefore developing a new end-to-end approach for graph data management and analysis at the Big Data center of excellence ScaDS Dresden/Leipzig. The system is called Gradoop (Graph analytics on Hadoop). Gradoop is designed around the so-called Extended Property Graph Data Model (EPGM) which supports semantically rich, schema-free graph data within many distinct graphs. A set of high-level operators is provided for analyzing both single graphs and sets of graphs. The operators are usable within a domain-specific language to define and run data integration workflows (for integrating heterogeneous source data into the Gradoop graph store) as well as analysis workflows. The Gradoop data store is currently utilizing HBase for distributed storage of graph data in Hadoop clusters. An initial version of Gradoop is operational and has been used for analyzing graph data for business intelligence and social network analysis.

About the Author

Erhard Rahm is full professor for databases at the computer science institute of the University of Leipzig. His current research focusses on Big Data and data integration. He has authored several books and more than 200 peer-reviewed journal and conference publications. His research on data integration and schema matching has been awarded several times, in particular with the renowned 10-year best-paper award of the conference series VLDB (Very Large Databases) and the Influential Paper Award of the conference series ICDE (Int. conf. on Data Engineering). Prof. Rahm is one of the two scientific coordinators of the new German center of excellence on Big Data ScaDS (competence center for SCALable Data services and Solutions) Dresden/Leipzig that started its operation in Oct. 2014.

Next-Generation Hardware for Data Management - more a Blessing than a Curse?

[Abstract]

Wolfgang Lehner
Technische Universität Dresden
Noethnitzer Str. 46
Dresden, Germany
wolfgang.lehner@tu-dresden.de

ABSTRACT

Recent hardware developments have touched almost all components of a computing system: the existence of many and potentially heterogeneous cores, the availability of volatile and non-volatile main memories with an ever growing capacity, and the emergence of economically affordable, high-speed/low-latency interconnects are only a few prominent examples. Every single development as well as their combination has a massive impact on the design of modern computing systems. However, it is still an open question, if, how, and at which level of detail, a database system has to explicitly be aware of those developments and exploit them using specifically designed algorithms and data structures. Within the talk I will try to give an answer to this question and argue for a clear roadmap of HW/SW-DB-CoDesign especially providing an outlook to upcoming technologies and discussion of their non-functional properties like energy-efficiency and resilience behavior.

About the Author

Wolfgang Lehner is full professor and head of the database technology group at the TU Dresden, Germany. His research is dedicated to database system architecture specifically looking at crosscutting aspects from algorithms down to hardware-related aspects in main-memory centric settings. He is part of TU Dresden's excellence cluster with research topics in energy-aware scheduling, resilient data structures on unreliable hardware, and orchestration of wildly heterogeneous systems; he is also a principal investigator of Germany's national "Competence Center for Scalable Data Services and Solutions" (ScaDS); Wolfgang also maintains a close research relationship with the SAP HANA development team. He serves the community in many PCs, is an elected member of the VLDB Endowment, serves on the review board of the German Research Foundation (DFG), and is an appointed member of the Academy of Europe.

Ontologie-basierte Fragmentierungs- und Replikationsverfahren für verteilte Datenbanksysteme

Lena Wiese
Forschungsgruppe Knowledge Engineering
Institut für Informatik
Georg-August-Universität Göttingen
wiese@cs.uni-goettingen.de

ABSTRACT

Das Auffinden von semantisch verwandten Daten in großen Datenmengen ist aufwändig und daher zur Laufzeit einer Anfrage nur schwierig durchzuführen. In diesem Artikel stellen wir ein Verfahren vor, das Datentabellen anhand einer Ontologie in semantisch zusammenhängende Cluster aufteilt. Dadurch ergibt sich eine Ontologie-basierte Fragmentierung der Tabelle, die eine flexible Anfragebeantwortung unterstützt. Bei mehreren derartigen Fragmentierungen überschneiden sich Fragmente; dies wird für ein intelligentes Replikationsverfahren ausgenutzt, um die Anzahl der Replikaserver zu reduzieren.

Keywords

Verteiltes Datenbanksystem, Replikation, Ontologie, Fragmentierung, flexible Anfragebeantwortung

1. EINLEITUNG

Die Verwaltung von großen Datenmengen in Datenbanken erfordert die Verteilung der Daten auf mehrere Datenbank-Server. Dies bietet mehrere Vorteile:

- Lastverteilung: Datenbankabfragen können auf mehrere Server verteilt und damit parallelisiert werden.
- Verfügbarkeit: Durch die Lastverteilung erhöht sich die Verfügbarkeit des Gesamtsystems, da einzelne Anfragen seltener verzögert werden müssen.

Ein geeignetes Verfahren zur Fragmentierung (auch: Partitionierung oder Sharding) der Daten in Teilmengen ist daher notwendig. Die gängigen Fragmentierungsverfahren (Round-Robin, Hash-basiert, Intervall-basiert) ignorieren jedoch semantische Zusammenhänge der Daten.

Aus Gründen der Ausfallsicherheit ist zusätzlich noch die Replikation von Daten (also die Spiegelung desselben Datensatzes auf mehreren Servern) erforderlich. Dies gilt insbesondere für Hauptspeicherdatenbanken, die nicht über einen Hintergrundspeicher verfügen. In dieser Arbeit stellen wir

Verfahren zur Ontologie-basierten Fragmentierung und zur intelligenten Replikation vor, womit folgende Eigenschaften sichergestellt werden:

- durch die Fragmentierung wird ein Verfahren der *flexiblen Anfragebeantwortung* unterstützt, die dem Anfrager auch relevante verwandte Werte als Antworten zurückliefert.
- durch die Fragmentierung wird eine Lastverteilung auf mehrere Server möglich.
- durch die Fragmentierung wird die Anzahl der kontaktierten Server pro Anfrage reduziert und damit eine bessere Parallelisierung möglich.
- durch die intelligente Replikation wird die Anzahl der benötigten Replikaserver reduziert.

1.1 Übersicht

Abschnitt 2 beschreibt den Hintergrund zu flexibler Anfragebeantwortung, Fragmentierung und Datenverteilung. Abschnitt 3 stellt die Ontologie-basierte Fragmentierung inklusive Clustering, Fragmentverteilung und Anfragebeantwortung vor. Abschnitt 4 beschreibt darauf aufbauend ein Replikationsverfahren mit überlappenden Fragmenten. Ein Wiederherstellungsverfahren anhand des Replikationsverfahrens wird in Abschnitt 5 dargestellt. Es folgt ein Verfahren für abgeleitete Fragmentierungen in Abschnitt 6 und abschließend eine Zusammenfassung in Abschnitt 7.

2. HINTERGRUND

Wir stellen hier kurz Vorarbeiten zu flexibler Anfragebeantwortung, Fragmentierung und Datenverteilung vor.

2.1 Flexible Anfragebeantwortung

Flexible Anfragebeantwortung ist ein intelligentes Verfahren, um Datenbanknutzern Antworten zu liefern, die zwar nicht exakt der Anfrage entsprechen, die jedoch dennoch interessante Informationen für den Benutzer darstellen. Dabei gibt es syntaktische und semantische Ansätze.

Zum einen gibt es syntaktische Änderungen, die Teile der Anfrage verändern. Dazu zählen [Mic83]:

- Dropping Conditions: Selektionsbedingungen werden aus der Originalanfrage entfernt.
- Goal Replacement: einige Bedingungen der Originalanfrage werden anhand einer Ableitungsregel durch andere Bedingungen ersetzt.

- **Anti-Instantiation:** Ein Term (eine Variable mit mehreren Vorkommen oder eine Konstante) wird durch eine neue Variable ersetzt.

Diese Operatoren können auch kombiniert werden [IW11].

Diese syntaktischen Änderungen können aber zu weit gehen und zu viele Antworten produzieren – insbesondere beim Ersetzen von Konstanten durch neue Variablen in der Anti-Instantiation. Daher ist es wichtig, dass die Werte, die der neuen Variable zugewiesen werden können, beschränkt werden: es sollen nur semantisch äquivalente oder semantisch nah verwandte Werte zugelassen werden. Diese semantische Ähnlichkeit kann anhand einer Ontologie oder Taxonomie von Werten bestimmt werden. Für Einzelrechner wurden bereits vor einiger Zeit Verfahren vorgeschlagen – ohne jedoch verteilte Datenspeicherung mit einzubeziehen. CoBase [CYC⁺96] und [SHL07] zum Beispiel benutzten sogenannte Abstraktionshierarchien, um einzelne Werte zu generalisieren. Auch für XML-Anfragen wurden Verfahren entwickelt [HTGC10].

Ein grundlegendes Problem ist dabei jedoch, dass die Bestimmung von ähnlichen Werten zur Laufzeit (während der Anfragebeantwortung) viel zu ineffizient ist [BDIW14]. Dieses Problem wird durch das hier vorgestellte Fragmentierungsverfahren gelöst.

2.2 Fragmentierung

Im relationalen Modell gibt es die Theorie der Datenfragmentierung, die sich aufgrund des strikten Tabellenformats aufteilen lässt in horizontale und vertikale Fragmentierung (siehe zum Beispiel [ÖV11]):

- **Vertikale Fragmentierung:** Fragmente entsprechen Teilmengen von Attributen (also Tabellenspalten). Die Einträge in den Fragmenten, die zur selben Tabellenzeile gehören, müssen durch einen Tuple-Identifikator verbunden werden. Vertikale Fragmentierung entspricht einer Projektion auf der Tabelle.
- **Horizontale Fragmentierung:** Fragmente sind Teilmengen von Tupeln (also Tabellenzeilen). Eine horizontale Fragmentierung entspricht einer Selektion auf der Tabelle.
- **Abgeleitete Fragmentierung:** Eine bereits bestehende „primäre“ horizontale Fragmentation einer Tabelle induziert eine horizontale Fragmentierung einer weiteren Tabelle; dazu wird der Semi-JOIN auf beiden Tabellen ausgeführt.

Drei grundlegende Eigenschaften sind wichtig für Fragmentierungen:

- **Vollständigkeit:** Keine Daten dürfen während der Fragmentierung verloren gehen. Bei vertikaler Fragmentierung ist jede Spalte in einem Fragment enthalten; bei horizontaler Fragmentierung ist jede Zeile in einem Fragment enthalten.
- **Rekonstruierbarkeit:** Daten aus den Fragmenten können wieder zur Originaltabelle zusammengefügt werden. Bei vertikaler Fragmentierung wird der JOIN-Operator benutzt (basierend auf einem zusätzlich eingefügten Tuple-Identifikator), um die Spalten zu verbinden. Bei horizontaler Fragmentierung wird der Vereinigungsoperator zum Zusammenführen der Fragmente verwendet.

- **Redundanzfreiheit:** Um Duplizieren von Daten zu vermeiden, sollen einzelne Datensätze nur einem Fragment zugewiesen werden. Bei vertikaler Fragmentierung ist jede Spalte nur in einem Fragment enthalten (abgesehen vom Tuple-Identifikator). Bei horizontaler Fragmentierung ist jede Zeile in nur einem Fragment enthalten.

In anderen, nicht-relationalen Datenmodellen (Schlüssel-Wert-Speicher, Dokumentdatenbanken oder Spaltenfamilien-datenbanken) gibt es Fragmentierung meist über den Zugriffsschlüssel entweder Hash-basiert [KLL⁺97] oder basierend auf Intervallen. Jedoch unterstützt keines dieser Verfahren die flexible Anfragebeantwortung; im Gegensatz dazu hat das im Folgenden vorgestellte Fragmentierungsverfahren den Vorteil, dass eine relaxierte Bedingung aus nur einem Fragment beantwortet werden kann.

2.3 Datenverteilung

In einem verteilten Datenbanksystem müssen Daten auf die verschiedenen Server verteilt werden. Wichtige Eigenschaften sind

- **Datenlokalität:** Daten, auf die innerhalb einer Anfrage oder Transaktion zugegriffen wird, sollten möglichst auf einem Server sein. Dadurch verringert sich die Anzahl der kontaktierten Server und somit auch die Dauer der Anfragebeantwortung. Außerdem kann so die Parallelisierung der Anfragebeantwortung verbessert werden, da die anderen Server neue Anfragen annehmen können.
- **Lastverteilung:** Daten sollen so verteilt werden, dass parallele Anfragen möglichst auch von verschiedenen Servern beantwortet werden können, damit nicht einzelne Server unter Lastspitzen („hot spots“) leiden.

Einige Arbeiten befassen sich mit horizontaler Fragmentierung und Verteilung; jedoch unterstützen diese nur exakte Anfragebeantwortung und keine flexible Anfragebeantwortung wie in unserem Ansatz. Die meisten Ansätze gehen von einer vorgebenen Menge von Anfragen oder Transaktionen aus („workload“) und optimieren die Lokalität der Daten, die innerhalb der Anfrage beziehungsweise Transaktion benötigt werden. Dies ist für die Anwendung der flexiblen Anfragebeantwortung jedoch nicht anwendbar, da hier auch Werte zurückgegeben werden, die nicht explizit in einer Anfrage auftauchen.

[CZJM10] stellen Tuple als Knoten eines Graphen dar. Für eine vorgegebene Menge von Transaktionen fügen sie Hyperkanten in den Graph ein, wenn die Tuple in derselben Transaktion benötigt werden. Mit einem Graphpartitionierungsalgorithmus wird dann eine Datenfragmentierung ermittelt, die Zahl der zerschnittenen Kanten minimiert. In einer zweiten Phase benutzen die Autoren einen Klassifizierer aus dem Maschinellen Lernen, der eine Intervallbasierte Fragmentierung herleitet. Experimentell vergleichen sie dann das Graph-basierten mit Intervall-basierten und Hash-basierten Verfahren. Im Gegensatz zu unserem Ansatz wenden sie jedoch volle Replikation an, bei der alle Server alle Daten vorhalten; dies ist wenig realistisch in großen verteilten Systemen. Zudem vergleichen sie drei verschiedene Arten von Lookup-Tabellen, die Tuple-Identifikatoren für jedes Fragment abspeichern: Indexe, Bitarrays und Bloomfilter. Bei der Analyse unseres Systems stellt sich jedoch her-

aus, dass Lookup-Tabellen ineffizienter sind als das Einführen einer zusätzlichen Spalte mit der Cluster-ID in anderen Fragmentierungen.

Auch [QKD13] modellieren das Fragmentierungsproblem durch Minimierung zerschnittener Hyperkanten in einem Graphen. Zur Verbesserung der Effizienz komprimieren sie den Graphen und erhalten so Gruppen von Tupeln. Die Autoren kritisieren dabei auch den tupelweisen Ansatz von [CZJM10] als unpraktisch für große Tupelmengen. Die Autoren vergleichen ihren Ansatz mit zufälligen und tupelweisen Fragmentierungen und betrachten auch Änderungen der vorgegebenen Transaktionen.

[TCJM12] gehen von drei existierenden Fragmentierungen aus: Hash-basiert, Intervall-basiert und Lookup-Tabellen auf einzelnen Zugriffsschlüsseln. Sie vergleichen diese drei bezüglich Kommunikationskosten und Anfragendurchsatz. Zur Effizienzsteigerung analysieren sie diverse Komprimierungstechniken. Sie beschreiben Hash-basierte Fragmentierung als zu ineffizient. Die Autoren beschreiben jedoch nicht, wie die Fragmentierungen für die Lookup-Tabellen berechnet werden; im Gegensatz dazu stellen wir ein Ontologie-basiertes Fragmentierungsverfahren vor.

Im Gegensatz zu den meisten anderen Ansätzen gehen wir nicht von einer vorgegebenen Menge von Anfragen oder Transaktionen aus sondern schlagen ein allgemein anwendbares Clusteringverfahren vor, das die flexible Anfragebeantwortung zum Auffinden semantisch ähnlicher Antworten ermöglicht. Unsere Ergebnisse zeigen, dass Lookup-Tabellen (selbst wenn sie auf allen Servern repliziert werden) für unseren Ansatz zu ineffizient sind, dadurch dass viele JOIN-Operationen durchgeführt werden müssen.

3. ONTOLOGIE-BASIERTE FRAGMENTIERUNG

Unser Verfahren der Ontologie-basierten Fragmentierung beruht darauf, dass

- zur Anti-Instantiierung ein Attribut (also eine Tabellenspalte) ausgewählt wird.
- ein Clusteringverfahren auf dieser Tabellenspalte eingeführt wird, um die ähnlichen Werte innerhalb dieser Spalte zu gruppieren.
- anhand der Cluster die Tabelle zeilenweise fragmentiert wird.

Wie in Abbildung 1 dargestellt werden dann die Anfragen so weitergeleitet, dass zu einer Konstante aus der Originalanfrage das semantisch ähnlichste Fragment ermittelt wird und anschließend alle Werte des Fragments als relevante Antworten zurückgeliefert werden. Daher werden zum Beispiel bei einer Anfrage nach Husten auch die ähnlichen Werte Asthma und Bronchitis gefunden.

3.1 Clustering

Zu dem zur Anti-Instantiierung ausgewählten Attribut A in der gegebenen Tabelle F werden alle in der Tabelle vorhandenen Werte (die sogenannte aktive Domäne) ausgelesen durch Projektion $\pi_A(F)$. Anhand einer gegebenen Ontologie werden Ähnlichkeitswerte sim zwischen jeweils zwei Termen $a, b \in \pi_A(F)$ bestimmt im Wertebereich 0 (keine Ähnlichkeit) bis 1 (volle Ähnlichkeit). Dazu gibt es verschiedene Metriken, die meist auf der Berechnung von Pfaden zwischen den Termen in der Ontologie beruhen [Wie14, Wie13].

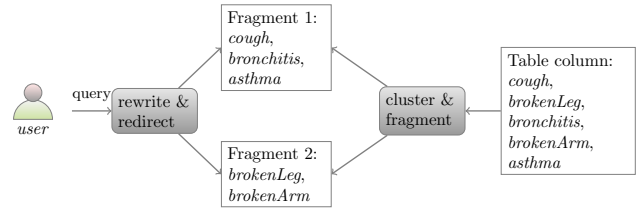


Figure 1: Fragmentation and query rewriting

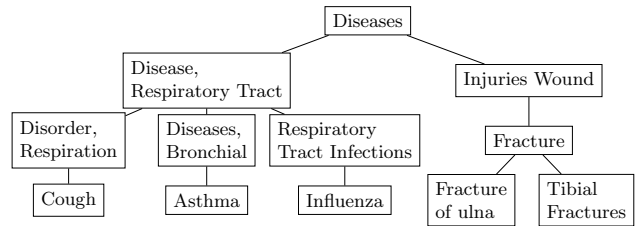


Figure 2: Beispieltaxonomie für Krankendaten

Ein Clustering-Verfahren benutzt diese Ähnlichkeitswerte um Cluster zu bestimmen (in Anlehnung an [Gon85]). Dazu wird in jedem Cluster ein prototypisches Element $head$ bestimmt, das das jeweilige Cluster repräsentiert. Zusätzlich gibt es einen Schwellenwert α und das Verfahren unterteilt Cluster solange in Teilcluster, bis für jedes Element innerhalb eines Clusters die Ähnlichkeit zu $head$ mindestens α ist. Das heißt, für jedes Cluster c_i und $head_i \in c_i$ gilt für jeden anderen Wert $a \in c_i$ (mit $a \neq head_i$), dass $sim(a, head_i) \geq \alpha$. Dieses Verfahren wird in Auflistung 1 dargestellt.

Listing 1 Clustering procedure

Input: Set $\pi_A(F)$ of values for attribute A , similarity threshold α

Output: A set of clusters c_1, \dots, c_f

- 1: Let $c_1 = \pi_A(F)$
- 2: Choose arbitrary $head_1 \in c_1$
- 3: $sim_{min} = \min\{sim(a, head_1) \mid a \in c_1; a \neq head_1\}$
- 4: $i = 1$
- 5: **while** $sim_{min} < \alpha$ **do**
- 6: Choose $head_{i+1} \in \bigcup_{1 \leq j \leq i} \{b \mid b \in c_j; b \neq head_j\}$
 $sim(b, head_j) = sim_{min}$
- 7: $c_{i+1} = \{head_{i+1}\} \cup \bigcup_{1 \leq j \leq i} \{c \mid c \in c_j; c \neq head_j; sim(c, head_j) \leq sim(c, head_{i+1})\}$
- 8: $i = i + 1$
- 9: $sim_{min} = \min\{sim(d, head_j) \mid d \in c_j; d \neq head_j; 1 \leq j \leq i\}$
- 10: **end while**

Zum Beispiel kann eine Taxonomie wie in Abbildung 2 benutzt werden, um die Tabellenspalte aus Abbildung 1 zu clustern.

3.2 Fragmentierung

Ein Clustering der aktiven Domäne von A induziert eine horizontale Fragmentierung der Tabelle F in Fragmente $F_i \subseteq F$. Jede aktive Domäne eines Fragments F_i entspricht genau den Werten in einem Cluster: $c_i = \pi_A(F_i)$. Die grundlegenden Eigenschaften einer Fragmentierung (Vollständig-

keit, Rekonstruierbarkeit, Redundanzfreiheit) sollen auch bei einer Clustering-basierten Fragmentierung gelten. Auch das Clustering muss vollständig sein: bei einer aktiven Domäne $\pi_A(F)$ muss dann für ein Clustering $C = c_1, \dots, c_n$ gelten, dass es die ganze aktive Domäne umfasst und kein Wert verloren geht: $c_1 \cup \dots \cup c_n = \pi_A(F)$. Die Eigenschaften einer Clustering-basierten Fragmentierung werden in Definition 1 zusammengefasst.

DEFINITION 1 (CLUSTERING-BAS. FRAGMENTIERUNG). Für ein Attribut A einer Tabelle F (eine Menge von Tupeln t) und ein Clustering $C = \{c_1, \dots, c_n\}$ der aktiven Domäne $\pi_A(F)$ und für $head_i \in c_i$ gibt es eine Menge von Fragmenten $\{F_1, \dots, F_n\}$ (definiert über denselben Attributen wie F), so dass folgende Eigenschaften gelten:

- *Horizontale Fragmentierung:* für jedes Fragment F_i gilt $F_i \subseteq F$
- *Clustering:* für jedes F_i gibt es in Cluster $c_i \in C$ so dass $c_i = \pi_A(F_i)$
- *Schwellenwert:* für jedes $a \in c_i$ (wobei $a \neq head_i$) gilt $sim(a, head_i) \geq \alpha$
- *Vollständigkeit:* für jedes Tupel t in F gibt es ein Fragment F_i , in dem t enthalten ist
- *Rekonstruierbarkeit:* $F = F_1 \cup \dots \cup F_n$
- *Redundanzfreiheit:* für jedes $i \neq j$, $F_i \cap F_j = \emptyset$ (oder auch $c_i \cap c_j = \emptyset$)

3.3 Fragmentverteilung

In verteilten Datenbanken müssen Fragmente verschiedenen Servern zugewiesen werden. Dieses Fragmentverteilungsproblem kann als ein Bin-Packing-Problem dargestellt werden:

- K Server entsprechen K Bins
- Jedes Bin hat maximale Kapazität W
- n Fragmente entsprechen n Objekten
- Jedes Objekt/Fragment hat ein Gewicht (oder Kapazitätsverbrauch) $w_i \leq W$
- alle Objekte müssen auf möglichst wenig Bins verteilt werden, wobei die Gewichtsgrenze W beachtet werden muss.

Dieses Bin-Packing-Problem kann auch als Problem der ganzzahligen linearen Optimierung dargestellt werden:

$$\text{minimize } \sum_{k=1}^K y_k \quad (1)$$

$$\text{s.t. } \sum_{k=1}^K x_{ik} = 1, \quad i = 1, \dots, n \quad (2)$$

$$\sum_{i=1}^n w_i x_{ik} \leq W y_k, \quad k = 1, \dots, K \quad (3)$$

$$y_k \in \{0, 1\} \quad k = 1, \dots, K \quad (4)$$

$$x_{ik} \in \{0, 1\} \quad k = 1, \dots, K, i = 1, \dots, n \quad (5)$$

Dabei entspricht x_{ik} einer Binärvariable die dann wahr (1) ist, wenn Fragment/Objekt i Server/Bin k zugewiesen wird; und y_k bedeutet, dass Server/Bin k belegt ist (also nicht leer). Gleichung (1) fordert, dass die Anzahl der belegten Server minimiert wird; Gleichung (2) erzwingt, dass jedes Fragment einem Server zugewiesen wird; Gleichung (3) bedeutet, dass die Kapazitätsgrenzen nicht überschritten werden; die letzten beiden Gleichungen stellen sicher, dass die Variablen binär sind.

Zusätzlich können noch die Eigenschaften der Datenlokalität und der Lastverteilung optimiert werden. Datenverteilung mit einer guten Datenlokalität platziert die Fragmente zusammen auf einen Server, die häufig gemeinsam innerhalb einer Datenbanktransaktion (oder auch innerhalb einer Anfrage) benutzt werden. Lastverteilung sorgt dafür, dass alle Server ungefähr dieselbe Anzahl von Anfragen beantworten müssen.

3.4 Metadaten

Für die Durchführung des Clustering, der Fragmentverteilung und der Anfrageumschreibung und -umleitung werden ein paar Metadaten benötigt.

Eine Tabelle **root** speichert einen Identifikator für jedes Cluster (Spalte ID), den Namen des Fragments (Name), den Repräsentanten des Clusters (Head), die Größe des Clusters (S) sowie den Server (Host), auf dem das Fragment gespeichert wird. Eine Beispieldatenbank sieht dann so aus:

ROOT	ID	Name	Head	S	Host
	101	Respiratory	Flu	4	S1
	107	Fracture	brokenArm	2	S2

Eine Tabelle **similarities** speichert die Ähnlichkeiten aller Werte des betrachteten Attributes zu den jeweiligen *head*-Werten der Cluster.

3.5 Anfragebeantwortung

Für die flexible Anfragebeantwortung wird die Konstante im entsprechenden Attribut A in der Anfrage anti-instantiiert und die entstehende Anfrage dann aus dem semantisch ähnlichsten Fragment beantwortet.

Als Beispiel betrachten wir eine Anfrage nach Husten in einer Tabelle *ill* mit Patienten IDs und Krankheiten:

```
SELECT patientid, disease
FROM ill WHERE disease='cough'
```

Über die Tabelle *similarities* wird das Fragment F_i ausgewählt, dessen *head* am ähnlichsten zu der anti-instantiierten Konstante (im Beispiel *cough*) ist.

```
SELECT TOP 1 root.name
FROM root, similarities
WHERE similarities.term='cough'
AND similarities.head = root.head
ORDER BY similarities.sim DESC
```

Der Name der Originaltabelle F wird durch den Namen des Fragmentes F_i ersetzt und die geänderte Anfrage an den entsprechenden Server gesendet. Dadurch werden alle Werte aus dem Fragment als relevante Antworten zurückgeliefert. Als Beispiel sei der Fragmentname *Respiratory*. Daher wird die Anfrage geändert zu:

```
SELECT patientid, disease FROM respiratory
und an den entsprechenden Server (hier S1) weitergeleitet.
```

Ähnliches gilt beim Einfügen neuer Zeilen:

```
INSERT INTO ill VALUES (349, 'asthma')
```

wird umgeschrieben zu:

```
INSERT INTO respiratory VALUES (349, 'asthma').
```

Auch das Löschen von Werten wird so umgesetzt:

```
DELETE FROM ill WHERE disease='cough'
```

wird zu:

```
DELETE FROM respiratory WHERE mesh='cough'
```

4. INTELLIGENTE REPLIKATION

Bisherige Replikationsverfahren kopieren die vorhandenen Fragmente auf andere Server; bei einer m -fachen Replikation verteilen sich so m verschiedene Kopien jedes Fragments auf m verschiedenen Servern. Dabei wird davon ausgegangen, dass die Fragmentierung redundanzfrei ist und sich die Fragmente daher nicht überschneiden: jedes Tupel ist in genau einem Fragment enthalten.

Bei der Ontologie-basierten Fragmentierung kann es jedoch sein, dass mehrere Attribute zur Anti-Instantiierung ausgewählt werden. Dadurch ergeben sich mehrere Fragmentierungen derselben Tabelle. Fragmente aus unterschiedlichen Fragmentierungen überschneiden sich deswegen. Bei α redundanzfreien Fragmentierungen ist daher jedes Tupel in α verschiedenen Fragmenten enthalten.

Beispielsweise kann unsere Beispieltabelle anhand eines Clusterings der Krankheitsdiagnose fragmentiert werden; so ergeben sich zwei Fragmente: Respiratory und Fracture.

<i>Respiratory</i>	<i>PatientID</i>	<i>Disease</i>
	8457	Cough
	2784	Flu
	2784	Asthma
	8765	Asthma
<i>Fracture</i>	<i>PatientID</i>	<i>Diagnosis</i>
	2784	brokenLeg
	1055	brokenArm

Zum Zweiten kann dieselbe Tabelle auch anhand der IDs der Patienten fragmentiert werden.

<i>IDlow</i>	<i>PatientID</i>	<i>Diagnosis</i>
	2784	Flu
	2784	brokenLeg
	2784	Asthma
	1055	brokenArm
<i>IDhigh</i>	<i>PatientID</i>	<i>Diagnosis</i>
	8765	Asthma
	8457	Cough

Dabei gilt, dass $Respiratory \cap IDlow \neq \emptyset$, $Respiratory \cap IDhigh \neq \emptyset$ und $Fracture \cap IDlow \neq \emptyset$.

Im Sinne der Minimierung der benutzten Server sollten nicht alle α Fragmentierungen m -fach kopiert werden, da dies zu $\alpha \cdot m$ Kopien jedes Tupels führt, obwohl m Kopien ausreichen würden. Das bedeutet, dass das hier vorgestellte Verfahren weniger Speicherplatzbedarf hat als eine konventionelle Replikation aller ontologie-basierter Fragmente.

Um eine m -fache Replikation pro Tupel zu erreichen, werden daher die Überschneidungen berücksichtigt. Wir gehen im Folgenden (ohne Beschränkung der Allgemeinheit) davon aus, dass $\alpha = m$ – andernfalls werden einige Fragmentierungen dupliziert bis die entsprechende Anzahl erreicht ist. Damit ist also jedes Tupel in m Fragmenten enthalten.

Das Fragmentverteilungsproblem lässt sich daher erweitern um die Bedingung, dass überlappende Fragmente ($i \cap i' \neq \emptyset$) auf verschiedenen Server platziert werden (Gleichung (9)). Im Beispiel kann also *Fracture* nicht mit *IDlow* auf einem Server platziert werden; jedoch können *Fracture* und *IDhigh* auf demselben Server liegen. Generell handelt es sich dann dabei um ein Bin Packing Problem with Conflicts

(BPPC):

$$\text{minimize } \sum_{k=1}^K y_k \quad (6)$$

$$\text{s.t. } \sum_{k=1}^K x_{ik} = 1, \quad i = 1, \dots, n \quad (7)$$

$$\sum_{i=1}^n w_i x_{ik} \leq W y_k, \quad k = 1, \dots, K \quad (8)$$

$$x_{ik} + x_{i'k} \leq y_k \quad k = 1, \dots, K, \quad i \cap i' \neq \emptyset \quad (9)$$

$$y_k \in \{0, 1\} \quad k = 1, \dots, K \quad (10)$$

$$x_{ik} \in \{0, 1\} \quad k = 1, \dots, K, \quad i = 1, \dots, n \quad (11)$$

Um diese Konflikte (überlappende Fragmente) zu identifizieren werden Fragmente aus verschiedenen Fragmentierungen verglichen. Im Beispiel gibt es Fragmente c_i mit einer Cluster-ID (Fragmentierung wie im obigen Beispiel über den Werten der Krankheiten) und Fragmente r_j mit einer Range-ID (Fragmentierung über den Werten der Patienten-ID):

```
SELECT DISTINCT clusterid, rangeid
```

```
FROM ci JOIN rj ON (rj.tupleid=ci.tupleid)
```

für jedes Cluster-Fragment c_i und jedes Range-Fragment r_j . Danach wird das resultierende BPPC gelöst und die Fragmente auf entsprechend viele Server verteilt mittels:

```
ALTER TABLE ci MOVE TO 'servername' PHYSICAL.
```

5. WIEDERHERSTELLUNG

Passend zum Replikationsverfahren müssen im Falle eines Serverausfalls einige Fragmente wiederhergestellt werden. Dazu werden der Originaltabelle Spalten für die jeweiligen Cluster-Identifikatoren hinzugefügt. Anhand der IDs können die entsprechenden Fragmente rekonstruiert werden:

```
INSERT INTO ci SELECT * FROM rj WHERE clusterid=i
```

Alternativ ist die Erstellung einer sogenannten Lookup-Tabelle [TCJM12] möglich, die zu jeder Cluster-ID die beteiligten Tupelidentifikatoren abspeichert. Diese benötigt jedoch einen JOIN-Operator:

```
INSERT INTO ci SELECT * FROM ill JOIN lookup
ON (lookup.tupleid=rj.tupleid)
WHERE lookup.clusterid=i
```

Die Lookup-Tabelle hat sich daher als ineffizienter herausgestellt.

6. DATENLOKALITÄT FÜR ABGELEITETE FRAGMENTIERUNGEN

Wenn auf mehrere Tabellen innerhalb einer Anfrage zugegriffen wird und diese Tabellen Join-Attribute gemeinsam haben, kann durch abgeleitete Fragmentierung die Datenlokalität für die Anfragen erhöht werden. Zum Beispiel sei zusätzlich zur Tabelle *ill* eine Tabelle *info* gegeben, die zu jeder Patienten-ID Adressangaben enthält. Eine mögliche Anfrage wäre daher, die Angaben zu Krankheiten und Adressen zu kombinieren:

```
SELECT a.disease, a.patientid, b.address
FROM ill AS a, info AS b WHERE disease='cough'
AND b.patientid= a.patientid
```

Anhand der vorgegebenen primären Fragmentierung der Tabelle *ill* wird dann auch die Tabelle *info* fragmentiert, zum Beispiel für das Fragment *Respiratory*:


```

INSERT INTO inforesp
SELECT a.patientid, b.address
FROM respiratory AS a, info AS b
WHERE b.patientid = a.patientid

```

Daher kann dann im Folgenden auch die Anfrage, die Angaben zu Krankheiten und Adressen kombiniert, entsprechend umgeschrieben werden:

```

SELECT a.disease, a.patientid, b.address
FROM respiratory AS a
JOIN inforesp AS b ON (a.patientid=b.patientid)

```

7. ZUSAMMENFASSUNG UND AUSBLICK

Flexible Anfragebeantwortung unterstützt Benutzer bei der Suche nach relevanten Informationen. In unserem Verfahren wird auf eine Ontologie zurückgegriffen aufgrund derer semantisch ähnliche Werte in einem Cluster zusammengefasst werden können. Eine Fragmentierung der Originaltabelle anhand der Cluster ermöglicht ein effizientes Laufzeitverhalten der flexiblen Anfragebeantwortung. Durch einige Metadaten (Root-Tabelle, Similarities-Tabelle, zusätzliche Spalte für Cluster-ID) werden alle typischen Datenbankoperationen unterstützt.

Zukünftig soll insbesondere das dynamische Anpassen der Fragmente untersucht werden: da sich durch Einfügungen und Löschungen die Größen der Fragmente stark ändern können, müssen zur Laufzeit Fragmente verschoben werden, sowie gegebenenfalls zu kleine Fragmente in ein größeres vereinigt werden beziehungsweise zu große Fragmente in kleinere aufgespalten werden.

8. REFERENCES

- [BDIW14] Maheen Bakhtyar, Nam Dang, Katsumi Inoue, and Lena Wiese. Implementing inductive concept learning for cooperative query answering. In *Data Analysis, Machine Learning and Knowledge Discovery*, pages 127–134. Springer, 2014.
- [CYC⁺96] Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow, and Chris Larson. CoBase: A scalable and extensible cooperative information system. *JHIS*, 6(2/3):223–259, 1996.
- [CZJM10] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism: a workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 3(1):48–57, 2010.
- [Gon85] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- [HTGC10] J. Hill, J. Torson, Bo Guo, and Zhengxin Chen. Toward ontology-guided knowledge-driven xml query relaxation. In *Computational Intelligence, Modelling and Simulation (CIMSIM)*, pages 448–453, 2010.
- [IW11] Katsumi Inoue and Lena Wiese. Generalizing conjunctive queries for informative answers. In *Flexible Query Answering Systems*, pages 1–12. Springer, 2011.
- [KLL⁺97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.
- [Mic83] Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.
- [ÖV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, Berlin/Heidelberg, Germany, 2011.
- [QKD13] Abdul Quamar, K. Ashwin Kumar, and Amol Deshpande. Sword: scalable workload-aware data placement for transactional workloads. In Giovanna Guerrini and Norman W. Paton, editors, *Joint 2013 EDBT/ICDT Conferences*, pages 430–441, New York, NY, USA, 2013. ACM.
- [SHL07] Myung Keun Shin, Soon-Young Huh, and Wookey Lee. Providing ranked cooperative query answers using the metricized knowledge abstraction hierarchy. *Expert Systems with Applications*, 32(2):469–484, 2007.
- [TCJM12] Aubrey Tatarowicz, Carlo Curino, Evan P. C. Jones, and Sam Madden. Lookup tables: Fine-grained partitioning for distributed databases. In Anastasios Kementsietsidis and Marcos Antonio Vaz Salles, editors, *IEEE 28th International Conference on Data Engineering (ICDE 2012)*, pages 102–113, Washington, DC, USA, 2012. IEEE Computer Society.
- [Wie13] Lena Wiese. Taxonomy-based fragmentation for anti-instantiation in distributed databases. In *3rd International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC'13) collocated with IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 363–368, Washington, DC, USA, 2013. IEEE.
- [Wie14] Lena Wiese. Clustering-based fragmentation and data replication for flexible query answering in distributed databases. *Journal of Cloud Computing*, 3(1):1–15, 2014.

Automated Silhouette Extraction for Mountain Recognition

Daniel Braun
Heinrich-Heine-Universität
Institut für Informatik
Universitätsstraße 1
40225 Düsseldorf, Deutschland
braun@cs.uni-duesseldorf.de

Michael Singhof
Heinrich-Heine-Universität
Institut für Informatik
Universitätsstraße 1
40225 Düsseldorf, Deutschland
singhof@cs.uni-duesseldorf.de

ABSTRACT

With the rise of digital photography and easy sharing of images over the internet, a huge amount of images with no notion of what they are showing exists. In order to overcome this problem, we – for the example of mountain recognition – introduce a method, that is able to automatically recognise a mountain shown in a photograph.

Our method does not require GPS information stored in the image, since most images are not GPS tagged, either because of the absence of a GPS sensor in the device or because it has been deactivated for a lesser power consumption, which often is the case with smartphones. Instead, we propose a method that is able to automatically extract the mountain’s silhouette from a given image. This silhouette is then cleaned by removing artefacts and outliers, such as trees and clouds, with a histogram based approach. Finally, the cleaned silhouette can be compared to reference data in order to recognise the mountain that is shown in the picture. For this, time series comparison techniques can be used to find matching silhouettes. However, because of the huge number of reference silhouettes to compare against, we argue, that a preselection of those silhouettes is necessary and point out approaches to this problem.

Categories and Subject Descriptors

I.4.8 [IMAGE PROCESSING AND COMPUTER VISION]: Scene Analysis—*Object recognition*; I.4.6 [IMAGE PROCESSING AND COMPUTER VISION]: Segmentation—*Edge and feature detection*; H.2.8 [DATABASE MANAGEMENT]: Database Applications —*Data mining*

Keywords

Object Recognition, Image Annotation, Outlier Detection, Image Segmentation, Skyline Recognition, Mountain Recognition, Time Series

1. INTRODUCTION

Sharing our experiences with digital images is a significant part of our today’s life, which is partly a result of the high availability of digital cameras, like in smartphones, and the high use of social networks, that simplifies the publication and sharing of images. As a consequence, the number of images in the world wide web increases significantly. For example, this can be seen on the image sharing platform Instagram, where users share an average of 70 million new photos per day [1].

As a result of such high numbers of images, searching photos which show specific objects is challenging, because the majority of these images is not properly tagged with the names of every object seen in them. So the need for efficient algorithms for automatic object recognition rises. In the last decades there were many advances in this research field, but especially the automatic identification of landmarks, which are subject to weather changes, areal erosion and vegetation, is still a challenging task, even if the amount of images with attached GPS data, which marks the geo-position of the camera when the photo was shot, is rising.

The growing spread of devices with the capability of generating GPS tags for the images, like smartphones and digital cameras with GPS units, enables many possibilities for an subsequent geo-localisation of images, due to the fact that GPS tags can significantly reduce the number of possible sights, buildings or landmarks to compare with. However, there exist too many images without the advantage of GPS tags, so that an automatic geo-localisation without prior knowledge of the camera position is still a valuable aim.

Our focus lies on the automatic landmark recognition, which we will describe by using the example of mountain recognition in images. To solve the question, which mountain can be seen on an given image, we match the skyline of the mountain in the image with silhouettes of mountains in our database. For this purpose, we have to automatically extract the exact skyline from the image, which is a difficult task, because the segmentation of the image can lead to artefacts, for instance due to weather conditions, noise, obstacles or overexposure.

In this paper we introduce a baseline segmentation algorithm, which uses an outlier detection algorithm to identify and eliminate these artefacts. The article is structured as follows: In the next section we discuss other papers related to our algorithm. We then introduce our algorithm, which consists of the three steps silhouette extraction, silhouette cleaning and silhouette matching. The section of the latter

one will be a perspective of how to use the cleaned silhouette in further steps. The last chapter summarises this paper and outlines future work.

2. RELATED WORK

The amount of publications for automated mountain recognition increased significantly in the last two decades. Given a high number of publicly available digital elevation maps (DEM), the consensus in many publications [2, 3, 4, 5, 10] is to use image-to-model matching for mountain recognition and orientation identification.

Many approaches, like [3, 4, 10], make use of known GPS data for the image to align in the terrain. This reduces the search space for possible mountain silhouettes significantly, because the search of the correct mountains is limited in checking the surrounding of the camera’s position.

Baboud et al. [3] need an estimated field-of-view to calculate the rotation which maps the image to the terrain. Therefore, the authors introduce a robust matching metric, using extracted edges in combination with a search space reduction to further reduce computation time. [10] uses a standard Sobel-filter for the edge extraction. To identify the edges which are part of the mountain’s contours, the authors propose the use of the Random Ferns classifier. Afterwards they match the emerged contour map with the contour map extracted from the DEM. In [4] a 360-degree panorama of the surroundings of the image location is synthesized out of a given DEM and used for matching the skyline extracted from the image. For that, they use a vector cross correlation (VCC) technique to find the candidate matches. After further refinement and recalculation of the VCC for each peak they can label the peaks with a high precision.

All three approaches show good results for their issue, but the need for GPS data for the processed image does not match our problem specifications, different from Baatz et al. [2], in which the focus lies on images without GPS tag. They use an approach based on a cost function for the belonging of a pixel to the sky respectively foreground. They combine this approach with a relevance feedback-like user interference, where the user can mark parts of the sky or the foreground. This user intervention was needed for 49% of the images in their dataset, which was collected during there research. Thankfully, they published this dataset, so that it will be used in this paper. After the contour extraction, they match the coded contourlet with the contours extracted from a DEM at several points on a predefined grid. At last, when they find a suitable match, they recalculate the geo-position of the camera. Naval et al. [5] also try to find the camera position and orientation, using a DEM to position the camera on the world. For this purpose they match the extracted skyline from an image with a synthetic skyline from a DEM. Different to both works we try to get an automatically cleaned silhouette, thus removing obstacles or other artefacts, out of the processed image.

Tao et al. [12] focus on the identification of the sky seen in an image and the search for images with a specific sky appearance. Therefore they define different sky attributes, like for example the sun position, which they extract individually afterwards. At last they present their complete system SkyFinder, in which an attribute based search is implemented. On top of that, they provide a sky replacement algorithm for changing the sky in an image. However, the recognition of the skyline is not part of this system.

3. SILHOUETTE RECOGNITION

The silhouette recognition process is basically a process in three steps. The first step is the extraction of the silhouette from a given picture. This silhouette is stored as a polygonal chain. During the second step, this silhouette is cleaned by identifying and removing outliers. The cleaned silhouette is then used as the input to the third and final step, which consists of matching the silhouette against the reference data in order to be able to identify the structure in the picture.

3.1 Silhouette Extraction

Extracting the silhouette is the first step in the mountain identification process described in this paper. The task of this step is the separation of the sky from the rest of the processed image. We therefore have a binary segmentation task to solve, in which we check every pixel p of an image and decide if p shows a part of the sky or not. For a human this would be easy to do in most cases, even though it would be too much work and time to segment great numbers of pictures manually. Because of that, we use a growing seed algorithm, like the algorithm described in [11], for which we use the fact, that in most cases the pixels at the upper bound of the image are part of the sky. In the following section, we will first describe some difficulties, which can make the segmentation task more complex. After that, our baseline segmentation algorithm will be further explained.

The segmentation of an image generally suffers from different problems, like under-/overexposure, which results in a smaller difference between the pixel colours, or blur, which can be, for example, a result of a lossy compression of the image. In addition, our binary segmentation task has even some own problems to deal with. The first difficulty is a consequence of the motif itself, because the weather in mountainous terrain is very volatile. This and the fact that, for example in the alps, many mountain peaks are covered with snow, can make the extraction of the mountain silhouette out of an image imprecise. Furthermore differently coloured bands of cloud can lead to an extracted silhouette which lies in the sky and is therefore not part of the real skyline. The second one is a result of possible obstacles, which hide the real silhouette of the mountain or lead to smaller segments within the sky segment.

Even though we are aware of these problems, our naive segmentation algorithm cannot handle all of them. For the identification of artefacts as result of segmentation errors, we use the second step of our chain, which is described in section 3.2. Our segmentation algorithm uses the upper row of pixels as seed for the sky segment. This means that we mark the upper pixels as sky and process every neighbouring pixel to let this segment grow. For this purpose we first convert the processed photo to a gray-scale image G . Now we can define V_G as overall variance of the brightness values. Having V_G , we now process every pixel $p_{(x,y)}$ which is not a part of the sky segment and mark it as sky candidate, if for $p_{(x,y)}$ it holds that

$$B_{p_{(x,y)}} - \text{mean}_{p_{(x,y)}}^r < \gamma \cdot \sqrt{V_G}$$

with $B_{p_{(x,y)}}$ the brightness of the pixel $p_{(x,y)}$, $\text{mean}_{p_{(x,y)}}^r$ as the mean of the brightness in an neighbourhood of the pixel with the radius of r and γ as a factor to scale the impact of the standard derivation. This means that we mark a pixel, if its distance to a local mean of brightness values

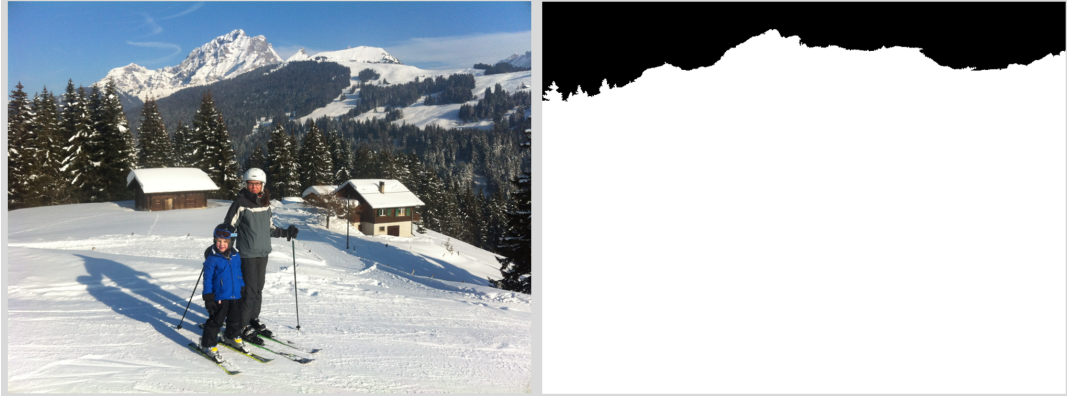


Figure 2: The result of the segmentation algorithm. (Left) The original image out of the dataset from [2]. (Right) The binary image after the segmentation. White pixels mark the ground segment.

is smaller than a fixed percentage of the global standard derivation of all brightness values. The idea behind this term is, that the border between sky and ground has in most cases a stronger contrast than the rest of the image, especially the sky. Therefore, the distance to the mean will be higher at the skyline as it will be at the border of possible clouds. With the connection of the upper bound to the standard derivation, we want to take into account the images where the brightness is very homogenous, for example due to overexposure, and therefore the contrast of the skyline decreases.

In our experience this naive assumption shows good results for $r = 5$ and $\gamma = 0.1$ on most images out of the swiss dataset published in [2]. In the future we will test more complex algorithms, like for instance the skyline extraction algorithm proposed in [8], with the expectation that they will yield even better results. After we have marked all pos-

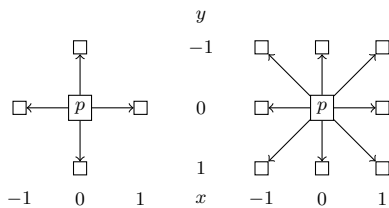


Figure 1: Two possible neighbourhoods of a pixel. (Left) 4-connected neighbourhood of the pixel p . (Right) 8-connected neighbourhood of the pixel p .

sible candidates, we can finally let the sky segment grow. For that, we check for every pixel $p(x, y)$, if it is a 4-connected neighbour, for explanation see the left side of figure 1, to a pixel of the sky segment and marked as sky candidate (see the previous step). If so, the pixel will get marked as sky. This step is repeated until no more pixels can be added to the sky segment. At this point, we have a binary image, which represents the skyline as transition between the sky and the ground, like the one shown in figure 2.

For the silhouette extraction, we search a silhouette $S = (v_1, \dots, v_n)$ where every vertex $v_i = (x_i, y_i)$ is a two dimensional point where x_i describes the x -coordinate and y_i

describes the y -coordinate of that point in the picture. We start at the upper left pixel $p_{(0,0)}$ and search for the first non-sky pixel as start vertex v_1 for our polygonal chain, which lies on the left pixel column with $x_1 = 0$. Now, we have two possibilities: First, we can find no pixel, which results in checking the next column until we find a pixel or we have reached the lower right pixel of the image. Otherwise, if a possible skyline pixel was found, the algorithm tracks the transition between the sky and the ground in the following manner.

Having v_i as the last extracted vertex of the silhouette, the algorithm checks the 8-connected neighbours of this point (see the right side of figure 1) and chooses the non-sky point as next vertex which has the lowest angle between the incoming line, defined by the two last vertices v_{i-1} and v_i , and the outgoing line, defined by v_i and the neighbouring point. This can easily be done, as shown in figure 3, by checking the 8-connected neighbours from v_i in clockwise direction, starting at the predecessor. The only exception is the start point v_1 , where we set $v_0 = (-1, y_1)$. This means that we choose in this case the left direct neighbour, which lies outside of the image, as predecessor.

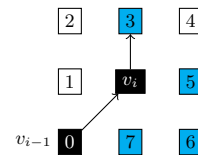


Figure 3: An example how to choose the next silhouette point. Starting at v_{i-1} as predecessor of the current vertex v_i , the algorithm searches in clockwise direction for the next ground pixel (blue). Therefore the third pixel will be chosen.

Now, we have the border of the mountain as chain of neighbouring pixels. To reduce the amount of vertices without losing any information, we eliminate all vertices which bring no further information gain to the final silhouette in the last step of the extraction phase. For this, we define the



Figure 4: Silhouette as black line with outliers in red on original image.

information gain I of a vertex v_j , with $1 < j < n$, as

$$I_{v_j} = \begin{cases} 1, & \text{if } \angle v_{j-1}v_jv_{j+1} \neq 180^\circ \\ 0 & \text{otherwise.} \end{cases}$$

This means that every vertex which lies on one line with his predecessor and successor carries no further information for the extracted silhouette. After deleting every vertex with $I_{v_j} = 0$, we obtain our final polygonal chain, which can now be analyzed in the the cleaning phase described in the following section.

3.2 Silhouette Cleaning

The extracted silhouette from the previous step may contain different disturbances. In this step, we try to get rid of those, so that they cannot affect the step of silhouette matching that gets described in section 3.3. Examples for such disturbances are manifold, ranging from objects in front of the mountain, such as trees, to errors produced during the extraction process that can be caused by a low contrast between the mountain line and the sky. Essentially, after finding an outlier, we currently cut elevations within this part of the silhouette away.

Some of such outliers are showcased in figure 4. Here, the black line is the silhouette that has been extracted by the extraction step as described in section 3.1. The red parts are those parts of the silhouette, that are automatically detected as outliers. Examples for outliers are the tree tops to the left, that have correctly been marked as outliers. In the center of the picture, there is a part of the mountain, that has been cut out. Here the contrast between rock and blue sky gets to low for the extraction step to distinguish them. Right next to this, clouds around the lower peak in the back

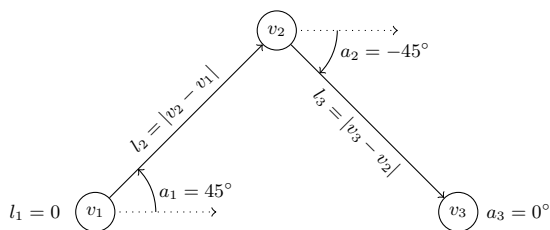


Figure 5: Conversion of polygonal chain for better pattern recognition.

get recognised as mountain. Since the outline of this outlier is not atypical for a mountain silhouette, this outlier is not recognised. Farther to the right, there are two other outliers where clouds are recognised as mountain by the extraction step. These are, in this case, marked red and such detected by the silhouette cleaning step.

Note, that there are a few parts of the silhouette that get marked as outliers but are not described above. These are false positives. However, since the image here is used to showcase the different kind of outliers, these are not discussed here.

The silhouette cleaning part of our algorithm is, again, divided into three parts. As an input, it gets the silhouette $S = (v_1, \dots, v_n)$ extracted from a picture.

For the recognition of patterns inside the polygonal chain, this representation has disadvantages, because all vertices are given in absolute positions. For outlier detection, a model where similar structures do look similar in their representation is beneficial. Therefore, we convert S to a representation $AP = (ap_1, \dots, ap_n)$, with $ap_i = (l_i, a_i)$. Here, we set

$$l_i = \begin{cases} |v_i - v_{i-1}|, & \text{if } i > 1 \\ 0 & \text{else} \end{cases}$$

and a_i as the angle between the vector $v_{i+1} - v_i$, and the x -axis for $i < n$ and $a_n = 0^\circ$. Figure 5 illustrates this. During this step, points where the angle does not change are removed. Also, artefacts consisting of angles of 180° between two following segments get removed.

The basic idea of the outlier detection method itself is to compare histograms created from parts of the polygonal chain AP to a reference histogram. The latter hereby is created from the silhouettes of the ground truth data as given in [2].

The mentioned histograms consist of the two dimensions segment length and angle size, just as the points in the transformed polygonal chain AP , and are normalised. This means, that the sum of the frequencies of each bucket is 1. For the reference data, we compute one histogram from each image's silhouette. Finally, the mean of all these histograms is computed and becomes the reference histogram H_r .

In order to find outliers, we use a sliding window approach over the polygonal chain AP_i of a given input image. We therefore use a section of m successive segments ap_i, \dots, ap_{i+m-1} in order to compute a histogram H_i with the same bucket distribution as in H_r . Then, for every point

used to compute H_i , we store the distance between H_i and H_r . By this approach, for most points, multiple distance values get stored when the sliding window is moved on. The final distance d_i for a point ap_i is now computed as the average distance of all distances stored for this point.

As distance function in this case we use the one given in the following

DEFINITION 1. Let $G = (g_1, \dots, g_k)$, $H = (h_1, \dots, h_k)$ be normalised histograms with the same bucket distribution.

The above average distance of G to H is defined by

$$D(G, H) := \max(|aab(G)|, |aab(H)|) - |aab(G) \cap aab(H)|,$$

where

$$aab(F) := \left\{ i \in \{1, \dots, k\} \mid f_i \geq \frac{1}{k} \right\}$$

with F being a normalised histogram with k buckets.

This can be implemented to compute in linear time to the number of buckets, or, for histograms of a fixed length, in constant time.

Based on a number of images used as training data, in the same way as described above, the medium distance μ and the standard deviation σ can be determined. We use these precomputed values in order to find two thresholds $\tau_{in} > \tau_{out}$, such that we can characterise a point ap_i as a strong anomaly if it holds that its distance

$$d_i \geq \mu + \tau_{in} \cdot \sigma$$

and a weak anomaly if

$$d_i \geq \mu + \tau_{out} \cdot \sigma.$$

We also determine a length threshold l . For a part of the polygonal chain, to be recognised as an outlier, it must hold that at least l successive points ap_i, \dots, ap_{i+l-1} must be strong anomalies. An outlier can have any length that is equal to or bigger than l . Finally, if we find such an outlier, we expand it by adding points to it that are adjacent to the outlier and weak anomalies. By this, it is possible for outliers to merge.

As an example, say, ap_7, \dots, ap_{20} are strong anomalies because their distances d_7, \dots, d_{20} are bigger than $\mu + \tau_{in} \cdot \sigma$. If we set $l = 4$ for this example, these thirteen points are recognised as an outlier $o = \{7, \dots, 20\}$. Now, let us assume that ap_5 and ap_6 are weak anomalies as well as ap_{22} . Then ap_6 and ap_5 belong to the extended outlier, because they are both adjacent to the outlier. On the other hand, ap_{22} does not belong to the extended outlier, because ap_{21} is not part of the outlier since it is not a weak anomaly.

Once all outliers have been detected, the next step is to remove them. Currently, we use a very simple approach for this where, in the original silhouette S , the overhanging part of the outlier is replaced by a straight line. This is based on the notion, that the reasons for most outliers in the silhouette are either trees or clouds. By just removing them, in many cases, we get results that resemble the original shape of the mountain in an easy way.

Let $o = \{i, \dots, j\}$ with $i + l \leq j$ be an outlier. Then we draw a straight line from v_i to v_j . Now, while the distance between v_s , starting with $s = i + 1$, and that straight is smaller than a preset value d , we find the vertex v_s with the largest index, that is close enough to the original straight line. The same is done from the other end of the outlier

so that we find a vertex v_e . This results in four vertices' indices $j \leq s < e \leq j^1$. From this, the part between v_s and v_e is now finally replaced by a straight line between these two vertices.

By this, we obtain a cleaned silhouette that can be used in further steps for the silhouette matching.

3.3 Silhouette Matching

Once the silhouette is extracted and cleared of outliers, it is now time to match it to the reference data in order to determine the mountain that is shown by the picture. Currently, we have not implemented any method, yet. However, we will introduce some methods that we are going to test in the future.

The converted silhouettes $AP = (ap_1, \dots, ap_n)$ from the previous section can be easily changed to be interpreted as time series by setting $AP' = (ap'_1, \dots, ap'_n)$ where

$$ap'_i = (l'_i, v_i) = \left(\sum_{j=1}^i l_j, v_i \right)$$

for $ap_i = (l_i, v_i)$. By this conversion, every point ap'_i has the length of the polygonal chain until that point as its first component. Since $l_i > 0$ for all $i > 1$, the length component of AP' is strictly monotonic increasing, just as the time dimension in a time series. This is because we do not allow identical points in our silhouette. With this, it is possible to compare different silhouettes for similarity by using time series comparison techniques such as [6, 9]. These methods have the advantage of being rotation invariant which, in our case, means that the image does not have to have the same image section as our reference data.

Due to the great number of mountains and peaks that exist and on the notion, that every peak can be photographed from different angles, there are hundreds of thousands of silhouettes to match for each query image. Due to this it is clear, that even with a high-performance matching method a preselection is necessary.

There exist many methods suitable for this, such as the technique presented in [7] that uses the position of the sun and the time of day, at which the photo has been taken, in order to compute the approximate location on earth. However, this method has an average localisation error of about 100 km. Because of this, it is useful for a rough preselection, however not sufficient in our case.

Therefore, we aim to reuse the idea of our outlier detection method for matching. Instead of computing histograms of short sequences of the silhouette, in this case the histogram H_S over the whole silhouette AP is computed. This is then compared to the reference images' histograms $H_{R_i}, i \in \{1, \dots, n_{ref}\}$ with n_{ref} the number of the reference image, which we, as discussed in section 3.2, have to compute anyway. The comparison between two histograms, with our distance function, is linear to the number of buckets in the histograms, or, since this number is fixed, constant. Now, let $d(H_S, H_{R_i})$ denote the distance between the histogram of the new silhouette S and the i th reference histogram. If $d(H_S, H_{R_i})$ is small, this does not necessarily mean, that the silhouettes are identical or even really similar, because the histogram representation does not preserve the order of the contained elements. On the other hand, if the distance

¹In general, it is possible, that $s \geq e$. In this case, no substitution is performed.

between two silhouettes is large, we can say that those silhouettes are not similar.

Due to this, we plan to only use the time series comparison methods from the beginning of this section on those reference silhouettes, that yield small histogram distances. Further on, we will evaluate if the position determination method presented in [7] is able to boost the performance of our solution.

4. CONCLUSION AND FUTURE WORK

In this work we presented a new approach to motif recognition based on silhouettes on the example of mountains. Our method consists of three steps, of which the first two have already been implemented while we are working on the third step. First results show that we are able to extract silhouettes with relatively few errors from images and that our outlier detection step does indeed find meaningful anomalies. We have currently tested the first two steps of our

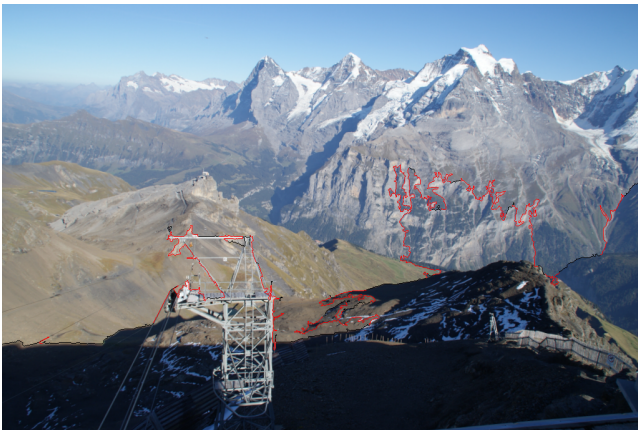


Figure 6: Silhouette as black line with outliers in red on original image.

approach as described in sections 3.1 and 3.2 on 18 images from the reference dataset. Results for the first of these images can be seen in figures 2 and 4. Of these 18 images, 17 result in good silhouettes, i.e. silhouettes with relatively few outliers of which most get marked and are correctable. The last image, though, does not get recognised correctly due to low contrast in terms of brightness. This is illustrated by figure 6. We do, however, notice this, because most of the silhouette gets marked as outlier.

The next step is to implement a silhouette matching algorithm. This has already been outlined in section 3.3. Of course, it is necessary, to benchmark the parts of that step in order to find weaknesses early on. Once the system is complete we aim to evaluate it on the whole of the dataset of [2]. Based on these result, we will tune the parameters of our method to find settings, that do work well generally.

We also aim to create a mountain recognition corpus of our own since [2] focuses on images of mountains in Switzerland only. Our corpus is aimed to be international and should feature images from mountains from all over the world.

Another interesting perspective is to test, whether our framework will work on other types of images, such as city skylines or pictures of single buildings. With these tasks the method itself could be quite similar, since skylines and

buildings are mostly photographed with the sky as background, too. Furthermore, we aim to test our method on more diverse areas, such as the recognition of certain objects in MRT screenings or X-rays. These tasks will make it necessary to change some parts of our approach, naturally, because there, it will be interesting to be able to tell, for example, which organs are shown in a picture, or, as a next step, to be able to identify different kinds of tumours automatically.

5. REFERENCES

- [1] Instagram @ONLINE, accessed April 7, 2015. <https://instagram.com/press/>.
- [2] G. Baatz, O. Saurer, K. Köser, and M. Pollefeys. Large Scale Visual Geo-Localization of Images in Mountainous Terrain. In *Computer Vision - ECCV 2012*, Lecture Notes in Computer Science, pages 517–530. 2012.
- [3] L. Baboud, M. Čadík, E. Eisemann, and H.-P. Seidel. Automatic Photo-to-terrain Alignment for the Annotation of Mountain Pictures. In *Proc. of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 41–48, 2011.
- [4] R. Fedorov, P. Fraternali, and M. Tagliasacchi. Mountain Peak Identification in Visual Content Based on Coarse Digital Elevation Models. In *Proc. of the 3rd ACM International Workshop on Multimedia Analysis for Ecological Data*, MAED '14, pages 7–11, 2014.
- [5] P. C. N. Jr, M. Mukunoki, M. Minoh, and K. Ikeda. Estimating Camera Position and Orientation from Geographical Map and Mountain Image. In *38th Research Meeting of the Pattern Sensing Group, Society of Instrument and Control Engineers*, pages 9–16, 1997.
- [6] E. J. Keogh, L. Wei, X. Xi, S.-H. Lee, and M. Vlachos. LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures. In *VLDB*, pages 882–893, 2006.
- [7] J.-F. Lalonde, S. G. Narasimhan, and A. A. Efros. What Do the Sun and the Sky Tell Us About the Camera? *International Journal of Computer Vision*, 88(1):24–51, 2010.
- [8] W.-N. Lie, T. C.-I. Lin, T.-C. Lin, and K.-S. Hung. A robust dynamic programming algorithm to extract skyline in images for navigation. *Pattern Recognition Letters*, 26(2):221–230, 2005.
- [9] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *J Intell Inf Syst*, 39(2):287–315, 2012.
- [10] L. Porzi, S. R. Buló, P. Valigi, O. Lanz, and E. Ricci. Learning Contours for Automatic Annotations of Mountains Pictures on a Smartphone. In *Proc. of the International Conference on Distributed Smart Cameras*, ICDSC '14, pages 13:1–13:6, 2014.
- [11] F. Y. Shih and S. Cheng. Automatic seeded region growing for color image segmentation. *Image and Vision Computing*, 23(10):877–886, 2005.
- [12] L. Tao, L. Yuan, and J. Sun. SkyFinder: Attribute-based Sky Image Search. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 68:1–68:5, 2009.

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 MuSAMA¹ werden Datenschutzkonzepte für die Anfrageverarbeitung in Assistenzsystemen entworfen. Diese werden allerdings nicht *on-top* 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 PArADISE²-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

¹Multimodal Smart Appliance Ensembles for Mobile Applications

²Privacy-Aware Assistive Distributed Information System Environment

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 Bei-

spiels. 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 abhandeln, 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, \dots, R_n zerlegt. Für jede Partition R_i wird eine Selektionsbedingung c_i angegeben, welche die Tupel aus R den Partitionen zuordnet:

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

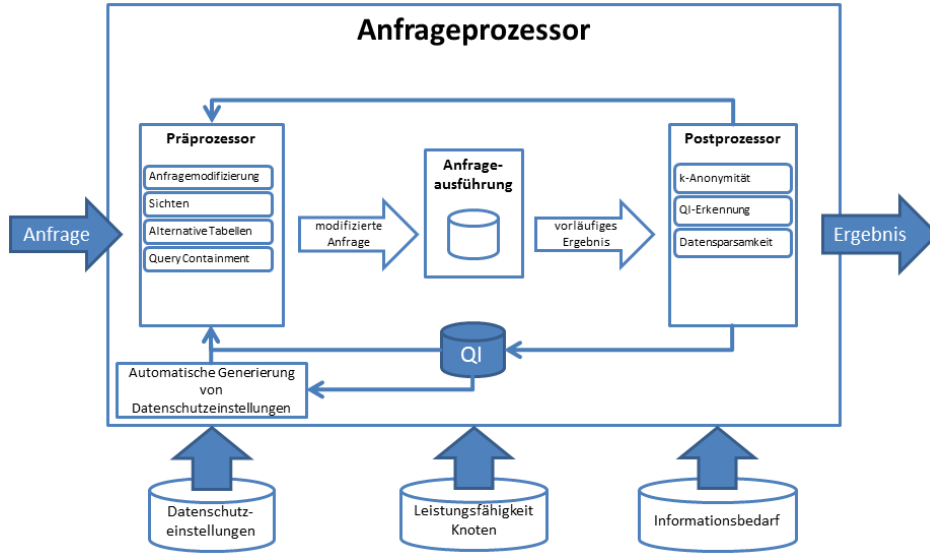


Abbildung 1: Das Konzept des datenschutzfreundlichen Anfrageprozessors

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 $\pi_{\text{attributmeng}_j}(R_i)$ 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:

$$R_{i1} := \pi_{\text{attributmeng}_1}(R_i), \dots, R_{im} := \pi_{\text{attributmeng}_m}(R_i). \quad (2)$$

Die Attributmengen in den Projektionen müssen dabei nicht zwangsweise disjunkt sein⁴. Sie dürfen keine identifizierende Attributmengung 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

³ v_j, v_k numerische Werte

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

auf unterschiedliche Partitionen zu verteilen. Die Ermittlung von identifizierenden Attributmengen, sogenannten Quasi-Identifikatoren [2], lässt sich mit geringem Aufwand [5] vor der Anonymisierung realisieren.

3.3 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:

$$L_{ij} := \tau_{\langle \text{Attribute} \rangle}(R_{ij}). \quad (3)$$

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:

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

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 *<attriblist>* 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 @2GHz und 4MB Cache) und 4GB 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 2,3GHz und 6MB Cache verfügten. Als Arbeitsspeicher standen 8GB 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 geparkt 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 abschließenden $Union$ -Operationen.

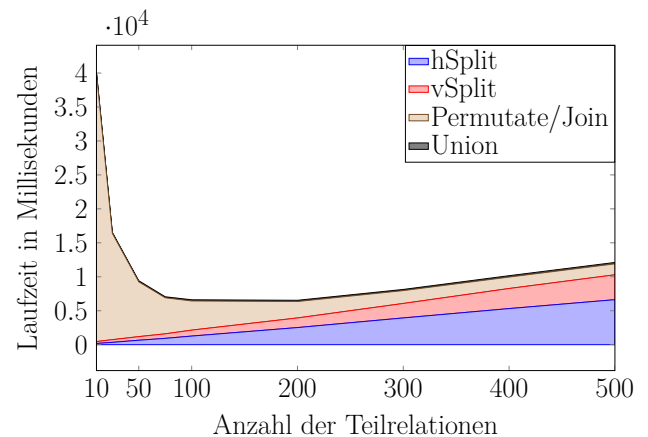


Abbildung 7: Betrachtung der Laufzeit des Slicing-Algorithmus für unterschiedlich große Partitionen. Bei großen Partitionen nimmt der Aufwand für die Permutation der einzelnen Teilrelationen drastisch zu. Kleinere Partitionen benötigen eine geringere Laufzeit für die Permutation; das Erstellen der Teilrelationen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

$$y_i = \alpha + \beta * x_i + \epsilon \quad (6)$$

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 stellen 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 Saygin. 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.

Towards Visualization Recommendation – A Semi-Automated Domain-Specific Learning Approach

Pawandeep Kaur

Heinz-Nixdorf Chair for Distributed
Information Systems
Friedrich-Schiller-Universität, Jena

pawandeep.kaur@uni-jena.de

Michael Owonibi

Heinz-Nixdorf Chair for Distributed
Information Systems
Friedrich-Schiller-Universität, Jena

michael.owonibi@uni-jena.de

Birgitta Koenig-Ries

Heinz-Nixdorf Chair for Distributed
Information Systems
Friedrich-Schiller-Universität, Jena

birgitta.koenig-ries@uni-jena.de

ABSTRACT

Information visualization is important in science as it helps scientists in exploring, analysing, and presenting both the obvious and less obvious features of their datasets. However, scientists are not typically visualization experts. It is therefore difficult and time-consuming for them to choose the optimal visualization to convey the desired message. To provide a solution for this problem of visualization selection, we propose a semi-automated, context aware visualization recommendation model. In the model, information will be extracted from data and metadata, the latter providing relevant context. This information will be annotated with suitable domain specific operations (like rank abundance), which will be mapped to the relevant visualizations. We also propose an interactive learning workflow for visualization recommendation that will enrich the model from the knowledge gathered from the interaction with the user. We will use biodiversity research as the application domain to guide the concrete instantiation of our approach and its evaluation.

Categories and Subject Descriptors

D.2.12 [Data mapping]

General Terms

Human Factors, Design

Keywords

Data Visualization, Machine Learning, Biodiversity Informatics, Text Mining, Recommender Systems

1. INTRODUCTION

The human brain can comprehend images a lot easier than words or numbers. This makes effective graphics an especially important part of academic literature [19]. Visualization that condenses large amounts of data into effective and understandable graphics is therefore an important component of the presentation and communication of scientific research [14]. Supporting scientists in choosing the appropriate visualization during the research process is very important. We believe that an optimal choice leads to more

interpretable graphics which keep the reader interested in the publication, and make them understand the research work and possibly build on it. Ultimately, this results in increased citation of such publications. In addition, it aids researchers in detecting recurring patterns, formulating hypotheses and discovering new knowledge out of those patterns [24].

In this paper, we will focus on the issue of visualization selection for data presentation and will be using biodiversity research as an application domain. In the next section, we will first explain the biodiversity research domain and then analyze the challenges and requirements of researchers with respect to the visualization selection. Then, we will present the literature review of the existing solutions (Section 3). In Section 4, we will present our approach to address to the challenges that we have identified.

2. REQUIREMENTS ANALYSIS

Biodiversity research aims to understand the enormous diversity of life on earth and to identify the factors and interactions that generate and maintain this diversity [20]. Biodiversity data is the data accumulated from the research done by biologists and ecologists on different taxa and levels, land use, and ecosystem processes. For proper preservation, reusability, and sharing of such data, metadata is provided along with the data. This metadata contains vital contextual information related to the datasets like purpose of the research work, data collection method and other important keywords. In order to answer the most relevant questions of biodiversity research, synthesis of data stemming from integration of datasets from different experiments or observation series is frequently needed. Collaborative projects thus tend to enforce centralized data management. This is true, e.g., for the Biodiversity Exploratories [16], a large scale, long-term project funded by DFG. The Exploratories use the BExIS platform [15] for central data management. The instance of BExIS used within the Biodiversity Exploratories (BE) serves as one of the primary sources for collecting requirements for this study. The large collection of data available in the BE BExIS is the result of research activities by many disciplines involved in biodiversity science over the last eight years. This data is highly complex, heterogeneous and often not easy to understand. To interpret, analyze, present, and reuse such data a system is required that can analyze and visualize these datasets effectively.

According to the survey of 57 journals conducted in [21], natural science journals use far more graphs than mathematical or social science journals [21]. The objective of any graphics in the context of scientific publications and presentations is to effectively communicate information [19]. For that, it is important to choose

the appropriate visualization with respect to available data and message to convey. However, the studies [23] have shown that the potential of visualization has not been fully utilized in scientific journals. In [23] Lauren et al identify two main reasons for this failure: scientists are overwhelmed by the numerous visualization techniques available and they lack expertise in designing graphs.

In general, a visualization process is considered as a ‘search’ process in which the user makes a decision about visualization tools and techniques at first, after which other decisions are made about different controls like layout, structure etc. until a satisfactory visualization is produced [13]. With the growing amount of data and increasing availability of different visualization techniques this ‘search’ space becomes wider [13]. In order to successfully execute this search process, one needs to have clear knowledge about the information contained in the data, the message that should be conveyed and the semantics of different visualizations.

We argue here that to understand this complex process and then work aptly, one needs to have some visualization expertise. However, scientists typically do not have the proficiency to manipulate the programs and design successful graphs [22]. Interactive visualization approaches make the visualization creation process more adaptive, but, due to their insufficient knowledge, scientists often have difficulties in mapping the data elements to graphical attributes [12]. The result of inappropriate mapping can impede analysis and even result in misleading conclusions [1].

Furthermore, matters related to visualization are made even more complex by human perception subjectivity [9], which means people perceive the same thing differently under different circumstances. For better understanding, readers primarily need to relate the visualization to the realm of their existing knowledge domain [2]. To ensure that the chosen visualization does indeed convey the intended message to the target readers, a model like the one proposed in [6] should be the base of visualization design.

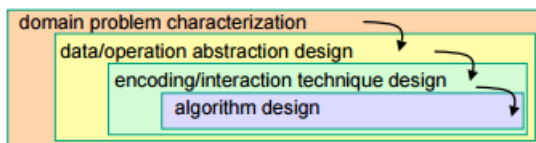


Figure 2. Nested Model for Visualization Design [6]

This model, as depicted in Figure 2, divides the design process into four levels which are: 1) characterize the tasks and data in the vocabulary of the problem domain, 2) abstract this information into visual operations and data types, 3) design visual encoding and interaction techniques, and lastly create algorithms to execute these techniques efficiently.

An approach as depicted in model above, needs to rely on the domain knowledge and visualization used in that domain. In Section 4 we will propose such an approach.

3. STATE OF THE ART

The literature on visualization recommendation can be found from the early Eighties of the last century. The earliest such work is BHARAT [25], APT [26], Vis-WIZZ [27], Vista [28] and ViA

[29]. BHARAT, APT and VIA have a similar direction: They all aim at encoding the data variables to the visual clues, human perception analysis, exploit the knowledge of graphic designs and displays. Such work was independent of any domain. Vis-WIZZ and VISTA have noticed the need of knowledge accelerated visualization mapping techniques, but their research is limited to numerical or quantitative data. Casner’s BOZ system [5] analyses task descriptions to generate corresponding visualizations. However, the task first needs to be fed manually to the mapping engine. Many Eyes [10] by IBM which uses the rapidly adaptive visualization engine (REVA) based on the grammar of graphics by Leland Wilkinson [11] is an example of commercial approaches in this area. Similarly, Polaris’ work on Visual query language (VISQL) is used in the Show_me data module of the Tableau [17] software. Both of these approaches do not consider contextual information for recommending visualization.

PRAVDA (Perceptual Rule-Based Architecture for Visualizing Data Accurately) [4] introduced a rule based architecture for assisting the user in making choices of visualization color parameters. The appropriate visualization rule is selected based on higher-level abstractions of the data, i.e., metadata. They were the first who introduced knowledge from the metadata into the visualization process.

Current knowledge-based visualization approaches are highly interactive [3] and use semantics from different ontologies to annotate visual and data components (see, e.g., Gilson et al [8]). They extract the semantic information from the input data and try to find the best match by mapping three different ontologies, where one is the domain ontology, another is the visualization ontology and the last one is their own ontology which is created by mapping first two.

Though knowledge-based systems reduce the burden placed upon users to acquire knowledge about complex visualization techniques, they lack expert knowledge [13]. Such solutions should be based on some ground truth collected from relevant domain experts. Additionally, we argue that limited user interaction to obtain feedback would be useful to enhance the knowledge base.

4. PROPOSED APPROACH

Based on the requirements identified in Section 2 and the shortcomings of existing approaches discussed in Section 3, we propose a visualization recommendation model which will help scientists in making appropriate choices for presenting their data.

It will be based on a knowledgebase created by reviewing the visualizations presented in biodiversity publications. Such knowledge will enrich our understanding on current trends in visualizations for representing biodiversity data. It will also enhance the system with scientific operations and concepts and variables used in the presenting those concepts. We will be extracting information from metadata (which contains a description of various characteristics of the data and the context of the data collection and usage), integrating the knowledge from the domain vocabularies, and classifying this information with respect to the visual operations performed on the dataset. The knowledge obtained in this way will serve as a key parameter in recommending visualization.

In addition, to deal with the problem of human perception subjectivity, we propose an interactive machine learning approach for visualization recommendation. We will track the input from the user at each interaction and will update that into the respective modules in mapping engine. This will make system learn from the user interaction. However, users will be only prompted to interact in case they do not get satisfactory results. Thus for a non-computer experts (biodiversity researchers in our case) the interaction would be nil, if his choice of result is present in our recommended list.

In general, our approach is made up of two main components namely: the Visualization Mapping Model (Section 4.1) and the Interactive Learning workflow (Section 4.2). The approach is explained using the metadata of a dataset from the BE BExIS as an example [30].

4.1 Visualization Mapping Model

Figure 3 provides an overview of the visualization mapping model. Each of the five phases identified and marked on the figure will be explained in detail below.

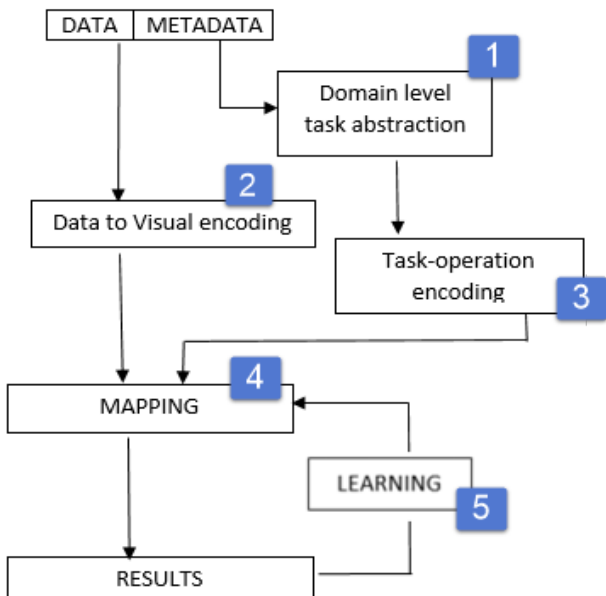


Figure 3: Visualization Mapping Model

1) **Domain level task abstraction:** Task here refers to domain specific analytic operations which are computed on several variables of the dataset in order to derive a concept. For instance, species distribution is an ecological concept which is about computation of distribution (a task) of some species over a geographical area.

To understand the domain problem well, first, we need to understand the dataset, the goals of the data collection, the analysis performed on the data and how these analytical operations can be mapped visually. Metadata provides information about the what, why, when, and who about data and context, methodology, keywords related to dataset and research. Extracting this information from the data and metadata and mapping it with the domain specific vocabularies can reveal the biodiversity related tasks that can be performed on the dataset. As

an example, consider the excerpt of metadata of a specific dataset from the BE BExIS [30] shown here:

Detection of forest activities (harvesting / young stock maintenance, etc.) of the forest owner (Forest Service) on the EPs of exploratories. amount and spatial distribution of forest harvesting measures by the Forest Service on the EPs.

To keep it short and precise, just one keyword (“spatial distribution”) has been extracted and will be analyzed and processed. By annotating it with terms from an ontology, e.g., the SWEET ontology [7], a relation such as shown in Fig. 4 has been found.

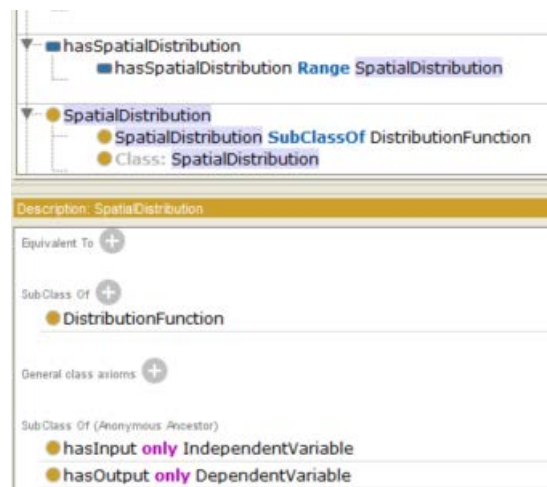


Figure 4 : Keyword Annotation

This annotation can be explained as:

Domain Specific Task: Spatial Distribution on any of the distribution functions like Probability Distribution Function or Chisquare Distribution.

Representational Task : Distribution

Representational Variables: atleast 2 (independent and dependent)

2) **Data to visual encoding:** Here, we will perform visual encoding on the variables and the values of the dataset. We will map the data variables to their equivalent visual marks/icons/variables (as in Figure 5 [18]) on the basis of some existing classification scheme (such as the one presented in [17]) for graphical presentation. Figure 5 shows how the relationship among various aspects of data can be represented within the visualization. For example, the variable that represent different size elements (like area or length) of some entity could be best represented via bars of different sizes in a visualization.

To give an example, we have used the same dataset as above and have extracted some variables as shown in Table 1. In the visualization creation process, first, the variables are identified with their respective datatypes (measurement units). This we have done and have appended another column as UNIT. Then, by taking a reference from Figure 5, we have transformed these variables into their respective visual icons/variables (shaded column named ‘Visual Icons’ in the figure). Trees species is a ordinal or categorical variable thus could be best represented as colour, shape or orientation styles. In the same way nominal variables could be used as X,Y scales in a 2-D visualization

In the next steps, we will be using these icons to represent the relations between variables in the visualization.

Table 1 : Dataset variables

name	description	Units	Visual Icons
Tree	Tree species shortcuts	Ordinal	Shape, Color, Orientation
NRderMas snahme	consecutive number of measures	Nominal	Position, Size, Value
Tree Height	Tree Height	Quantitative	Position, Size, Value
PLOT	Number of experimental plots	Nominal	Position, Size, Value

Position: changes in the x,y location	
Size: change in length, area or repetition	
Shape: infinite number of shapes	
Value: changes from light to dark	
Colour : changes in hue at a given value	
Orientation: changes in alignment	
Texture: variation in 'grain'	

Figure 5: Bertin's Visual Variables [18]

3) Task to operation encoding: At this stage, we will combine the information from the conceptual knowledge gained from metadata and the visual representation knowledge. Visual representation knowledge will be derived by analyzing existing publications. We will be creating a domain knowledgebase about visualizations used in biodiversity publications and will ask scientists to verify it and provide their feedback. This phase is important to get the domain expertise about current visualization trends for representations of different studies. The candidate visualization will be chosen from this knowledgebase according to the domain tasks that we have extracted in Step 1.

In our preliminary work, we have tried to understand the different visualizations used in biodiversity research by reviewing the publications from the information system of the Biodiversity Exploratories. A small sample of the results is depicted in Figure 6. This figure shows what visualization has been used to represent which biodiversity study/analysis within the reviewed publications. Taking the same example as in the previous steps, with the information contained in Figure 6 we can accomplish two jobs: First, we can infer concepts that are related to the identified concept. For example, spatial distribution can be associated with other spatial analysis methods like Spatial Heterogeneity, Spatial

Autocorrelation etc. Distribution itself relates to various concepts like Trajectory Distribution, Diversity Distribution, PFT Distribution. Second, we can identify related visualizations. In our example, these are Grid Heatmap, Kriged Map and Line Graph.

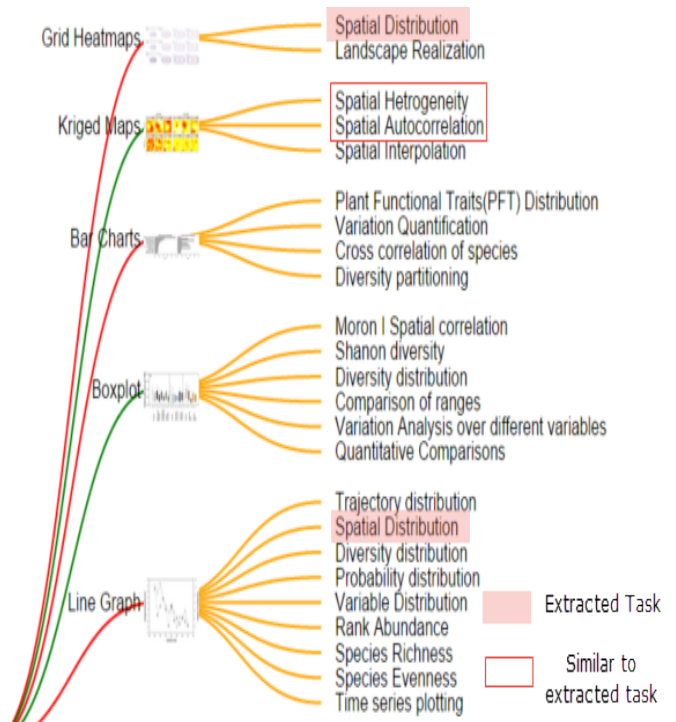


Figure 6: Sample Visualizations used in Biodiversity Research

4) Mapping: Our mapping model is an algorithm that will integrate the knowledge from the previous stages. This algorithm will generate a visualization recommendation list based on the priority of domain specific tasks and feedback from users on the results. The following tasks will be performed:

- It will use the knowledge from previous steps to understand and define the structure of the visualizations appropriate for this dataset.

In our example, in Step 1 we have understood that the task to perform is 'Distribution' with the use of some distribution function for 'Spatial Analysis'. From Step 2, we have identified three candidate visualizations.

- It will integrate the knowledge from Step 2, to map the data attributes within the candidate visualizations.

For example if the user selects 'heatmap', then a possible mapping of variables to visual icons are depicted in Table 3 (consider Table 1 and Figure 5 also)

- It will score/rank the candidate visualizations based on: Review Phase (Step 3): By choosing the candidate visualizations most popular for that study.

System learning: Based on user's feedback as introduced in Section 4.2 below.

Table 2 : Variable attribute mapping

Visual Icons	Variables
X axis	PLOT
Y axis	NRderMassnahme
Value	Tree Height
Colour	Tree

5) Learning: If the user is not satisfied with the results of our automatic mapping process, he or she will be presented with an interactive workflow which will be explained in detail in the next section and which will improve future system suggestions.

4.2 Interactive Learning Workflow

We believe that trying to fully automate the task of visualization recommendation is an extremely difficult area. Classical machine learning approaches, in which the system can be trained on visualization mapping for different domain concepts, might be an option. However, this is an expensive process as it takes tremendous effort in gathering knowledge about the domain (especially for wide domain areas like ours) and then takes a long time to train the model on the huge database. Moreover, this approach is not user centric. Therefore, we suggest the use of interactive machine learning approaches to overcome these problems. Algorithms used in the mapping process can be continuously refined, by training them from the logs of user interaction.

Such an interactive learning workflow is presented in Figure 7.

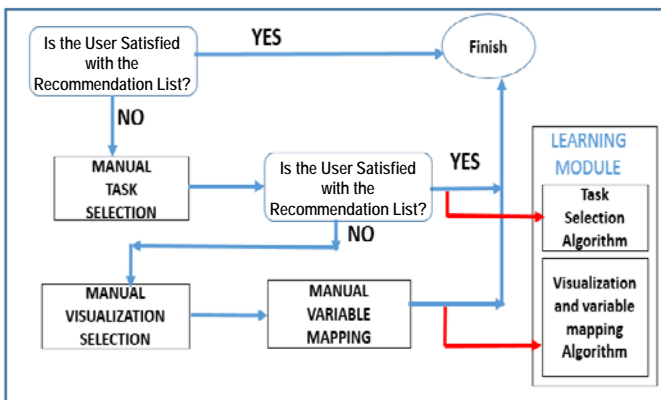


Figure 7: Interactive learning workflow

The learning aspect of the visualization will be triggered in three different cases: In the first case, if the user is satisfied with the list of recommended visualizations, the system will consider it as a *hit* case. Every *hit* case will trigger the following actions:

- 1) The weight parameter will be increased for that recommended list
- 2) Within the list, the visualization that a user selected will score higher than the other visualizations. Returning to the example from Section 4.1, consider the identified visualization list and the respective probability of the visualizations to be selected. Initially,

all will have the same probability of being chosen by the user. For example, given the following visualizations:

- o Grid Heatmap: 33.33 %
- o Krig map: 33.33 %
- o Line graph: 33.33 %

Suppose now the user has selected “Line Graph”. Then it will rank higher in the list and will have a higher probability to get selected. As, it is here:

- o Line graph: 66.66 %
- o Grid Heatmap: 16.67 %
- o Krig map: 16.67 %

The second case is, if the user is not satisfied with the list of recommended visualizations. We will consider this as a *miss* case. Then, we need to know why the intended visualization (i.e., the visualization that a user wants) could not be generated. Therefore, we will ask the user to do a manual task selection. When the user has selected the task, a new visualization list will be recommended. If that is a *hit* case, then we will update our semantic algorithm (which we used in Step 3 of Section 4.1) with this task. In other words, we will make our algorithm consider this task (that the user has selected), when similar context (metadata) is encountered next time. This has been depicted via red lines in Figure 7. Now if the user is still not satisfied with the result, or it is a *miss* case again, then we know that the problem is not with the task extraction algorithm, but with something else. So, we will ask the user to select the visualization and variables. The selected visualization will be updated in the list (from the publication review phase, which we used in Step 3 of Section 4.1), with the corresponding variables that the user has selected.

5. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an approach to semi-automatic visualization recommendation. It is based on understanding the problem domain and capturing knowledge from the domain vocabularies. We are certain that this will assist users (biodiversity researchers in our case) in making suitable choices of visualization from the recommended list without needing to get into any technical details. We have also presented an interactive learning workflow that will improve the system from the users' feedback in case the recommended visualizations are not suitable for them. This will make the system more human centric by inculcating knowledge from different viewpoints, which will produce more effective and interpretable graphics.

Our work is in its initial stages and we are in the process of gathering the visualization requirements from the domain experts via surveys and publications review. This knowledge will be used as a ground truth for mapping the conceptual knowledge to the visual operations.

6. REFERENCES

- [1] Grammel,L., Troy,M., and Story,M. 2010. How information visualization novices construct visualizations. *IEEE transactions on visualization and computer graphics*,16(6).943-952.DOI= 10.1109/TVCG.2010.164
- [2] Amar, R. and Stasko, J. 2004. A Knowledge Task-Based Framework for Design and Evaluation of Information

- Visualizations. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS '04)*. IEEE Computer Society, Washington, DC, USA, 143-150. DOI=10.1109/INFOVIS.2004.10
- [3] Martig, S., Castro, S., Fillotrani, P. and Estévez, E. 2003. Un Modelo Unificado de Visualización. *Proceedings, 9º Congreso Argentino de Ciencias de la Computación*. Argentina. 881-892
- [4] Bergman, L.D., Rogowitz, B.E. and Treinish L.A. 1995. A rule-based tool for assisting colormap selection. In *Proceedings of the 6th conference on Visualization '95 (VIS '95)*. IEEE Computer Society, Washington, DC, USA, 118-125
- [5] Stephen M. C., 1991. Task-analytic approach to the automated design of graphic presentations. *ACM Trans. Graph.* 10, 2 (April 1991), 111-151. DOI=10.1145/108360.108361
- [6] Munzner T. 2009. A Nested Model for Visualization Design and Validation. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (November 2009), 921-928. DOI=10.1109/TVCG.ma2009.111
- [7] NASA JPL California Institute of Technology. Semantic Web for Earth and Environmental Technology (SWEET) version 2.3. Available at <https://sweet.jpl.nasa.gov/download>
- [8] Gilson, O., Silva, N., Grant, P.W. and Chen, M. 2008. From web data to visualization via ontology mapping. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC conference on Visualization (EuroVis'08)*, Eurographics Association, Aire-la-Ville, Switzerland, 959-966. DOI=10.1111/j.1467-8659.2008.01230.x
- [9] Rui, Y., Huang, T.S., Ortega, M. and Mehrotra, S. 1998. Relevance Feedback: A Power Tool for Interactive Content-Based Image Retrieval. *IEEE Transactions On Circuits and Systems for Video Technology*, 8(5).1-13
- [10] IBM. Many Eyes. Available at <http://www-969.ibm.com/software/analytics/manyeyes>
- [11] Wilkinson, L. *Statistics and Computing, The Grammar of Graphics*. Springer Press. Chicago, 2005
- [12] Heer, J., Ham, F., Carpendale, S., Weaver, C. and Isenberg, P. 2008. Creation and Collaboration: Engaging New Audiences for Information Visualization. In *Information Visualization, Lecture Notes In Computer Science*, 4950. 92-133. DOI=10.1007/978-3-540-70956-5_5
- [13] Chen, M., Ebert, D., Hagen, H., Laramee, R.S., Liere, R.V., Ma, K.L., Ribarsky, W., Scheuermann, G., and Silver, D. 2009. Data, Information, and Knowledge in Visualization. *IEEE Computer Graphics Application*, 29(1). 12-19. DOI=10.1109/MCG.2009.6
- [14] Ware, C. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, San Francisco, CA. 2000
- [15] Lotz, T., Nieschulze, J., Bendix, J., Dobbermann, M. and König-Ries, B. 2012. Diversity or uniform? Intercomparison of two major German project databases for interdisciplinary collaborative functional biodiversity research. *Ecological Informatics*, 8, 10-19 DOI=10.1016/j.ecoinf.2011.11.004
- [16] Fischer, M., Boch, S., Weisser, W.W., Prati, D. and Schoning, I. 2010. Implementing large-scale and long-term functional biodiversity research: The Biodiversity Exploratories. *Basic and Applied Ecology*, 11(6).473-485. DOI=10.1016/j.baae.2010.07.009
- [17] Tableau. Available at <http://www.tableau.com/products/trial?os=windowsbertin> book
- [18] Bertin, J. 1983. *Semiology of graphics*. University of Wisconsin Press, Berlin.
- [19] Kelleher, C., Wagener, T. 2011. Ten guidelines for effective data visualization in scientific publications, *Environmental Modelling & Software*. 1-6. DOI=10.1016/j.envsoft.2010.12.006
- [20] The Biodiversity Research Centre, University of British Columbia. Available at <http://www.biodiversity.ubc.ca/research/groups.html>
- [21] Cleveland, W.S. 1984. Graphs in Scientific Publications. *The American Statistician*. 38(4). 261-269. DOI=10.1080/00031305.1984.10483223
- [22] Schofield, E.L. 2002. Quality of Graphs in Scientific Journals: An Exploratory Study. *Science Editor* 25(2). 39-41
- [23] Lauren, E.F., Kevin, C.C. (2012). Graphs, Tables, and Figures in Scientific Publications: The Good, the Bad, and How Not to Be the Latter, *the Journal of Hand Surgery*, 37(3). 591-596, DOI=10.1016/j.jhssa.2011.12.041
- [24] Reda, K., Johnson, A., Mateevitsi, V., Offord, C., & Leigh, J. (2012). Scalable visual queries for data exploration on large, high-resolution 3D displays. In *High Performance Computing, Networking, Storage and Analysis (SCC)*. IEEE. 196-205. DOI=10.1109/SC.Companion.2012.35
- [25] Gnanamgari S. 1981. Information presentation through default displays. Ph.D. dissertation, Philadelphia, PA, USA
- [26] Mackinlay J. 1986. Automating the design of graphical presentations of relational information. *ACM Transactions of Graph*, 5(2), 110-141. DOI=10.1145/22949.22950
- [27] Wehrend S. and Lewis C. 1990. A problem-oriented classification of visualization techniques. In *Proceedings of the 1st conference on Visualization '90 (VIS '90)*, Arie Kaufman (Ed.). IEEE Computer Society Press, Los Alamitos, CA, USA, 139-143.
- [28] Senay, H. and Ignatius, E. 1994. A Knowledge-Based System for Visualization Design. *IEEE Computer Graphics and Applications*. 14(6), 36-47. DOI=10.1109/38.329093
- [29] Healey C. G., Amant R. S., and Elhaddad M. S.. 1999. Via: A perceptual visualization assistant, In *28th Workshop on Advanced Imagery Pattern Recognition (AIPR-99)*, pp.2-11.
- [30] Biodiversity Exploratories Information System (BEXIS). Available at <https://www.bexis.uni-jena.de/Data/ShowXml.aspx?DatasetId=4020>. Accessed on 07/05/2015

Transparente Datenbankunterstützung für Analysen auf Big Data

Dennis Marten
Universität Rostock, Institut für Informatik
18051 Rostock
dm@informatik.uni-rostock.de

Andreas Heuer
Universität Rostock, Institut für Informatik
18051 Rostock
ah@informatik.uni-rostock.de

Zusammenfassung

Es ist kein Geheimnis, dass in den letzten Jahren in der Entwicklung und Forschung unterschiedlichster Bereiche ein enormer Anstieg an genutzten Datenmengen zu beobachten ist. Grund hierfür sind hauptsächlich neue technische Möglichkeiten, wie beispielsweise bessere oder günstigere Sensortechnik. Diese unzähligen Datenmengen bieten den Entwicklern und Forschern oftmals unfassbare Möglichkeiten, führen aber auch teilweise zu neuen Problemen. So werden oftmals Programme für Auswertungen genutzt, die nicht für enorm große Datensätze entwickelt wurden und demnach entweder deutlich zu langsam sind oder nicht funktionsfähig. Ein gutes Beispiel hierfür ist das open-source Programm R [14], welches für sein umfangreiches Repertoire statistischer Methoden, wie beispielsweise Machine-Learning-Algorithmen, bekannt und beliebt ist. R's Performance jedoch hängt maßgeblich mit der Datengröße zusammen. Steigt diese über die Größe des Hauptspeichers, fällt die Rechengeschwindigkeit drastisch ab. Aus diesem Grund stellen wir in diesem Artikel unseren Framework für eine transparente Datenbank- und Parallelisierungsunterstützung von R vor. Hierbei legen wir großen Wert auf die Nutzerfreundlichkeit, d.h. in diesem Fall, dass der Nutzer seine gewohnte Entwicklungsumgebung (etwa sein R-Skript) nicht zu ändern braucht und trotzdem die Vorteile paralleler Berechnung, sowie die In- und Output-Fähigkeiten von Datenbanksystemen ausnutzen kann.

ACM Klassifikation

H.2.4 [Database management]: Systems; H.2.8 [Database management]: Database applications

Schlagworte

Performance

Stichworte

Big Data, R, Datenbanksysteme, Parallel Computing, Machine-Learning

1. EINLEITUNG

Durch das Voranschreiten der Technik ist in den letzten Jahren ein enormer Anstieg genutzter Datenmengen in Entwicklung und Forschung zu beobachten. Dies eröffnete einige Möglichkeiten und Herausforderungen, die teilweise durch konventionelle Ansätze nicht mehr realisierbar sind. Dies hat in den letzten Jahren zu einem Hype in der Entwicklung neuer Umgebungen oder Programmen geführt, die gezielt auf große Datenverarbeitung ausgelegt sind. Ein gutes Beispiel hierfür ist das MapReduce Framework [7] mit seinem wohl populärsten Stellvertreter Hadoop [2]. Der Einschlag dieser Umgebung war enorm und führte zu zahlreichen weiteren Projekten, die auf ihr aufbauten. Jedoch muss auch hier anerkannt werden, dass nicht jedes „Big Data Problem“ mit MapReduce gelöst werden kann, bzw. nicht jeder Entwickler die Expertise hat es in MapReduce zu lösen. Letzteres ist ein Punkt den wir in diesem Artikel betonen wollen. Die gewohnten Entwicklungsumgebungen, das sind beispielsweise Programme für statistische Auswertungen, sind in den wenigsten Fällen solche, die für große Datenverarbeitung ausgelegt sind. Wir beziehen uns in diesem Artikel auf die open-source Programmiersprache R, welches für ihr umfangreiches Repertoire statistischer Methoden bekannt ist und eine äußerst große und stetig wachsende Community besitzt. Das Problem, dem viele R-Entwickler begegnen ist das Einbrechen der Performance bei der Behandlung von Daten, welche die Hauptspeichergöße überschreiten. Es gibt natürlich Wege diesem Problem entgegenzutreten, wie beispielsweise die Hardware zu verbessern, auf Performance-verbessernde R-Erweiterungen zurückzugreifen oder ganz auf ein neues Programm oder Tool umzusteigen. Alle diese Lösungen sind, falls sie ihr eigentliches Ziel überhaupt erreichen, mit einigen Nachteilen, wie materiellen Kosten oder erhöhtem Zeitaufwand durch Einarbeitung in neue Tools und Sprachen, verbunden. Dies stellt den Einstiegspunkt des hier vorgestellten Projektes dar: Ein Framework das R mit Datenbank- und Parallelisierungstechnologie ausstattet. Hierbei wird der R-Code analysiert und in SQL- und parallelisierbare Segmente zerlegt. Einer der Hauptaspekte liegt hierbei auf der transparenten Umsetzung, das heißt, dass der Nutzer nur sein R-Skript benötigt und keine Kenntnisse von SQL oder von Parallelisierungstechniken haben muss.

Der Rest des Artikels ist folgendermaßen gegliedert: In Abschnitt 2 werden Forschungsprojekte und Entwicklungen vorgestellt die entweder R und Datenbanksysteme oder R und paralleles Rechnen vereinigen. In Abschnitt 3 stellen wir einige Aspekte unseres eigenen Projektes vor, wobei die Architektur und Schlüsselcharakteristiken und die Wahl eines

geeigneten Datenbanksystems diskutiert werden. Im letzten Abschnitt beschreiben wir künftige Teilprojekte, die wir in Angriff nehmen werden.

2. STAND DER FORSCHUNG

Das Verbinden von R mit Datenbank- oder Parallelisierungstechnologie ist sicherlich keine neue Idee. Es gibt zahlreiche verschiedene Projekte, die genannt werden sollten.

Die R-Community selbst ist sehr lebendig und hat in den letzten Jahren verschiedene Erweiterungen („Pakete“) hervorgebracht. Datenbankverbindungen mittels allgemeinem JDBC-Treiber oder für spezielle Datenbanksysteme wie MySQL, PostgreSQL und ähnliche sind alle verfügbar und einfach handhabbar. Allerdings eröffnen diese Anbindungen oftmals nur eine Schnittstelle um SQL-Anfragen in R durchzuführen. Eine deswegen sehr interessante Erweiterung ist das `dplyr`-Paket [6], welches eine automatische Übersetzung einiger, simpler R-Methoden in SQL ermöglicht.

MonetDB [8] nutzt diese, um ein automatisches Prefiltering mittels SQL zu ermöglichen, welches die Datenmenge die R verarbeiten muss deutlich reduzieren kann. Zusätzlich bietet MonetDB eine Möglichkeit R-Methoden in seine SQL-Anfragen einzubetten. In dieser Verbindung zwischen dem Datenbanksystem und R wird zusätzlich ein geteilter Hauptspeicheransatz verfolgt, welcher beiden Schnittstellen eine schnelle und günstige Kommunikation erlaubt.

Ein weiteres Datenbankprojekt das hervorzuheben ist, ist RIOT-DB [17], welches MySQL und R verbindet. Hier werden Variablen zwischen R und MySQL mittels *Mapping* verbunden, welche mittels überladener Operatoren in R bearbeitet werden. Das Projekt konzentriert sich dabei hauptsächlich auf die Optimierung von Matrix-Operationen.

Einen anderen Ansatz schlägt `paradigm4`'s SciDB [13] vor. SciDB ist ein Datenbanksystem, welches im Gegensatz zu typischen spalten- oder zeilenorientierten Datenbanksystemen auf einem Array-Datenmodell basiert. Dieses Modell soll sich bei wissenschaftlichen Berechnungen äußerst günstig auf die Rechenzeit auswirken. SciDB unterstützt, ähnlich wie RIOT-DB, eine durch *Mapping* realisierte Verbindung der Schnittstellen und eine, durch überladene Operatoren in R realisierte, Verarbeitung.

Als Zwischenfazit lässt sich sagen, dass mehrere Projekte für Datenbankenverbindungen mit R bereits existieren, welche eine gute Basis für weitere Forschung bieten. Ein Punkt der hierbei meist vernachlässigt wird ist die Übersetzung von komplexeren R-Funktionen, welche ein Schlüsselaspekt unseres Framework sein wird. Zudem werden für einige dieser Projekte SQL-Kenntnisse des Nutzers vorausgesetzt oder zumindest das manuelle Aufbauen von Verbindungen und Nutzen neuer vorgefertigter Methoden.

Wie auf der Datenbankseite, existieren auch für paralleles Rechnen zahlreiche R-Pakete die den Nutzern etwa Multi-Core-Verarbeitung (siehe etwa das `pR`-Paket [9]) oder sogar eine Schnittstelle zu Hadoop [2] ermöglicht.

Als eine Alternative zu R sollte IBM's SystemML [1, 10] genannt werden. Dieses arbeitet mit einer R-ähnlichen Sprache und führt MapReduce-Strategien für zahlreiche Operationen ein. Besonders an SystemML ist der Fokus auf die numerische Stabilität von Algorithmen, welche gerade bei einer großer Anzahl von algebraischen Operationen oftmals leidet und so Rundungsfehlereinflüsse deutlich verstärken kann.

Weitere nennenswerte Verbindungen von R und MapReduce sind RHIPE [5], Ricardo [15] und Cumulon [3]. Schließ-

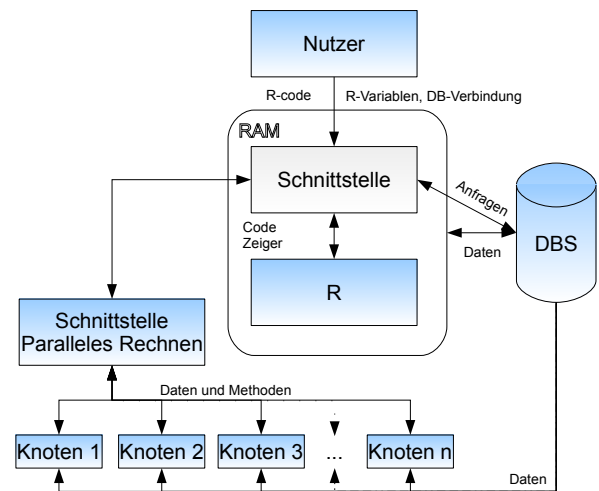


Abbildung 1: Architektur des vorgestellten Frameworks.

lich sei der kommerzielle Vertreter Revolution Analytics' „Revolution R“ mit seinem ScaleR-Paket genannt. Dieses bietet eine breite Reichweite an automatischen Übersetzungen von R-Funktionen für parallele Plattformen, wie beispielsweise Hadoop-Cluster.

Unser Projekt wird sich von diesen Entwicklungen zum einen durch das Nutzen von Datenbank- und Parallel Computing-Technologie und zum anderen durch den Fokus des *automatischen Übersetzens* von bereits existierenden, komplexen R-Methoden unterscheiden. Damit soll erreicht werden, dass Nutzer ohne Änderungen ihrer R-Skripte parallele Verarbeitung nutzen können. Die einzig nötige Anpassung sollte eine Zuordnung von Datenbank- und R-Variablen sein.

3. ARCHITEKTUR

In diesem Abschnitt stellen wir unsere Architektur vor, die in Abbildung 1 dargestellt ist. Das Augenmerk dieser, ist die eigens implementierte Schnittstelle, die verschiedene Funktionen inne hat. Sie ist mit R, der Parallelisierungsschnittstelle und dem Datenbanksystem verbunden und ist demnach zur Verwaltung der Kommunikation, welche in den nächsten Unterabschnitten näher beschrieben wird, verantwortlich. Es sollte außerdem hervorgehoben werden, dass zudem keine der anderen vier Hauptschnittstellen mit einer anderen verbunden ist. Weiterhin wird hier ein Ansatz des geteilten Hauptspeichers zwischen R und der Schnittstelle verfolgt, welcher eine günstige Kommunikation ermöglicht um beispielsweise Zwischenergebnisse zu bearbeiten. Ursprünglich war ein klassischer MapReduce-Ansatz vorgesehen, wobei jedoch das Senden der Daten zur Schnittstelle und der darauffolgenden erneuten Aufteilung zu unnötigen Kosten geführt hätte. Daher werden die Knoten zur parallelen Berechnung selbst mit der Datenbank verbunden um unnötiges Senden von Daten zu meiden und so optimal die Vorteile aller drei Welten nutzen zu können.

Im Folgenden werden Einblicke in den Ablaufprozess gegeben und Vor- und Nachteile des Ansatzes diskutiert.

3.1 Konzeptuelle Charakteristiken

In diesem Abschnitt werden verschiedene zu implementierende Charakteristiken der Zielumgebung erläutert.

3.1.1 Input

Eines unser Hauptbedenken gilt der Nutzerfreundlichkeit, da ein Einarbeiten in neue Programme oder Techniken für Nutzer sehr zeitintensiv und frustrierend sein kann. Demnach sollte nur das gewohnte R-Skript übergeben werden und zusätzlich eine Tabelle, die die Beziehung der Variablen im R-Skript und der Datenbank beschreibt.

3.1.2 Wahl der Plattform

Es ist sicherlich nicht sinnvoll jede Methode direkt auf einem Cluster ausführen zu lassen. Viele verschiedene Faktoren spielen bei der Wahl, ob eine Funktion auf einem Cluster, lokal parallel oder vielleicht nur auf der Datenbank selbst ausgeführt werden sollte, eine Rolle. Schlüsselfaktoren sind etwa die Clustergröße mit Hauptspeicher und Prozessordaten, die Netzwerkgeschwindigkeit, die zu verarbeitende Datengröße, der genutzte Algorithmus und welche Teilprobleme überhaupt parallelisierbar sind. Ein hilfreicher Ansatz ist das Aufstellen von Kostenfunktionen, die diese Aspekte mit einbeziehen und bei der Wahl Hilfe leisten können. Da bei den meisten der geplanten Methoden keine iterativen Probleme mit Abbruchbedingung enthalten sind, ist die Herleitung in vielen Fällen unkompliziert.

Der R-Code wird anfangs durchlaufen und ein Ablaufplan erstellt, welcher einzelne Methoden einzelnen Plattformen zuordnet. Bei der Erstellung des Plans werden komplexere (unterstützte) R-Methoden in Kombinationen einfacher, algebraischer Operationen und Aggregationsfunktionen aufgespalten. Für diese grundlegenden Operationen können dann einfache Parallelisierungsstrategien eingesetzt werden, wie beispielsweise bei algebraischen oder distributiven Aggregationsfunktionen. Ist ein Ablaufplan erstellt, liegen in günstigen Fällen keine Datenabhängigkeiten vor, sodass die Datenbank, der Cluster und der Hauptknoten (etwa R) gleichzeitig arbeiten können und so der Leerlauf einzelner Instanzen minimiert wird.

3.1.3 Geschachtelte Anfragen und lazy-SQL-Auswertung

Ein weiterer, wichtiger Ansatz den wir verfolgen ist das *lazy* Auswerten von SQL-Anweisungen, was unweigerlich zur Verarbeitung geschachtelter SQL-Anfragen führt. Diese Methodik hat zur Folge, dass der Hauptspeicher von eventuell unnötigen Daten verschont bleibt. Eines der erstaunlich häufig vorkommenden Beispiele aus der Praxis ist das Einlesen von CSV-Dateien (Comma-Separated-Values), welche keine vorgefilterten Read ermöglichen. Im konventionellen R-Code würde zwangsläufig die gesamte Datei eingelesen, auch Daten die vielleicht niemals benötigt würden.

Ein anderes Beispiel sind Sequenzen von Berechnungen in denen nach und nach Daten gefiltert werden. Können hier Anfragen an die Datenbank verzögert und miteinander verbunden werden, entstehen geschachtelte SQL-Anfragen, deren Ergebnismenge mitunter deutlich geringer sein kann. Dadurch wird unnötiges Senden von Daten und deren Aufnahme in den Hauptspeicher vermindert. Ferner ist davon auszugehen, dass die Datenbankoptimierer mit solchen, ty-

pischerweise einfachen, Schachtelungen problemlos umgehen können und diese intern entschachteln.

3.2 Architekturdiskussion

In diesem Abschnitt wird das Potenzial der Architektur näher erläutert. Dazu werden einige Schlüsselfragen gezielt beantwortet.

3.2.1 Warum nicht ein paralleles Datenbanksystem benutzen?

Parallele Datenbanksysteme werden seit Jahrzehnten entwickelt und gepflegt. Sie enthalten demnach viele hochentwickelte Techniken für paralleles Rechnen. In [12] wird gezeigt, dass solche Systeme es vermögen auch MapReduce-Programme teilweise deutlich in den Schatten zu stellen. Es wird aber auch darauf eingegangen, dass solche Systeme typischerweise sehr teuer und äußerst schwierig zu installieren, beziehungsweise optimal zu konfigurieren sind.

Einer der Hauptkritikpunkte an parallelen Datenbanksystemen ist die starke Abhängigkeit der Performance bezüglich der Aufteilung der Daten. Das Problem hierbei ist, dass verschiedene Algorithmen von verschiedenen Partitionierungen profitieren. Jedoch ist es im Vorfeld oftmals nicht absehbar welche Methoden auf den Daten angewendet werden oder es ist bekannt, aber es werden viele verschiedene Methoden genutzt. Dies macht eine günstige Verteilung in vielen Fällen nicht möglich.

Ein weiterer Aspekt paralleler Datenbanksystemen ist, dass diese typischerweise Inter- anstatt Intraoperatorparallelrechnung unterstützen, welche momentan nicht in unserem Fokus liegt.

Außerdem gilt es ebenfalls zu bedenken, dass parallele Datenbanksysteme nicht für komplexere, nicht (auf einfache Weise) SQL-darstellbare Methoden geeignet sind.

3.2.2 Warum nicht MapReduce mit R benutzen?

Datenbanksysteme zu integrieren hat wesentliche Vorteile, die solch eine Umgebung bereichern können. Hier seien einige genannt:

- Die Verwaltungsstärke eines DBMS; dies beinhaltet die Reduktion von Redundanz, paralleles Arbeiten auf aktualisierten Daten, die In- und Output-Fähigkeiten des DBMS, ...
- Die Verbesserung des Dateizugriffes der Clusterknoten
- Vorfilterung von Dateien durch Selektion oder Berechnung von Methoden auf der Datenbank ...

Einer der Gründe warum Datenbanken MapReduce-Umgebungen drastisch übertreffen *können* ist die Analyse der Daten beim Einleseprozess. Eine strukturierte Datenanordnung, inklusive Indexstrukturen für schnellere Berechnungen, sind zwei der Hauptvorteile. Es gibt natürlich viele Wege MapReduce bezüglich dieser Aspekte zu optimieren [7], was jedoch sehr viel Zeit in Anspruch nimmt und für ein transparentes System nicht unbedingt vorteilhaft ist.

Ob parallele Datenbanksysteme oder MapReduce genutzt wird; viele Probleme lassen sich durch spezifische Systeme und spezifische Konfigurationen sehr schnell lösen. Aber die entscheidende Frage ist, inwiefern es sich lohnt für jedes solcher Probleme die „optimale“ Konfiguration zu suchen,

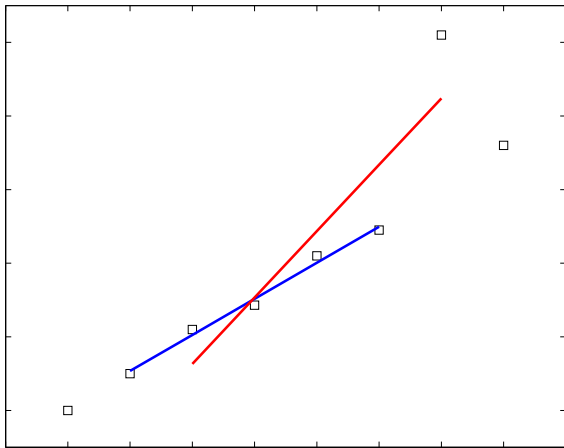


Abbildung 2: Beispielhafte Darstellung der Anstiegsänderung der linearen Regressionsgeraden bei Ausreißern.

anstatt möglicherweise lieber eine etwas schlechtere Performance in Kauf zu nehmen, dafür jedoch kein Expertenwissen und viel Zeit investieren zu müssen.

Als Fazit sollte gesagt werden, dass es nicht immer sinnvoll ist das Rad neu erfinden zu wollen, sondern durchaus auch das Nutzen jahrzehntelanger Forschung und Expertise von Datenbanksystemen sinnvoll ist.

3.3 Wahl des Datenbanksystems

In diesem Abschnitt wird die Suche und Auswahl eines geeigneten Datenbanksystems näher erläutert. Prinzipiell ist jedes gängige Datenbanksystem, welches SQL unterstützt, eine mögliche Wahl für solch eine Aufgabe. Jedoch sind sicherlich gravierende Performance-Unterschiede zwischen einzelnen Systemen zu erwarten.

Zur Entscheidungshilfe wurden einige Experimente in unserer Forschungsgruppe durchgeführt. Zunächst galt es jedoch eine adäquate Methode zu finden, die (zumindest teilweise) auf SQL darstellbar ist, eine gewisse Komplexität aufweist und außerdem eine signifikante Bedeutung für wissenschaftliches Rechnen besitzt. Im Rahmen des Graduiertenkollegs *MuSAMA* (Multimodal Smart Appliance Ensembles for Mobile Applications) [www.musama.de] fand diesbezüglich ein informativer Austausch statt. Der Forschungsschwerpunkt des Kollegs liegt in der Untersuchung und Entwicklung von Modellen und Methoden für Assistenzsysteme. Hierfür wurden beispielsweise Experimente durchgeführt, die Bewegungsdaten mittels xsens-motion-tracking für typische Bewegungsmuster in einer Küche aufgenommen haben. Mit diesen Daten sollen durch gezieltes Anwenden von Machine-Learning-Algorithmen, Rückschlüsse auf momentane Handlungen des Testsubjektes gemacht werden. Eine der Kernmethoden in solchen statistischen Analysen ist die *lineare Regression*. Diese tritt bei der Erstellung linearer Modelle in statistischen Rechnungen häufig auf, besitzt aber auch andere wichtige Einsatzfelder. Ein weiteres Einsatzgebiet ist beispielsweise die Bestimmung von Ausreißern in Datensätzen. Hierbei werden Regressionsgeraden über kleinen Fenstern berechnet. Geht das Verschieben des Fensters um einen Datenpunkt mit einer signifikanten Änderung des Anstiegs der Regressionsgerade einher, wurde im Falle von glatten Daten wahrscheinlich ein Ausreißer gefunden (siehe Abbildung 2).

Sei nun zur formalen Beschreibung der linearen Regression $x, y \in \mathbb{R}^n$ Sensordaten, welche voneinander in etwa linear abhängig sind, das heißt

$$y \approx \alpha x + \beta$$

mit $\alpha, \beta \in \mathbb{R}$. Die gesuchten Koeffizienten werden dann durch die Minimierung der Quadrate der Residuen berechnet:

$$\left(\sum_{i=1}^n (y_i - (\alpha x_i + \beta))^2 \right) \rightarrow \min_{\alpha, \beta \in \mathbb{R}}$$

Die Koeffizienten sind dann durch

$$\alpha = \frac{\sum_{i=1}^n x_i^2 \sum_{i=1}^n y_i - \sum_{i=1}^n x_i \sum_{i=1}^n x_i y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2}$$

$$\beta = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2}$$

beschrieben und können in SQL durch

```
SELECT ((COUNT(x) * SUM(x*y)) - (SUM(x)*SUM(y)))/
(COUNT(x) * SUM(x*x)-SUM(x) * SUM(x)) AS alpha,
AVG(y) - (COUNT(x) * SUM(x*y) - SUM(x)*SUM(y))/
(COUNT(x) * SUM(x*x)-SUM(x)*SUM(x)) * AVG(x) as
beta
FROM data
```

oder mit analytischen Funktionen durch

```
SELECT COVAR_POP(x,y)/COVAR_POP(x,x) as beta,
AVG(y) - COVAR_POP(x,y)/COVAR_POP(x,x) *
AVG(x) as alpha
FROM data
```

berechnet werden.

Die iterative Berechnung von Regressionsgeraden in kleinen Fenstern ist deutlich schwieriger, kann aber in SQL:1999 mittels `WITH RECURSIVE` oder `OVER` in SQL:2003 berechnet werden. Der im Experiment genutzte Code kann im Anhang gefunden werden. Dessen Komplexität ist ein gutes Beispiel dafür, weshalb ein automatisches *Parsen* für Entwickler deutlich attraktiver ist, als nur die Möglichkeit SQL in R zu nutzen.

Die vorgestellte Methode wurde nach dem Aufstellen des Konzepts in SQL auf den Plattformen PostgreSQL 9.4.1, DB2 v10.1.0.0 (BLU) und MonetDB v1.7 implementiert und deren Performance untersucht. Hierbei wurde für jedes System ein virtueller Server mit jeweils 2.8 GHz Octacore CPU und 4 GB DDR2 Arbeitsspeicher und dem Betriebssystem CentOS 6.6 genutzt. Gearbeitet wurde auf einer Tabelle mit 4000 Tupeln und 666 Spalten, sowie einer festen Fenstergröße von 5.

Aus Gründen der Platzbeschränkung kann hier keine detaillierte Auswertung vorgenommen werden. Eines der Kernergebnisse ist jedoch in Abbildung 3 dargestellt. Hierbei wurden die bereits erwähnten Systeme und zusätzlich MonetDB's Embedded R, welches das Aufrufen von R in SQL ermöglicht, verglichen. Da MonetDB momentan keine Rekursion unterstützt, wurden in diesem Experiment einzelne Tupel per Hand gruppiert. Alle Ergebnisse liegen jedoch trotzdem weit unter R's Performance, selbst wenn das Einlesen der Daten in R nicht mit eingerechnet wird. Unterschieden wurde hier zudem die Berechnung mittels einfacher

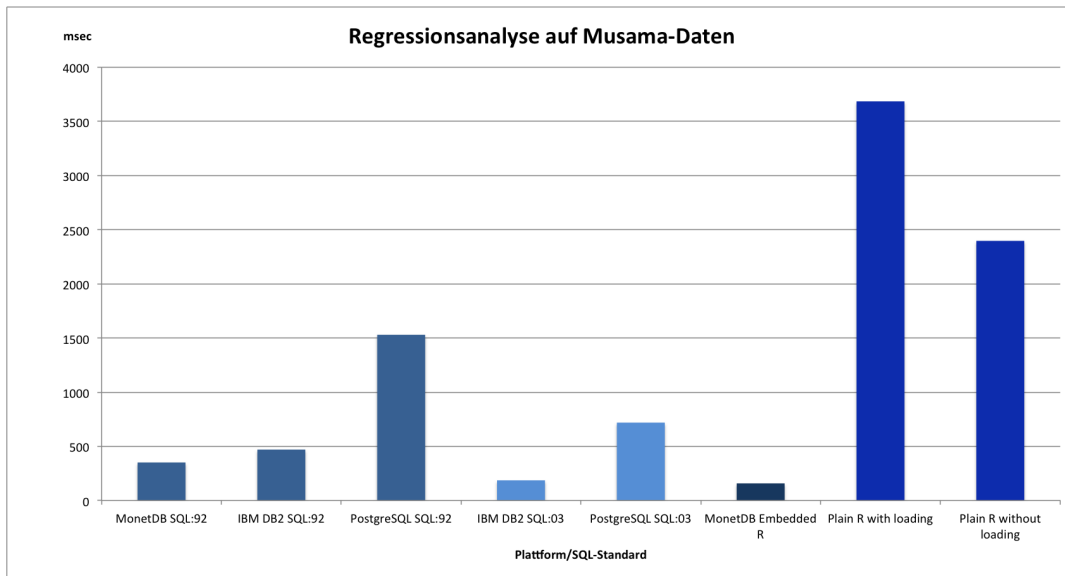


Abbildung 3: Zeitliche Auswertung der Regressionsanalyse ohne Rekursion. Verglichen wurden verschiedene Datenbanksysteme und verschiedene SQL-Standards.

algebraischer Operatoren unter SQL-92 und der Berechnung mittels der Funktionen `regr_slope` und `regr_intercept` in SQL:2003. Letztere führen, wie zu erwarten war, zu einer zusätzlichen Performanceverbesserung.

Unter der zusätzlichen Betrachtung unserer anderen Experimente kann gesagt werden, dass MonetDB die Kontrahenten PostgreSQL und DB2 (BLU) deutlich schlug. Da PostgreSQL eine zeilenorientierte Architektur besitzt, war deren vergleichsweise schlechte Performance im Rahmen unserer Erwartung.

Die schwache Leistung von IBM's DB2 BLU in anderen Experimenten stieß zunächst auf Verwunderung. Weitere Untersuchungen zeigten jedoch, dass einzelne Operationen momentan noch nicht auf die spaltenorientierte Architektur angepasst worden sind und dadurch zusätzliche Konvertierungen in die zeilenorientierte Darstellung die Rechengeschwindigkeit drastisch reduzierten. Aus diesem Grund ist zu diesem Zeitpunkt ein Vergleich mit MonetDB nicht fair.

Wie bereits erwähnt stellte sich heraus, dass spaltenorientierte Architekturen für wissenschaftliche Berechnungen deutlich günstiger sind. Als Vertreter dieser bietet MonetDB jedoch noch zusätzlich ausgeprägte Techniken (siehe etwa [11, 16]) und zeigt in vielen Benchmarks beeindruckende Ergebnisse. Dies macht es zu unserem Datenbanksystem der Wahl.

Abschließend sei erwähnt, dass die oben beschriebene Methode zur Berechnung der Regressionskoeffizienten eine nicht stabile Variante ist und weiterhin auch nicht in dieser Form in R genutzt wird.

Typischerweise wird eine QR-Matrixfaktorisierung der zugehörigen Designmatrix $A \in \mathbb{R}^{n \times 2}$ erstellt um eine stabile Berechnung der Koeffizienten zu ermöglichen. Dafür wird das zugehörige lineare Gleichungssystem

$$Rw = Q^T b$$

gelöst, wobei $w \in \mathbb{R}^2$ die gesuchten Koeffizienten beinhaltet und $b \in \mathbb{R}^n$ die Sensorantworten.

Der von uns durchgeführte Ansatz kann sich jedoch für

kleine Teildatensätze, wie sie bei *Rolling-Window-Methoden* eingesetzt werden, als durchaus günstig erweisen. Trotzdem ist es wichtig stabile Methoden bereitzustellen, etwa durch Einpflegen von Matrixfaktorisierungen und andere Möglichkeiten, wie die Kahan-Summation, da gerade bei der Verarbeitung großer Datensätze Rundungsfehler verheerende Einflüsse haben können.

4. AUSBLICK

Da unser Projekt noch am Anfang steht, ist noch ein Großteil an Forschungspunkten offen. Der unmittelbar nächste Schritt ist das gezielte Aufstellen und Auswerten von Experimenten, die verschiedenste Methoden auf R, der Datenbank und dem Cluster berechnen lassen. Dabei konzentrieren wir uns auf komplexere Methoden wie das Aufstellen linearer Modelle (1m). Dafür müssen weiterhin Matrixfaktorisierungen und Methoden wie die Kahan-Summation auf SQL implementiert werden. Für solche Methoden werden dann Teilablaufpläne entwickelt und Strategien, diese mit anderen Plänen zu verbinden und optimieren. Hierfür wird das Aufstellen von Kosten- und Schätzfunktion ein weiterer wichtiger Aspekt unserer Forschung sein.

Schließlich werden die Parallelisierungsmethoden auf Schleifen ausgeweitet um fundamentale, statistische Ansätze, wie beispielsweise *leave-one-out Kreuzvalidierung*, zu unterstützen. Solche Ansätze sind besonders günstig für Datenbankunterstützungen, da sie häufig auf dieselben Daten zugreifen und damit für das mühsame Einspeisen der Daten entschädigen.

Das große Ziel unserer Forschung ist es, unsere Umgebung mit einer weiteren Schicht für die Datensicherheit (siehe [4]) zu kombinieren. Diese wird direkt vor der Datenbank platziert um mögliche Verletzungen der Datensicherheit von SQL-Anfragen durch Veränderung dieser zu meiden.

5. SCHLUSSWORT

In dem vorliegenden Artikel wurde unser Projekt, welches

das Statistik-Programm R, Clusterparallelisierung und Datenbanktechnologie vereinigt, vorgestellt. Ein entscheidender Aspekt ist hierbei die gezielte Abarbeitung bestimmter R-Methoden in SQL, wobei dieser Prozess automatisch, ohne manuelles Einwirken des Nutzers, geschehen soll. Für diese Arbeit wurden Vor- und Nachteile diskutiert, aufkommende Probleme beschrieben und ein Ausblick für unsere zukünftigen Teilprojekte gegeben.

6. DANKSAGUNG

Dennis Marten wird durch die Deutsche Forschungsgemeinschaft (DFG) im Rahmen des Graduiertenkollegs 1424 (Multimodal Smart Appliance Ensembles for Mobile Applications - MuSAMA) gefördert.

Für Ausarbeitungen in den oben beschriebenen Experimenten gilt Jan Svacina, Dennis Weu, Stefan Lütke, Pia Wilsdorf, Felix Köppl und Steffen Sachse unser Dank.

7. LITERATUR

- [1] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhvani, S. Tatikonda, Y. Tian, S. Vaithyanathan. SystemML: Declarative Machine Learning on MapReduce. In *2011 IEEE 27th International Conference on Data Engineering*.
- [2] Apache. Hadoop. <http://hadoop.apache.org>.
- [3] B. Huang, S. Babu, J. Yang. Cumulon: optimizing statistical data analysis in the cloud. In *2013 ACM SIGMOD/PODS*.
- [4] H. Grunert. Distributed denial of privacy. In *Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2014)*.
- [5] S. Guha. *Computing Environment for the statistical analysis of large and complex data*. PhD thesis, Purdue University, 2010.
- [6] H. Wickham, R. Francois, RStudio. dplyr: A grammar of data manipulation, 2015.
- [7] J. Dean, S. Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1):66–71, January 2010.
- [8] J. Lajus, H. Mühleisen. Efficient data management and statistics with zero-copy integration. In *International Conference on Scientific and Statistical Database Management*, volume 24, 2014.
- [9] J. Li, X. Ma, S. Yoginath, G. Kora, N. F. Samatova. Transparent runtime parallelization of the R script language. *Journal of Parallel and Distributed Computing*, 2011.
- [10] M. Boehm, S. Tatikonda, B. Reinwald, P. Sen, Y. Tian, D. Burdick, S. Vaithyanathan. Hybrid Parallelization Strategies for Large-Scale Machine learning in systemml. *Proceedings of the VLDB Endowment*, 2014.
- [11] M. G. Ivanova, M. L. Kersten, N. J. Nes, R. A. P. Goncalves. An architecture for recycling intermediates in a column-store. *ACM Trans. Database Syst.*, 35(4):24, 2010.
- [12] M. Stonebraker, D. Abadi, D. J. Dewitt, S. Madden, E. Paulson, A. Pavlo, A. Rasin. MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM*, 53(1):66–71, January 2010.
- [13] Paradigm4. SciDB. <http://www.scidb.org>.
- [14] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.
- [15] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, J. McPherson. Ricardo: Integrating R and Hadoop. In *2010 ACM SIGMOD*.
- [16] S. Idreos, M. L. Kersten, S. Manegold. Self-organizing tuple reconstruction in column-stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 297–308, 2009.
- [17] Y. Zhang, H. Herodotou, J. Yang. RIOT: I/O-efficient numerical computing without SQL. In *4th Biennial Conference on Innovative Data Systems Research*, 2009.

APPENDIX

Im Folgenden ist der SQL-Code eines *Rolling-Window-Ansatzes* zur Berechnung von linearen Regressionen dargestellt. Hierbei wurde das iterierende Fenster durch `WITH RECURSIVE` umgesetzt. Die Fenstergröße wurde im Quelltext auf 5 gesetzt. Je nach Kompatibilität kann der Anstieg mittels SQL:2003-Kommandos (`regr_slope(y,x)`) oder wie bereits erklärt, ausprogrammiert berechnet werden.

```
WITH RECURSIVE retest (n, koeff, anstieg) AS
(
  SELECT 0,
         0::double precision,
         0::double precision
  UNION ALL
  SELECT retest.n+1,
         (
           SELECT koeff
           FROM
           (
             SELECT regr_intercept(y,x) AS koeff
             FROM
             (
               SELECT n AS grpe, time -99999 AS x,
                Segment_RightUpperLeg_Position_Z AS y
               FROM test
               LIMIT 5 OFFSET n
             ) AS sublv12a
             GROUP BY grpe
           ) AS sublv11a
         ),
         (
           SELECT anstieg
           FROM
           (
             SELECT regr_slope(y, x) AS anstieg
             FROM
             (
               SELECT n AS grpe, time -99999 AS x,
                Segment_RightUpperLeg_Position_Z AS y
               FROM test
               LIMIT 5 OFFSET n
             ) AS sublv12b
             GROUP BY grpe
           ) AS sublv11b
         )
  ) FROM retest
WHERE n < (SELECT COUNT(time) FROM test)
) SELECT * FROM retest OFFSET 1;
```

Ansätze zur Erkennung von Kommunikationsmodi in Online-Diskussionen

Matthias Liebeck
Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Universitätsstr. 1
D-40225 Düsseldorf, Deutschland
liebeck@cs.uni-duesseldorf.de

Zusammenfassung

Bei der automatisierten Analyse von Textbeiträgen aus Online-Plattformen erfolgt oft eine Einteilung in positive und negative Aussagen. Bei der Analyse von Textbeiträgen eines kommunalen Online-Partizipationsverfahrens ist eine Aufteilung der geäußerten Meinungen in Kommunikationsmodi sinnvoll, um eine Filterung nach Argumenten und Emotionsäußerungen für nachfolgende Verarbeitungsschritte zu ermöglichen. In dieser Arbeit werden zwei Ansätze zur Erkennung von Kommunikationsmodi vorgestellt. Das erste Verfahren unterscheidet verschiedene Kommunikationsmodi anhand von Wortlisten. Die zweite Methode berücksichtigt Wortarten und extrahiert weitere sprachliche Eigenschaften. Zur Evaluation der Ansätze wird ein Datensatz aus Schlagzeilen von Nachrichtenartikeln der Internetseite ZEIT ONLINE und der Satire-Website Postillon erstellt. Die Ansätze werden zur Erkennung des Kommunikationsmodus Satire eingesetzt. Das beste Ergebnis mit einem durchschnittlichen F_1 von 75,5 % wird durch den zweiten Ansatz mit einer Support Vector Machine erreicht.

Kategorien

I.2.7 [Natural Language Processing]: Text Analysis;
H.3.1 [Information stogare and retrieval]: Text Mining

Schlüsselwörter

Natural Language Processing, Text Mining, Text Analysis, Sentiment Analysis, Opinion Mining, Emotion Recognition, Satire Detection, Postillon

1. EINLEITUNG

Im Internet gibt es viele verschiedene Plattformen, auf denen Meinungen, z. B. über Produkte, Filme oder politische Themen, als Textbeiträge geäußert werden können. Die Benutzer können untereinander Diskussionen führen, in denen sie idealerweise argumentativ ihre Meinungen darlegen. Die

se Textbeiträge können automatisiert analysiert werden, um Meinungsbilder über individuelle Themen zu erstellen.

1.1 Analyse von Online-Diskussionen

Von besonderem Interesse ist die automatisierte Analyse von Online-Partizipationsverfahren, bei denen Bürger die Möglichkeit nutzen, ihre Meinung zu lokalkommunalen Themen zu äußern. Bei einer oft erwünschten, hohen Teilnehmerzahl an Bürgern kann schnell das Problem auftreten, dass die beteiligten Bürger viele Textbeiträge erstellen und dadurch ein hoher Aufwand für eine manuelle Auswertung entsteht. Dieser nicht unerhebliche Arbeitsaufwand kann für Kommunen mit geringem Budget zu dem Problem führen, dass für die Analyse nicht genügend personelle Ressourcen zu Verfügung stehen und eine Analyse durch einen externen Dienstleister finanziell ebenfalls nicht möglich ist.

Eine weitere Schwierigkeit entsteht, wenn in einer Kommune erstmalig ein Online-Beteiligungsverfahren eingesetzt wird und die beteiligten Bürger vermehrt Inhalte äußern, die nicht zu dessen Thema passen. Werden in einem Verfahren beispielsweise gemeinsam Sparmaßnahmen diskutiert, so sind Beiträge, in denen Bürger kostenintensive Baumaßnahmen an der städtischen Infrastruktur vorschlagen, nicht konstruktiv und sollten herausgefiltert werden können.

Um diese Probleme zu reduzieren, sind mehrere automatisierte Schritte denkbar, die zu einer Arbeitsreduktion bei einer qualitativen Analyse führen. Durch diese Arbeitsreduktion kann eine Verwaltung umfassender mit den beteiligten Bürgern über eingereichte Verbesserungsvorschläge diskutieren. Zu diesen automatisierten Schritten gehören die thematische Gruppierung von Textbeiträgen und die themenspezifische Bestimmung einer Tonalität $t \in \{\text{positiv, negativ, neutral}\}$, um ein Stimmungsbild abschätzen zu können. Dadurch kann beispielsweise ermittelt werden, dass sich viele Bürger über eine Parkplatzsituation in einem Stadtteil beschweren und Anpflanzungen neuer Bäume in einem Park befürworten.

Ein üblicher Ansatz zur Bestimmung von Tonalitäten ist der Einsatz eines Tonalitätslexikons, in dem für einzelne Wörter jeweils ein numerischer Tonalitätswert angegeben ist. In [10] wurde gezeigt, dass für das deutsche Tonalitätslexikon SentiWS [14] nur eine geringe Abdeckung für die untersuchten Kommentare aus einem kleineren Online-Partizipationsverfahren und einem Nachrichtenportal erreicht wurde und daher weitergehende Ansätze zur Bestimmung von positiven und negativen Aussagen nötig sind. In dieser Publikation werden daher zwei Ansätze vorgestellt, die eine differenziertere Analyse von Meinungsäußerungen

ermöglichen sollen, indem genauer auf Kommunikationsmodi und geäußerte Emotionen eingegangen wird.

1.2 Kommunikationsmodi

In Online-Diskussionen verwenden die Teilnehmer verschiedene Kommunikationsmodi. Die einzelnen Beteiligten können beispielsweise Aussagen tätigen (1), Argumente für oder gegen einen Standpunkt formulieren (2) oder Emotionsäußerungen zum Ausdruck bringen (3).

- (1) *Ich bin für den Bau eines Schwimmbads.*
- (2) *Es sollte kein Geld für die Oper ausgegeben werden, da unsere Schulen das Geld dringender benötigen.*
- (3) *Die steigende Kriminalitätsrate macht mir Angst.*

Die Emotionsäußerungen können wiederum in verschiedene Emotionen differenziert werden. Bei der Untersuchung von Textbeiträgen aus Online-Partizipationsverfahren sind zunächst die Emotionen $E := \{\text{Freude, Hoffnung, Empörung, Enttäuschung, Angst}\}$ für ein Stimmungsbild der Bürgermeinungen interessant. Eine fundiertere Einteilung wird in zukünftigen Arbeiten durch Experten erfolgen. Die folgenden Beispiele aus einem fiktiven Online-Partizipationsverfahren veranschaulichen die unterschiedenen Emotionen.

- (4) Freude: *Das wäre wirklich schön.*
- (5) Hoffnung: *Ich hoffe, dass an der Hauptstraße neue Bäume gepflanzt werden können.*
- (6) Empörung: *Das gehört doch verboten!*
- (7) Enttäuschung: *Die zur letzten Wahlperiode versprochene Änderung konnte meine Erwartungen nicht erfüllen.*
- (8) Angst: *Ich befürchte, dass meine Buslinie durch diese Änderung eingestellt wird.*

Bei der automatisierten Erkennung von Emotionen variiert die Einteilung der Emotionen je nach Textmaterial. [17] unterscheidet in die sechs Emotionen *anger, disgust, fear, joy, sadness* und *surprise*, wohingegen [8] nur die vier Emotionen *anger, fear, joy* und *sadness* betrachtet.

Der Grund für die Untersuchung von Kommunikationsmodi ist die Arbeitshypothese, dass durch eine automatische Erkennung der Emotionen E ein detaillierteres, themenspezifisches Stimmungsbild angegeben werden kann, als es ein Mittelwert über numerische Tonalitätsangaben ermöglicht. Dazu muss ein Klassifikator K erstellt werden, der jedem Satz eines Textbeitrags individuell eine Emotion oder die Klasse *neutral* zuordnet. Erschwerend bei der Klassifikation ist die subjektive und kontextabhängige Wahrnehmung von Emotionen.

Der Rest dieser Arbeit ist wie folgt aufgebaut: Im nächsten Kapitel werden verwandte Arbeiten vorgestellt. Nachdem in Kapitel 3 zwei Ansätze zur Erkennung von Kommunikationsmodi präsentiert werden, erfolgt in Kapitel 4 eine Evaluation beider Ansätze am Beispiel des Kommunikationsmodus Satire. Anschließend wird in Kapitel 5 ein Fazit gezogen und Ideen für zukünftige Arbeiten angegeben.

2. VERWANDTE ARBEITEN

Der Bereich Sentiment Analysis beschäftigt sich mit der automatisierten Bestimmung von Tonalitäten in Textdokumenten. Übliche Anwendungsgebiete sind die Analyse von Produktrezensionen [7] und Filmrezensionen [12], die in positive und negative Äußerungen kategorisiert werden. Die automatisierte Extraktion von Tonalitäten, bei der einer Aussage eine Tonalität $t \in \{\text{positiv, negativ, neutral}\}$ zuge-

ordnet wird, hat sich für die Analyse von Zeitungsartikeln im Rahmen einer Medienresonanzanalyse [16] bewährt.

Die Erkennung von Emotionen in Texten ist bereits mehrfach [1, 8, 17] untersucht worden. Das Ziel von [1] ist die Erkennung von Emotionen in Märchentexten. Die Autoren fokussieren sich dabei auf die binäre Klassifikation von englischsprachigen Sätzen und untersuchen, ob in einem Satz Emotionen auftreten. Dafür annotieren sie einen Datensatz von 1580 Sätzen aus 22 Geschichten der Gebrüder Grimm, auf dem ein linearer Klassifikator trainiert und evaluiert wird. Zur vektoriellen Repräsentation der einzelnen Sätze verwendet [1] mehrere sprachliche Eigenschaften wie die Verteilung von POS-Tags, Satztlängen, Interpunktionszeichen und mehrere Listen von Wörtern, die auf Emotionen hindeuten.

Eine automatisierte Erkennung von sechs verschiedenen Emotionen erfolgt in [17]. Die Autoren untersuchen mehrere Techniken auf einem Datensatz aus 1000 Schlagzeilen von Nachrichtenartikeln. Für ein Baseline-Verfahren erstellen die Autoren sechs Wortlisten aus *WordNet-Affect* [18], einer um Emotionen annotierte Erweiterung von *WordNet* [11]. In dem Baseline-Verfahren erfolgt die Klassifikation eines Satzes s zu Emotionen durch das Auftreten der einzelnen Wörter aus s in den Wortlisten. In einem fortgeschrittenen Verfahren, das aus einer Kombination aus *Latent Semantic Analysis* [9] und Synonymen aus *WordNet* und *WordNet-Affect* besteht, erreicht [17] einen durchschnittlichen F_1 Wert von 17,57 % als bestes Ergebnis für die Erkennung der sechs Emotionen.

In [3] wird ebenfalls das binäre Klassifikationsproblem der Satire-Erkennung in Zeitungsartikeln behandelt. Dabei berücksichtigen die Autoren zusätzlich den Inhalt der Nachrichtenartikel. Als Datensatz untersucht [3] dabei insgesamt 4000 englischsprachige Zeitungsartikel, von denen 233 Satire beinhalten. Als Basismodell wählen die Autoren ein Bag-of-Words-Modell mit binärer Gewichtung. Eine deutliche Steigerung der Ergebnisse kann mit *Bi-Normal Separation (BNS)* [6] als Gewichtung, dem Nachschlagen von Wörtern in einem Lexikon und einer Google-Suche nach auftretenden Personen und Organisationsnamen erreicht werden. Zur Klassifikation setzt [3] eine lineare Support Vector Machine ein.

3. ANSÄTZE

Im Folgenden werden zwei Ansätze vorgestellt, die für die Erkennung von Kommunikationsmodi in Textbeiträgen aus Online-Partizipationsverfahren naheliegend sind. Beide Ansätze versuchen, die menschliche Erkennung von Emotionen nachzuahmen, indem sie auf die in den Textbeiträgen vorhandenen Wörter, in Form von Signalwörtern und bestimmten sprachlichen Konstruktionen, achten.

Dazu müssen die zu untersuchenden Texte satzweise analysiert werden. Die Eingabetexte werden zunächst durch eine Natural Language Processing Pipeline aufbereitet. Die Anzahl an Verarbeitungsschritten der Pipeline ist von der konkreten Aufgabenstellung abhängig. Für die vorgestellten Ansätze werden insgesamt vier Schritte in der NLP-Pipeline durchgeführt: Mittels eines Tokenizers wird ein Eingabetext in einzelne Wörter zerlegt. Durch einen Sentence Splitter werden die Wörter in Sätze gruppiert. Anschließend werden für jedes Wort ein Part-of-Speech Tag (POS-Tag) bzw. eine Wortart bestimmt und eine Lemmatisierung durchgeführt, durch die für jedes Wort zusätzlich eine Grundform (z. B.

Schwimmbäder → Schwimmbad) angegeben wird.

Der erste Ansatz untersucht, inwiefern bestimmte Schlüsselwörter auf einzelne Kommunikationsmodi oder Emotionen hinweisen. Der zweite Ansatz arbeitet unter der Hypothese, dass eine Korrelation bestimmter sprachlicher Eigenschaften zu einzelnen Kommunikationsmodi auftritt. Dabei werden die verwendeten Wortarten untersucht. Motiviert wird dieser Ansatz dadurch, dass eine positive Emotion beispielsweise mit einer überdurchschnittlichen Anzahl an Adjektiven korrelieren könnte.

In beiden Ansätzen wird jedem Satz mittels eines Klassifikators ein Kommunikationsmodus bzw. eine Emotion oder die Klasse *neutral* zugeordnet. Um beide Ansätze evaluieren zu können, muss ein annotierter Datensatz bzw. ein Korpus als Trainingsmenge verwendet werden, in dem auf Satzebene alle Sätze mit entsprechenden Kommunikationsmodi annotiert sind. Der Korpus wird in eine Trainings- und eine Testmenge aufgeteilt, anhand derer ein Klassifikator trainiert bzw. bewertet werden kann.

3.1 Wortlisten

Der erste Ansatz basiert auf der Annahme, dass das Auftreten bestimmter Wörter mit einem Kommunikationsmodus bzw. einer Emotion korreliert. Das Ziel des Ansatzes ist die Verwendung von Wortlisten, die eine Klassifikation eines Satzes, basierend auf den in ihm enthaltenen Wörtern, ermöglichen. Für den nachfolgenden Satz können die Wörter *Angst* und *verliere* auf die Emotion *Angst* hinweisen:

(9) *Ich habe Angst, dass ich verliere.*

Um diese Erkennung zu automatisieren, können für jede Emotion charakteristische Wörter aus einer Trainingsmenge extrahiert werden. Dazu werden in der Trainingsmenge auftretende Wörter untersucht und in disjunkte Wortlisten eingeteilt. Bei der Konstruktion dieser Wortlisten könnten für Beispiel (9) die Wörter *Angst* und *verliere* als charakteristisch identifiziert werden, falls sie auch in anderen Sätzen der Trainingsmenge auftreten, die ebenfalls mit der Emotion *Angst* annotiert sind. Die größte Schwierigkeit bei diesem Ansatz ist die passende Auswahl der Wörter für die Wortlisten. So muss darauf geachtet werden, keine Wörter zu verwenden, die in allen Klassen häufig vorkommen. Daher bietet es sich an, nur Wörter zu betrachten, die überwiegend in einer Klasse (relative Häufigkeit größer als ein Schwellwert τ) und damit nur selten in anderen Klassen vorkommen.

Wird allein auf den Schwellwert τ geachtet, so entsteht das Problem, dass auch Wörter in die Wortlisten aufgenommen werden, die insgesamt nur selten auftreten. Tritt beispielsweise das Wort *Glück* nur einmal in der Trainingsmenge auf, und zwar in einem mit der Emotion *Empörung* annotierten Satz, so würde das Wort *Glück* in die Wortliste für die Emotion *Empörung* aufgenommen werden, anstatt in die Wortliste der Emotion *Freude*. Um diese Problematik zu vermeiden, bietet sich ein Parameter *supp* an, der angibt, wie häufig ein Wort insgesamt in der Trainingsmenge auftreten muss, bevor es in eine Wortliste eingefügt werden darf.

Durch diese beiden Parameter werden häufig in allen Emotionen auftretende Wörter, wie Artikel und Pronomen, herausgefiltert. Die konkrete Wahl der beiden Parameter kann durch ein Experiment geschätzt werden.

Um einen neuen Satz einer Emotion zuzuordnen, kann jedes Wort w des Satzes in den Wortlisten nachgeschlagen werden. Dem Satz wird diejenige Emotion zugeordnet, für die am meisten Wörter in der jeweiligen Wortliste gefunden

werden.

In einer Modifikation dieses Ansatzes werden nicht die Wörter, sondern deren, durch eine Lemmatisierung bestimmte, Grundformen in Wortlisten geführt bzw. nachgeschlagen.

3.2 Sprachliche Eigenschaften

Der zweite Ansatz basiert auf der Hypothese, dass die verschiedenen Kommunikationsmodi bzw. Emotionen im Text charakteristische sprachliche Merkmale besitzen. Um diese Vermutung auf einem Datensatz zu überprüfen, ist eine Methode M notwendig, die sprachliche Eigenschaften eines Satzes in eine vektorielle Darstellung überführt. Dazu werden aus allen Sätzen einer Trainingsmenge sprachliche Eigenschaften durch M extrahiert. Für einen zu klassifizierenden Satz einer Testmenge werden ebenfalls sprachliche Eigenschaften mittels M extrahiert, die anschließend mit einem Klassifikationsverfahren und einer Distanzfunktion zu einem Kommunikationsmodus zugeordnet werden können.

Eine einfache Annahme ist, dass bestimmte Kommunikationsmodi bzw. Emotionen mit einer bestimmten Verteilung von POS-Tags korrelieren. Als erste vektorielle Modellierung eines Satzes s wird daher für jeden POS-Tag p eines Tagsets die Häufigkeit von p in s angegeben.

Diese vektorielle Darstellung kann um weitere sprachliche Eigenschaften ergänzt werden, die eventuell charakteristisch für eine Emotion sein können, beispielsweise welche Wortart am Satzanfang und am Satzende steht oder welches Interpunktionszeichen (Punkt, Fragezeichen oder Ausrufezeichen) einen Satz beendet. Ferner kann das Auftreten von Negationen oder von verschachtelten Nebensätzen berücksichtigt werden.

4. KOMMUNIKATIONSMODUS SATIRE

Da zum aktuellen Zeitpunkt noch keine ausreichende Datenmenge an Diskussionsbeiträgen aus Online-Partizipationsverfahren vorliegt, werden die in Kapitel 3 beschriebenen Ansätze zur Erkennung von Kommunikationsmodi konkret auf den Kommunikationsmodus Satire angewendet, indem die Erkennung von Satire in Nachrichtenartikeln evaluiert wird. In zukünftigen Arbeiten werden diese Techniken ebenfalls für die Erkennung von Emotionsäußerungen in Online-Partizipationsverfahren erprobt und evaluiert.

Zur Evaluation wird ein Datensatz aus Nachrichtenartikeln zusammengestellt. Basierend auf den Überschriften der Nachrichtenartikel soll das binäre Klassifikationsproblem gelöst werden, ob ein Nachrichtenartikel von der Satire-Webseite *Postillon*¹ stammt oder auf ZEIT ONLINE² veröffentlicht wurde.

4.1 Datensatz

Der zur Satire-Erkennung verwendete Datensatz setzt sich aus den beiden Nachrichtenquellen *Postillon* und *ZEIT ONLINE* zusammen. Die Schlagzeilen der Artikel beider Webseiten wurden jeweils über einen JSON-Webservice heruntergeladen. Für den *Postillon* werden 3650 Artikel aus dem Zeitraum Oktober 2008 bis März 2015 betrachtet. Die beiden nachfolgenden Schlagzeilen sind Beispiele für Satire-Artikel aus dem *Postillon*:

¹<http://www.der-postillon.com>

²<http://www.zeit.de>

- (10) *Sensation! Autobahn-Fahrer entdeckt weitere Fahrbahn rechts neben der Mittelspur*
 (11) *Kölner Dom von Unbekannten über Nacht um 360 Grad gedreht*

Eine genauere Betrachtung der Satire-Artikel hat ergeben, dass bestimmte Artikelformate in regelmäßigen Abständen vorkommen, wie z. B. *Sonntagsfragen* oder *Newsticker*. Diese wurden für die weitere Betrachtung entfernt, um das Klassifikationsproblem zu erschweren. Da einige Artikel mehrfach veröffentlicht wurden, wird von allen Artikeln mit demselben Namen jeweils nur die chronologisch erste Veröffentlichung verwendet. Durch diese Filterungsschritte reduziert sich die Anzahl der zur Verfügung stehenden Postillon-Artikel auf 2260.

Die zweite Klasse des Datensatzes setzt sich aus klassischen Zeitungsartikeln zusammen, die auf ZEIT ONLINE veröffentlicht wurden. Aus den Kategorien Wirtschaft, Gesellschaft, Sport, Wissen und Digital wurden jeweils die 2000 aktuellsten Artikel vor dem Stichtag 1.4.2015 mittels *ZeitOnlineAPISharp*³ heruntergeladen. Für die weitere Verarbeitung wurden aus diesen 10000 Artikeln insgesamt 2260 Artikel zufällig ausgewählt, um einen balancierten Datensatz betrachten zu können. Die beiden folgenden Schlagzeilen stammen aus Artikeln von ZEIT ONLINE:

- (12) *Lehrerverband warnt vor Risiken für Bildung*
 (13) *Energiekonzern verzichtet auf Atomenergie, Kohle und Gas*

Zur nachfolgenden Evaluation werden die Artikel in eine Trainings- und eine Testmenge aufgeteilt. Zum Training werden pro Klasse 1000 Artikel zufällig ausgewählt, sodass das Training auf insgesamt 2000 Artikeln stattfindet. Die Evaluation erfolgt auf der Grundlage der verbleibenden 2520 Artikel.

4.2 Evaluation

Der zusammengestellte Datensatz wird zunächst durch eine NLP-Pipeline aufbereitet: Für jede Schlagzeile erfolgt durch OpenNLP⁴ eine Zerlegung in einzelne Wörter, eine Trennung in Sätze und eine Bestimmung von Wortarten für jedes einzelne Wort. Die ermittelten Wortarten stammen aus dem *Stuttgart-Tübingen-Tagset (STTS)* [15], welches aus 54 verschiedenen Wortarten besteht. Die Lemmatisierung von Wörtern erfolgt durch Mate Tools [2].

4.2.1 Wortlisten

Für die Satire-Erkennung über Wortlisten müssen zwei disjunkte Wortlisten W_P und W_Z erstellt werden, in denen jeweils Wörter vertreten sind, die überwiegend nur in den Schlagzeilen des Postillons bzw. in den Schlagzeilen von ZEIT ONLINE auftreten. Zur Erstellung dieser Wortlisten werden zunächst die einzelnen Wörter als Datengrundlage verwendet. Im Postillon treten 4025 (3583 Lemmata) und in ZEIT ONLINE 3050 (2696 Lemmata) verschiedene Wörter auf. Für die Erstellung der Wortlisten wird zunächst pro Wort bestimmt, wie häufig es in der Trainingsmenge auftritt. Wörter, die weniger als supp mal auftreten, werden ignoriert. Anschließend wird für jedes Wort w berechnet, wie groß die relative Häufigkeit p von w in den Schlagzeilen des Postillons bzw. in den Schlagzeilen aus ZEIT ONLINE ist. Falls $p \geq \tau$ ist, so wird w in die entsprechende Wortliste eingefügt. In einer zweiten Variante werden anstelle der

³<https://github.com/Liebeck/ZeitOnlineAPISharp>

⁴<https://opennlp.apache.org/>

Tabelle 1: Satire-Erkennung mit Wortlisten

τ	Wort		Lemma	
	+ Default		+ Default	
0,55	59,24 %	56,84 %	62,88 %	60,21 %
0,6	61,51 %	57,74 %	64,21 %	60,38 %
0,65	59,23 %	55,73 %	57,59 %	54,03 %
0,7	58,87 %	52,11 %	53,75 %	47,70 %
0,75	52,36 %	46,11 %	50,90 %	45,24 %
0,8	49,02 %	41,52 %	48,13 %	40,97 %
0,85	43,41 %	33,58 %	43,81 %	33,58 %
0,9	42,42 %	30,76 %	42,96 %	30,06 %
0,95	41,39 %	28,77 %	42,04 %	29,30 %
1	41,39 %	28,77 %	42,04 %	29,30 %

Wörter die durch die Lemmatisierung bestimmten Lemmata der Wörter untersucht.

Da bisher noch keine Erfahrungswerte für die Parameterwahl vorliegen, werden die Auswirkungen verschiedener Parameter experimentell bestimmt, indem eine Gittersuche mit $\tau \in \{0.55, 0.6, \dots, 1\}$ und $\text{supp} \in \{3, 4, \dots, 10\}$ durchgeführt wird.

Die Ergebnisse der Satire-Erkennung durch Wortlisten sind in Tabelle 1 als durchschnittlicher F_1 Wert über beide Klassen angegeben, wobei für jeden Wert von τ das jeweils beste Ergebnis angegeben ist, welches durchgehend mit $\text{supp} = 3$ erreicht wird. Bei dem Mehrheitsentscheid des Ansatzes ist es möglich, dass ein Unentschieden vorliegt. Dies bedeutet, dass keines der Wörter einer zu klassifizierenden Schlagzeile in W_P oder W_Z auftritt oder dass ein Gleichstand vorliegt. Für jede Parameterbelegung von τ und supp werden zwei Evaluationen mit einem unterschiedlichen Standardwert $d \in \{\text{Postillon}, \text{ZEIT ONLINE}\}$ zur Auflösung eines Gleichstands durchgeführt, von denen in Tabelle 1 jeweils das schlechtere der beiden Ergebnisse aufgeführt ist. Um die Auswirkungen eines Standardwerts beurteilen zu können, sind in Tabelle 1 ebenfalls die Ergebnisse einer Klassifikation aufgeführt, bei der ein Unentschieden bei einem Mehrheitsentscheid als falsche Klassifikation behandelt wird.

Für die Satire-Erkennung erreicht der erste Ansatz mit den auftretenden Wörtern als Datengrundlage als bestes Ergebnis den Wert 61,51 % für $\tau = 0,6$ und $\text{supp} = 3$. Durch eine Lemmatisierung kann das Ergebnis auf 64,21 % gesteigert werden. Bei einem fixierten Wert für τ und einem steigenden Wert für supp werden die Klassifikationsergebnisse schlechter, da die Größen der Wortlisten entsprechend abnehmen. Für den untersuchten Datensatz verschlechtern sich die Ergebnisse bei einem steigenden τ aus demselben Grund.

4.2.2 Sprachliche Features

In einem ersten vektoriiellen Modell wird jede Schlagzeile durch die absoluten Häufigkeiten der auftretenden POS-Tags aus dem STTS-Tagset repräsentiert. Zum Vergleich wird ein zweites Modell untersucht, in dem die Auswirkungen einer Reduktion der 54 POS-Tags auf die 12 POS-Tags des UTS-Tagsets [13] beobachtet werden. Beide Modelle werden mit den drei Klassifikationsverfahren k-Nearest Neighbors (kNN), Support Vector Machine (SVM) und out-of-place measure [4] evaluiert.

Als SVM-Implementierung wird LIBSVM [5] verwendet. Eingesetzt wird eine soft-margin SVM mit einem RBF-Kernel $K(x, y) = \exp(-\gamma \|x - y\|^2)$. Die für das Training der

Tabelle 2: Satire-Erkennung mit sprachlichen Eigenschaften

Modell	SVM	kNN		Out-of-place
		+ Default		
STTS	73,65 %	71,77 %	71,04 %	67,13 %
STTS, Variante B	75,50 %	73,13 %	72,94 %	—
UTS	71,51 %	74,75 %	74,48 %	60,74 %
UTS, Variante B	73,55 %	75,25 %	75,05 %	—

SVM benötigten Werte für den Strafterm C und für γ werden pro Modell jeweils über eine Gittersuche mittels einer Kreuzvalidierung über die Trainingsmenge bestimmt. Als Konvergenzkriterium der SVM wird $\epsilon = 10^{-3}$ gesetzt.

Für den kNN-Algorithmus werden verschiedene Werte für $k \in \{1, \dots, 12\}$ erprobt. Bei der Bestimmung der nächsten Nachbarn werden jeweils die k nächsten Nachbarn per euklidischer Distanz ermittelt. Sollten mehrere Kandidaten für die Auswahl des k -nächsten Nachbarn p vorhanden sein, so wird die Liste der nächsten Nachbarn um alle Nachbarn erweitert, die zum Anfrageobjekt o denselben Abstand haben wie o zu p . Die Klassenzugehörigkeit erfolgt über einen Mehrheitsentscheid der Klassen aller gefundenen nächsten Nachbarn. Tritt dabei ein Gleichstand auf, so wird ebenfalls ein Standardwert verwendet.

Das out-of-place measure wird gewöhnlich für die Spracherkennung durch N-Gramme eingesetzt. Für die Evaluation wird es für den Vergleich von Verteilungen von POS-Tags verwendet, indem für beide Nachrichtenquellen sogenannte Kategorienprofile bestimmt werden. Ein Kategorienprofil besteht dabei jeweils aus einer nach absoluten Häufigkeiten absteigend sortierten Liste von POS-Tags der jeweiligen Trainingsmenge. Zur Klassifikation einer Schlagzeile wird ein Anfrageprofil mittels derselben Methode berechnet. Die Indexpositionen des Anfrageprofils werden mit den Indexpositionen der Kategorienprofile verglichen. Einer Schlagzeile wird dann diejenige Nachrichtenquelle zugeordnet, zu deren Kategorienprofil der kleinste Abstand zum Anfrageprofil besteht.

Die Modellierung wird in einer Variante B ergänzt, in der weitere sprachliche Eigenschaften als binäre Dimension hinzugefügt werden. Dabei wird berücksichtigt, ob in der Überschrift das erste Wort ein Nomen ist, ob das erste Wort ein Verb ist, ob das letzte Wort ein Verb ist, ob in der Überschrift ein Anführungszeichen vorhanden ist und ob ein Fragezeichen, ein Ausrufezeichen oder ein Komma (jeweils binär) auftritt.

Die Ergebnisse der drei Klassifikationsverfahren sind in Tabelle 2 mit durchschnittlichen F_1 Werten über beide Klassen dargestellt. Das insgesamt beste Ergebnis von 75,5 % erreicht eine SVM mit STTS POS-Tags und Variante B. Das beste Ergebnis des kNN-Algorithmus ist minimal schlechter mit 75,25 %. Die Klassenzugehörigkeit beim kNN-Algorithmus kann in fast allen Fällen per Mehrheitsentscheid bestimmt werden. Interessant zu beobachten sind die Auswirkungen der Reduktion des STTS-Tagsets auf das UTS-Tagset, die je nach Klassifikationsverfahren unterschiedlich sind. Für den kNN-Algorithmus konnte eine Verbesserung der Ergebnisse erzielt werden. Bei den anderen Verfahren verschlechtert sich das Ergebnis im Vergleich zu den STTS-Tags.

5. FAZIT UND AUSBLICK

In dieser Arbeit wurden zwei Ansätze zur Erkennung von Kommunikationsmodi präsentiert. Für die nahe Zukunft ist geplant, beide Ansätze auf Textbeiträge eines Online-Partizipationsverfahrens anzuwenden. Dazu wird ein Codebuch entwickelt werden, mit dem der Datensatz in Bezug auf Kommunikationsmodi annotiert wird. Beide Ansätze werden dann auf das Multiklassenproblem der Erkennung von Emotionsäußerungen transferiert und evaluiert.

Bei der Untersuchung des Kommunikationsmodus Satire wurde gezeigt, dass die beiden Ansätze gute Ergebnisse von bereits 75,5 % erreicht haben. Bei der Evaluation ist aufgefallen, dass das Festlegen eines Standardwerts bei einem Gleichstand für den mit Wortlisten arbeitenden Ansatz eine größere Auswirkung auf die Klassifikationsergebnisse hat, als beim kNN-Algorithmus für die sprachlichen Eigenschaften. Für das binäre Klassifikationsproblem der Satire-Erkennung sind ein Reihe von weitergehenden Untersuchungen möglich. Es könnte untersucht werden, welche Auswirkungen durch die Filterung von Stoppwörtern, durch den Vergleich unterschiedlicher Distanzfunktionen für den kNN-Algorithmus und durch die Verwendung weiterer sprachlicher Eigenschaften entstehen können und ob dadurch die Ergebnisse gegenüber Variante B noch gesteigert werden können.

Die größte Schwierigkeit des ersten Ansatzes ist die Auswahl charakteristischer Wörter. Bei einem Transfer des Ansatzes auf andere Datenquellen werden die Auswirkungen von manuell vorgegebenen Wortlisten untersucht. Insbesondere ist geplant, die Auswirkungen einer Erweiterung der Listen durch Synonyme aus einem Thesaurus zu beobachten. Bei der Generierung der Wortlisten konnten die Ergebnisse durch eine Lemmatisierung verbessert werden. In zukünftigen Arbeiten wird untersucht werden, inwiefern Wiktionary⁵ zur Grundformreduktion eingesetzt werden kann. Außerdem wird der Ansatz für den Umgang mit Negationen erweitert werden.

Für den zweiten Ansatz sind weitere sprachliche Eigenschaften in einer vektoriiellen Darstellung vorstellbar, wie beispielsweise das Auftreten von POS-Tag-Bigrammen. Um bei einer Klassifikation gute Ergebnisse erzielen zu können, ist eine Filterung nach sprachlichen Eigenschaften notwendig, die besonders gut mit den einzelnen Klassen korrelieren. Darüber hinaus wird in zukünftigen Arbeiten untersucht werden, inwiefern beide Ansätze kombinierbar sind. Anstelle eines Mehrheitsentscheids des ersten Ansatzes kann die vektorielle Darstellung für jeden Eintrag einer Wortliste um binäre Dimensionen erweitert werden, die jeweils angeben, ob in dem zu klassifizierenden Satz das entsprechende Wort einer Wortliste auftritt. In nachfolgenden Arbeiten wird untersucht werden, welchen Einfluss eine Verkleinerung des STTS-Tagsets auf das UTS-Tagset auf anderen Datensätzen hat.

⁵<https://de.wiktionary.org/>

6. LITERATUR

- [1] C. O. Alm, D. Roth, and R. Sproat. Emotions from Text: Machine Learning for Text-based Emotion Prediction. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, HLT '05*, pages 579–586. Association for Computational Linguistics, 2005.
- [2] A. Björkelund, B. Bohnet, L. Hafdell, and P. Nugues. A High-Performance Syntactic and Semantic Dependency Parser. In *Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations, COLING '10*, pages 33–36. Association for Computational Linguistics, 2010.
- [3] C. Burfoot and T. Baldwin. Automatic Satire Detection: Are You Having a Laugh? In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, ACLShort '09*, pages 161–164. Association for Computational Linguistics, 2009.
- [4] W. B. Cavnar and J. M. Trenkle. N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.
- [5] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.
- [6] G. Forman. BNS Feature Scaling: An Improved Representation over TF-IDF for SVM Text Classification. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08*, pages 263–270. ACM, 2008.
- [7] M. Hu and B. Liu. Mining and Summarizing Customer Reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04*, pages 168–177. ACM, 2004.
- [8] S. M. Kim, A. Valitutti, and R. A. Calvo. Evaluation of Unsupervised Emotion Models to Textual Affect Recognition. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text, CAAGET '10*, pages 62–70. Association for Computational Linguistics, 2010.
- [9] T. Landauer, P. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25:259–284, 1998.
- [10] M. Liebeck. Aspekte einer automatischen Meinungsbildungsanalyse von Online-Diskussionen. In *Proceedings BTW 2015 - Workshops und Studierendenprogramm*, pages 203–212, 2015.
- [11] G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.
- [12] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs Up?: Sentiment Classification Using Machine Learning Techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10, EMNLP '02*, pages 79–86. Association for Computational Linguistics, 2002.
- [13] S. Petrov, D. Das, and R. McDonald. A Universal Part-of-Speech Tagset. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. European Language Resources Association, 2012.
- [14] R. Remus, U. Quasthoff, and G. Heyer. SentiWS – a Publicly Available German-language Resource for Sentiment Analysis. In *Proceedings of the 7th International Language Resources and Evaluation (LREC'10)*, pages 1168–1171, 2010.
- [15] A. Schiller, S. Teufel, C. Stöckert, and C. Thielen. Guidelines für das Tagging deutscher Textcorpora mit STTS (kleines und großes Tagset). Technical report, Universität Stuttgart, Universität Tübingen, 1999.
- [16] T. Scholz and S. Conrad. Opinion Mining in Newspaper Articles by Entropy-Based Word Connections. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1828–1839. Association for Computational Linguistics, 2013.
- [17] C. Strapparava and R. Mihalcea. Learning to Identify Emotions in Text. In *Proceedings of the 2008 ACM Symposium on Applied Computing, SAC '08*, pages 1556–1560. ACM, 2008.
- [18] C. Strapparava and A. Valitutti. WordNet-Affect: An affective extension of WordNet. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 1083–1086. European Language Resources Association, 2004.

Ausführungspläne und -planoperatoren relationaler Datenbankmanagementsysteme

Christoph Koch

Friedrich-Schiller-Universität Jena
Lehrstuhl für Datenbanken und
Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena

Christoph.Koch@uni-jena.de

Katharina Büchse

Friedrich-Schiller-Universität Jena
Lehrstuhl für Datenbanken und
Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena

Katharina.Buechse@uni-jena.de

KURZFASSUNG

Ausführungspläne sind ein Ergebnis der Anfrageoptimierung relationaler Datenbankoptimierer. Sie umfassen eine Menge von Ausführungsplanoperatoren und unterscheiden sich je nach Datenbankmanagementsystem auf verschiedene Weise. Dennoch repräsentieren sie auf abstrakter Ebene inhaltlich ähnliche Informationen, sodass es für grundlegende systemübergreifende Analysen oder auch Interoperabilitäten naheliegt einen einheitlichen Standard für Ausführungspläne zu definieren. Der vorliegende Beitrag dient dazu, die Grundlage für einen derartigen Standard zu bilden und stellt für die am Markt dominierenden Datenbankmanagementsysteme Oracle, MySQL, SQL-Server, PostgreSQL, DB2 (LUW und z/OS) ihre Ausführungspläne und -planoperatoren anhand verschiedener Kriterien vergleichend gegenüber.

Kategorien und Themenbeschreibung

Database Performance, Query Processing and Optimization

Allgemeine Bestimmungen

Documentation, Performance, Standardization, Languages

Schlüsselwörter

relationale DBMS, Ausführungsplan, Operator, Vergleich

1. EINLEITUNG

In der Praxis sind Ausführungspläne ein bewährtes Hilfsmittel beim Tuning von SQL-Anfragen in relationalen Datenbankmanagementsystemen (DBMS). Sie werden vom Optimierer berechnet und bewertet, sodass abschließend nur der (vermeintlich) effizienteste Plan zur Bearbeitung einer SQL-Anfrage benutzt wird. Jeder Ausführungsplan repräsentiert eine Menge von miteinander verknüpften Operatoren. Zusätzlich umfasst er in der Regel Informationen zu vom Optimierer für die Abarbeitung geschätzten Kosten hinsichtlich CPU-Last und I/O-Zugriffen.

Auf Basis dieser Daten bewerten DBMS-spezifische Werkzeuge wie beispielsweise der für DB2 for z/OS im InfoSphere Optim Query Workload Tuner verfügbare Access Path Advisor [1] die Qualität von Zugriffspfaden und geben Hinweise zu möglicher-

weise kritischen Bereichen wie etwa Sortierungen von umfangreichen Zwischenergebnissen. DBMS-übergreifende Werkzeuge zur Ausführungsplananalyse existieren dagegen kaum. Ausnahmen davon beschränken sich auf Werkzeuge wie Toad [2] oder Aqua Data Studio [3]. Diese können zwar DBMS-übergreifend Ausführungspläne verarbeiten, verwenden dazu allerdings je nach DBMS separate Speziallogik, sodass es sich intern ebenfalls um quasi eigene „Werkzeuge“ handelt.

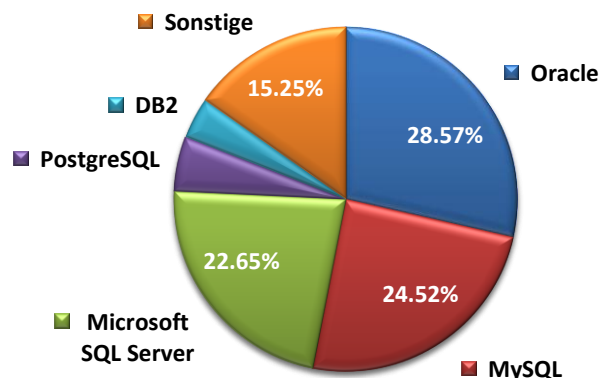


Abbildung 1: Top 5 der relationalen DBMS

Je nach DBMS unterscheiden sich Ausführungspläne und darin befindliche Ausführungsplanoperatoren in verschiedenen Aspekten wie Umfang und Format. Inhaltlich wiederum sind sie sich jedoch recht ähnlich, sodass die Idee einer DBMS-übergreifenden Standardisierung von Ausführungsplänen naheliegt. Der vorliegende Beitrag evaluiert diesen Ansatz. Dazu vergleicht er die Ausführungspläne und Ausführungsplanoperatoren für die Top 5 der in [4] abhängig von ihrer Popularität gelisteten relationalen DBMS (siehe *Abbildung 1*) anhand unterschiedlicher Kriterien. Im Detail werden dabei Oracle 12c R1, MySQL 5.7 (First Release Candidate Version), Microsoft SQL-Server 2014, PostgreSQL 9.3 sowie die bedeutendsten DBMS der DB2-Familie DB2 LUW 10.5 und DB2 z/OS 11 einander gegenübergestellt.

Der weitere Beitrag gliedert sich wie folgt: **Kapitel 2** gibt einen Überblick über die für den Vergleich der Ausführungspläne und Ausführungsplanoperatoren gewählten Kriterien. Darauf aufbauend vergleicht **Kapitel 3** anschließend die Ausführungspläne der ausgewählten DBMS. **Kapitel 4** befasst sich analog dazu mit den Ausführungsplanoperatoren. Abschließend fasst **Kapitel 5** die Ergebnisse zusammen und gibt einen Ausblick auf weitere mögliche Forschungsarbeiten.

2. VERGLEICHSKRITERIEN

Ausführungspläne und –planoperatoren lassen sich anhand verschiedener, unterschiedlich DBMS-spezifischer Merkmale miteinander vergleichen. Der folgende Beitrag trennt dabei strikt zwischen allgemeinen Plan-Charakteristiken und Operator-spezifischen Aspekten. Diese sollen nun näher erläutert werden.

2.1 Ausführungspläne

Der Vergleich von Ausführungsplänen konzentriert sich auf externalisierte Ausführungspläne, wie sie vom Anwender über bereitgestellte DBMS-Mechanismen zur Performance-Analyse erstellt werden können. DBMS-intern vorgehaltene Pläne und die Kompilierprozesse von Anfragen, bei denen sie erstellt werden, sollen vernachlässigt werden. Einerseits sind sie als inhaltlich äquivalent anzusehen, andererseits sind ihre internen Speicherstrukturen im Allgemeinen nicht veröffentlicht. Für den Vergleich von Ausführungsplänen wurden im Rahmen des Beitrags folgende Kriterien ausgewählt.

Erzeugung bezieht sich auf den Mechanismus, mit dem durch den Anwender Zugriffpläne berechnet und externalisiert werden können. Dies sind spezielle Anfragen oder Kommandos.

Speicherung ist die Art und Weise, wie und wo externalisierte Zugriffspläne DBMS-seitig persistiert werden. In der Regel handelt es sich dabei um tabellarische Speicherformen.

Ausgabeformate/Werkzeuge erfassen die Möglichkeit zur Ausgabe von Zugriffsplänen in unterschiedlichen Formaten. Dabei wird zwischen rein textueller, XML- und JSON-basierter, sowie visueller und anderweiter Ausgabe differenziert. Unterstützt ein DBMS ein Ausgabeformat, erfolgt zusätzlich die Angabe des dazu zu verwendenden, vom DBMS-Hersteller vorgesehenen Werkzeugs. Drittanbieterlösungen werden nicht berücksichtigt.

Planumfang gibt Aufschluss über im Zugriffsplan verzeichnete generelle Verarbeitungsabläufe. Dahingehend wird verzeichnet, ob Index- und „Materialized Query Table (MQT)“-Pflegeprozesse sowie Prüfungen zur Erfüllung referentieller Integrität (RI) bei Datenmanipulationen im Zugriffsplan berücksichtigt werden. Mit MQTs sind materialisierte Anfragetabellen gemeint, die je nach DBMS auch als materialisierte Sichten (MV) oder indexierte Sicht (IV) bezeichnet werden.

Aufwandsabschätzung/Bewertungsmaße bezieht sich auf Metriken, mit denen Aufwandsabschätzungen im Zugriffsplan ausgewiesen werden. Es wird unterschieden zwischen CPU-, I/O- und Gesamtaufwand. Sofern konkrete Bewertungsmaße des DBMS bekannt sind, wie etwa, dass CPU-Aufwand als Anzahl von Prozessorinstruktionen zu verstehen ist, werden diese mit erfasst.

2.2 Ausführungsplanoperatoren

Die Vergleichskriterien für die Ausführungsplanoperatoren gehen über einen reinen Vergleich hinaus. Während einerseits anhand verschiedener Aspekte eine generelle Gegenüberstellung von vorhandenen Operatordetails erfolgt, befasst sich ein überwiegender Teil der späteren Darstellung mit der Einordnung DBMS-spezifischer Operatoren in ein abstrahiertes, allgemeingültiges Raster von Grundoperatoren. Dieses gliedert sich in Zugriffs-, Zwischenverarbeitungs-, Manipulations- sowie Rückgabeoperatoren. Sowohl die Operatordetails, als auch die zuvor genannten Operatorklassen werden nachfolgend näher beschrieben.

Operatordetails umfassen die Vergleichskriterien Rows (Anzahl an Ergebniszeilen), Bytes (durchschnittliche Größe einer

Ergebniszeile), Aufwand (in unterstützten Metriken), Projektionen (Liste der Spalten der Ergebniszeile) und Aliase (eindeutige Objektkenung). Es soll abgebildet werden, welche dieser Details allgemein für Ausführungsplanoperatoren abhängig vom DBMS einsehbar sind. Für den Aufwand wird dabei nochmals unterschieden in absoluten Aufwand des einzelnen Operators, kumulativen Aufwand aller im Plan vorangehenden Operatoren (einschließlich dem aktuellen) und einem Startup-Aufwand. Letzterer bezeichnet den kumulativen Aufwand, der notwendig ist, um den ersten „Treffer“ für einen Operator zu ermitteln.

Zugriffsoperatoren sind Operatoren, über die auf gespeicherte Daten zugegriffen wird. Die Daten können dabei sowohl physisch in Datenbankobjekten wie etwa Tabellen oder Indexen persistiert sein, oder als virtuelle beziehungsweise temporäre Zwischenergebnisse vorliegen. Ähnlich dazu erfolgt die Einteilung der Zugriffsoperatoren in tableAccess-, indexAccess-, generatedRowAccess- und remoteAccess-Operatoren. TableAccess und indexAccess bezeichnen jeweils den Tabellenzugriff und den Indexzugriff auf in den gleichnamigen Objekten vorgehaltenen Daten. Unter generatedRowAccess sind Zugriffe zu verstehen, die im eigentlichen Sinn keine Daten lesen, sondern Datenzeilen erst generieren, beispielsweise um definierte Literale oder spezielle Registerwerte wie etwa „USER“ zu verarbeiten. RemoteAccess repräsentiert Zugriffe, die Daten aus externen Quellen lesen. Bei diesen Quellen handelt es sich in der Regel um gleichartige DBMS-Instanzen auf entfernten Servern. Für weitere spezielle Zugriffsoperatoren, die sich nicht in eine der genannten Kategorien einteilen lassen, verbleibt die Gruppe otherAccess.

Zwischenverarbeitungsoperatoren entsprechen Operatoren, die bereits gelesene Daten weiterverarbeiten. Dazu zählen Verbund(join), Mengen- (set), Sortier- (sort), Aggregations- (aggregate), Filter- (filter) und Bitmap-Operatoren (bitmap). Spezielle Zwischenverarbeitungsoperatoren, die sich nicht einordnen lassen, werden in einer separaten Kategorie otherIntermediate erfasst.

Manipulationsoperatoren sind Operatoren, die geänderte Daten in physische oder auch temporäre Datenbankobjekte schreiben. Abhängig von in SQL standardisierten Manipulationsanfragen erfolgt die grundsätzliche Einteilung in die Operatoren insert, update, delete und merge. Zusätzlich werden auch remoteManipulation-Operatoren unterschieden, die analog dem bereits betrachteten remoteAccess eine Datenmanipulation auf einem fernen Server durchführen. Analog den vorangehenden Operatorklassen werden spezielle, nicht kategorisierbare Manipulationsoperatoren unter otherManipulation geführt.

Rückgabeoperatoren umfassen Operatoren, die die Wurzel eines (hierarchischen) Ausführungsplans bilden. Oftmals repräsentieren diese nochmals den Typ des SQL-Statements im Sinne einer Unterscheidung in SELECT, INSERT, UPDATE, DELETE und MERGE.

3. AUSFÜHRUNGSPLÄNE IN RDBMS

Ausführungspläne sind grundlegende Elemente in der (relationalen) Datenbanktheorie. Damit sind sie zwar allgemein DBMS-übergreifend von Bedeutung, unterscheiden sich jedoch im Detail voneinander. So existiert beispielsweise für die im Beitrag gewählten Vergleichskriterien und den daran bewerteten Systemen kein Merkmal, in dem sich alle DBMS in ihren Ausführungsplänen gleichen. Abbildung 2 veranschaulicht die Ergebnisse des Vergleichs, auf die gegliedert nach System nun ausführlicher Bezug genommen wird.

DBMS	Erzeugung	Speicherung	Ausgabeformate/Werkzeuge				Planumfang			Aufwandsabschätzung /Bewertungsmaße			
			TEXT	XML	JSON	andere grafisch	Index- pflege	MQT-/MV- /IV-Pflege	RI- Prüfung	CPU	I/O	Gesamt	
Oracle (12c R1)	EXPLAIN PLAN	PLAN_TABLE	DBMS_XPLAN. DISPLAY, DBMS_XPLAN. DISPLAY_PLAN	DBMS_XPLAN. DISPLAY_PLAN	-	HTML	SQL Developer, Enterprise Manager MySQL Work-bench	-	-	-	proportional zur Anzahl an Maschinen- zyklen	proportional zur Anzahl gelesener Datenblöcke	x
MySQL (5.7)	EXPLAIN	(nur intern)	TRADITIONAL SHOWPLAN_TEXT, STATISTICS PROFILE, SHOWPLAN_ALL (mehr Details)	-	JSON	-	Work-bench	-	-	-	x	x	x
Microsoft SQL-Server (2014)	(stets mit Ausgabe verknüpft) EXPLAIN	(nur intern)	SHOWPLAN_TEXT, STATISTICS PROFILE, SHOWPLAN_ALL (mehr Details)	SHOWPLAN_XML, STATISTICS XML	-	-	Management Studio	x	x	x	x	x	x
PostgreSQL (9.3)	[ANALYZE] VERBOSE]	(nur intern)	TEXT	XML	JSON	YAML	pgAdmin	-	-	-	-	-	x
DB2 LUW (V10.5)	EXPLAIN	tabellarisch (17 Tabellen)	db2exfmt/ db2expln	-	-	-	Data Studio, OQWT	-	x	x	CPU- Instruktionen	Datenseiten- I/Os	Time- rons
DB2 z/OS (V11)	EXPLAIN	tabellarisch (20 Tabellen)	-	-	-	-	Data Studio, OQWT	-	-	-	Millisekunden, Service Units	x	x

Abbildung 2: Vergleich der Ausführungspläne

Oracle [5, 6]: Ausführungspläne werden in Oracle über die SQL-Anfrage EXPLAIN_PLAN **erstellt** und daraufhin innerhalb einer speziellen Tabelle mit der Bezeichnung PLAN_TABLE **gespeichert**. Jede Zeile dieser Tabelle enthält Informationen zu einem Planoperator und bildet deren Kombination im Ausführungsplan über die Spalten ID und PARENT_ID ab. Zusätzlich zur Möglichkeit, per SQL-Anfrage Ausführungspläne aus der PLAN_TABLE auszulesen, unterstützt Oracle die **Planausgabeformate** TEXT, XML und HTML. Diese können jeweils über die im Package DBMS_XPLAN enthaltenen Tabellenfunktionen DISPLAY und deren Erweiterung DISPLAY_PLAN erzeugt werden. Ebenfalls möglich in Oracle ist die grafische Ausgabe von Ausführungsplänen, entweder über das Werkzeug SQL-Developer oder den mächtigeren Enterprise Manager. Hinsichtlich des **Planumfangs** sind Ausführungspläne in Oracle sehr beschränkt. Weder Index- noch MV-Prozesse noch RI-Prüfungen werden im Ausführungsplan als Operatoren ausgewiesen und sind damit nur in einem erhöhten Anfrageverarbeitungsaufwand ersichtlich. **Aufwände** allgemein weist Oracle in allen betrachteten Metriken CPU, I/O und als Gesamtaufwand aus. Während der CPU-Aufwand proportional zu den Maschineninstruktionen und der I/O-Aufwand zur Anzahl gelesener Datenblöcke ausgewiesen werden, existiert für die sich daraus berechneten Gesamtkosten keine Maßeinheit.

MySQL [7]: Ausführungspläne werden in MySQL mittels SQL-Anfrage EXPLAIN **erstellt** und direkt in Tabellenform **ausgegeben**. Eine explizite **Speicherung** der Pläne erfolgt nicht. Seit der Version 5.6 besteht auch die Möglichkeit, sich die Ausführungspläne in MySQL Workbench grafisch darstellen oder sie mit erweiterten Details im JSON-Format ausgeben zu lassen. Bei der Planerstellung wird in MySQL die Gültigkeit der Anfrage nicht überprüft, es können also Pläne für Anfragen erstellt werden, die unzulässig sind (beispielsweise kann beim Einfügen der Datentyp falsch sein). Auch bezüglich des **Planumfangs** zeigt sich MySQL eher rudimentär. Es gibt keine materialisierten Sichten und über Indexpflege oder das Überprüfen referentieller Integrität macht der Plan keinerlei Angaben. Bezüglich der **Aufwandsabschätzung** weist MySQL CPU-, I/O- und den Gesamtaufwand aus.

Microsoft SQL-Server [7]: Im SQL-Server werden Pläne erst bei ihrer Ausgabe externalisiert. Ein Mechanismus zur reinen **Erzeugung** und spezielle Strukturen zur **Speicherung** sind damit nicht nötig und folglich nicht vorhanden. Zur **Ausgabe** von Ausführungsplänen existieren verschiedene SHOWPLAN SET-Optionen.

Sobald diese aktiviert sind, werden für nachfolgend ausgeführte SQL-Anfragen automatisch Pläne in den Formaten TEXT oder XML erstellt. Eine grafische Ausgabe bietet das Management Studio. Bezogen auf den **Planumfang** sind Informationen zu sämtlichen betrachteten Prozessen enthalten. Dies gilt analog für die **Aufwandsabschätzung**, für die CPU-, I/O- und Gesamtaufwand ausgewiesen werden. Maßeinheiten sind allerdings nicht bekannt.

PostgreSQL [10]: Ausführungspläne werden in PostgreSQL mittels SQL-Anfrage EXPLAIN **erstellt** und ohne explizite **Speicherung** direkt **ausgegeben**. Soll der Plan dabei zusätzlich ausgeführt werden, muss dies mittels Schlüsselwort ANALYZE proklamiert werden. Über ein weiteres Schlüsselwort VERBOSE lassen sich besonders ausführliche Informationen abrufen. Auch das **Format**, in welchem der Plan ausgegeben werden soll, lässt sich mit entsprechendem Schlüsselwort (FORMAT {TEXT | XML | JSON | YAML}) angeben. Eine grafische Ausgabe bietet das Werkzeug pgAdmin. Zu beachten ist zudem, dass PostgreSQL die Gültigkeit der Anfrage beim Erstellen des Plans (ohne Schlüsselwort ANALYZE) nicht überprüft. Bezüglich des **Planumfangs** macht PostgreSQL keine Angaben über Indexpflege oder die Prüfung referentieller Integrität. Es existieren zwar materialisierte Sichten, diese können aber lediglich manuell über den Befehl REFRESH aktualisiert werden. Für die **Aufwandsabschätzung** liefert PostgreSQL nur „costs“, deren Einheit die Zeit darstellt, die für das Lesen eines 8KB-Blocks benötigt wird.

DB2 LUW [13]: Ausführungspläne werden in DB2 LUW über die SQL-Anfrage EXPLAIN **erstellt** und in einem Set von 20 Explain-Tabellen **gespeichert**. Für die **Ausgabe** werden darin abgelegten Informationen besteht neben dem Auslesen per SQL-Anfrage nur die Möglichkeit der textuellen Darstellung mithilfe der mitgelieferten Werkzeuge db2exfmt und db2expln. Über das Data Studio und den Infosphere Optim Query Workload Tuner können Ausführungspläne grafisch ausgegeben werden. In ihrem **Umfang** berücksichtigen Pläne von DB2 LUW zwar die MQT-Pflege und RI-Prüfungen, die Pflege von Indexen jedoch wird nicht dargestellt. Hinsichtlich der **Aufwandsabschätzung** deckt DB2 LUW als einziges der betrachteten Systeme nicht nur alle Metriken ab, sondern liefert zu diesen auch konkrete Bewertungsmaße. CPU-Aufwand wird in CPU-Instruktionen, I/O-Aufwand in Daten-seiten-I/Os und der Gesamtaufwand in Timerons angegeben. Da letztere eine IBM-interne Maßeinheit darstellen, entkräftet sich allerdings diese positive Sonderrolle wieder.

DB2 z/OS [16]: In der **Erstellung** und **Speicherung** von Ausführungsplänen ähnelt DB2 z/OS dem zuvor betrachteten DB2 LUW. Es wird ebenfalls die EXPLAIN-Anfrage und eine tabellarische Speicherung verwendet. Die Struktur der verwendeten Tabellen ist jedoch grundverschieden. Neben der Variante, DB2-Zugriffspläne direkt aus der tabellarischen Struktur auszulesen, ist lediglich die grafische **Planausgabe** über separate Werkzeuge wie Data Studio oder den Infosphere Optim Query Workload Tuner möglich. Der **Umfang** des Ausführungsplans in DB2 z/OS ist sehr beschränkt. Es werden keine der betrachteten Prozesse dargestellt. Die **Aufwandsschätzung** dagegen umfasst Angaben zu CPU-, I/O- und Gesamtaufwand. Erstere wird sowohl in CPU-Millisekunden als auch in sogenannten Service Units ausgewiesen.

4. PLANOPERATOREN IN RDBMS

Ähnlich zu den Ausführungsplänen setzt sich die Verschiedenheit der betrachteten DBMS auch in der Beschreibung ihrer in weiten Teilen gleichartig arbeitenden Planoperatoren fort. Diese werden in den folgenden Ausführungen gegenübergestellt. Abschnitt 4.1. vergleicht sie anhand der bereits beschriebenen Kriterien. Im anschließenden Abschnitt 4.2. wird eine Kategorisierung der einzelnen Operatoren nach ihrer Funktionalität beschrieben.

4.1 Operatordetails

Auf Basis der in Abschnitt 2.2 beschriebenen Vergleichskriterien ergibt sich für die Operatordetails der miteinander verglichenen DBMS die in *Abbildung 3* gezeigte Matrix. Auf eine Darstellung der DBMS-spezifischen Bezeichnungen ist darin aus Kompaktheitsgründen verzichtet worden. Die in der Matrix ersichtlichen Auffälligkeiten sollen nun näher ausgeführt werden.

DBMS	Operatordetails						
	Rows	Bytes	Aufwand absolut	Aufwand kumulativ	Aufwand StartUp	Projektion	Aliase
Oracle (12c R1)	x	x	-	x	-	x	x
MySQL (5.7)	x	-	x	-	x	x	x
Microsoft SQL-Server (2014)	x	x	x	x	-	x	x
PostgreSQL (9.3)	x	x	-	x	x	x	x
DB2 LUW (V10.5)	x	-	-	x	x	x	x
DB2 z/OS (V11)	x	-	-	x	-	-	x

Abbildung 3: Vergleich der Operatordetails

Eine der wesentlichen Aussagen der Matrix ist, dass ein überwiegender Teil an zentralen Details existiert, die von nahezu allen DBMS für Ausführungsplanoperatoren ausgewiesen werden. Dies sind die Ergebniszeilenanzahl (Rows), der kumulative Aufwand, Projektionslisten und Aliase. Andere Details wie die Größe des Zwischenergebnisses (Bytes), der absolute oder der Startup-Aufwand hingegen stehen nur bei einzelnen DBMS zur Verfügung. Bei horizontaler Betrachtung fällt auf, dass kein DBMS alle Detailinformationen bereithält. Stattdessen schwankt der Detailumfang von lediglich nicht ausgewiesenen Startup-Aufwand beim SQL-Server oder dem einzig fehlenden absoluten Aufwand bei PostgreSQL bis hin zum DB2 z/OS, das nur die zuvor als zentral betitelten Details mitführt.

4.2 Operatoren-Raster

Planoperatoren sind in den betrachteten DBMS verschieden granular ausgeprägt. Damit gemeint ist die Anzahl von Operatoren, die von knapp 60 im SQL-Server bis hin zu etwa halb so vielen Operatoren in PostgreSQL reicht. Trotzdem decken die einzelnen

DBMS, wie in *Abbildung 4* auf Basis einer Matrix dargestellt, mit wenigen Ausnahmen die betrachteten Grundoperatoren funktional mit mindestens einem spezifischen Operator ab. Die weiteren Ausführungen nehmen detaillierter Bezug auf die Matrix. Dabei werden aus Umfangsgründen nur ausgewählte Besonderheiten behandelt, die einer zusätzlichen Erklärung bedürfen.

Oracle [5, 6]: Ausführungsplanoperatoren in Oracle charakterisieren sich durch eine Operation und zugehörige Optionen. Während erstere eine allgemeine Beschreibung zum Operator liefert, beschreiben zweite den konkreten Zweck eines Operators näher. Beispielsweise kann für eine SORT-Operation anhand ihrer Option unterschieden werden, ob es sich um eine reine Sortierung (ORDER BY) oder einen Aggregationsoperator (AGGREGATE) handelt. Allgemein fällt zu den Planoperatoren auf, dass in Oracle viele OLAP-Operatoren wie etwa CUBESCAN oder PIVOT ausgewiesen werden, die sich keinem der Grundoperatoren zuordnen lassen und damit jeweils als otherIntermediate kategorisiert wurden. Besonders ist auch die Tatsache, dass zwar ein REMOTE-Zugriffsoperator, jedoch kein entsprechender Manipulationsoperator existiert. Ein solcher wird jedoch nicht benötigt, da Oracle ändernde Ausführungspläne stets von der DBMS-Instanz ausgehend erstellt, auf der die Manipulation stattfindet, und gleichzeitige Manipulationen auf mehreren Servern nicht möglich sind.

MySQL [7]: MySQL unterstützt lediglich "Left-deep-linear" Pläne, in denen jedes zugegriffene Objekt direkt mit dem gesamten bisherigen Vorergebnis verknüpft wird. Dies wirkt sich auch auf die tabellarische Planstruktur aus, die mit wenigen Ausnahmen wie etwa UNION RESULT je involvierter Tabelle genau eine Zeile enthält. Operatoren werden darin nicht immer explizit verzeichnet, was ihre Kategorisierung sehr erschwert. Die Art des Datenzugriffs ergibt sich im Wesentlichen aus der „type“-Spalte, welche in der MySQL-Dokumentation mit „join type“ näher beschrieben wird. Als eigentlicher Verbund-Operator existiert nur der „nested loop“-Join. Welche Art der Zwischenverarbeitung zum Einsatz kommt, muss den Spalten „Extra“ und „select_type“ entnommen werden. Manipulationsoperatoren werden in Plänen erst seit Version 5.6.3 und dort unter „select_type“ ausgewiesen.

Microsoft SQL-Server [8, 9]: Im SQL-Server kennzeichnen sich Ausführungsplanoperatoren durch einen logischen und einen physischen Operator. Für die abgebildete Matrix sind letztere relevant, da, wie ihr Name bereits suggeriert, sie diejenigen sind, die Aufschluss über die tatsächliche physische Abarbeitung von Anfragen geben. Charakteristisch für den SQL-Server sind die verschiedenen Spool-Operatoren Table, Index, Row Count und Window Spool. All diesen ist gemein, dass sie Zwischenergebnisse in der sogenannten tempDB materialisieren („aufspulen“) und diese im Anschluss wiederum nach bestimmten Werten durchsuchen. Die Spool-Operatoren sind daher sowohl den Manipulations- als auch den Zugriffsoperatoren zugeordnet worden. Einzig der Row Count Spool wurde als otherIntermediate-Operator klassifiziert, da sein Spulen ausschließlich für Existenzprüfungen genutzt wird und er dabei die Anzahl von Zwischenergebnisseilen zählt. SQL-Server besitzt als einziges der betrachteten DBMS dedizierte remoteManipulation-Operatoren, die zur Abarbeitung einer Manipulation bezogen auf ein Objekt in einer fernen Quelle verwendet werden. Zuletzt sollen als Besonderheit die Operatoren Collapse und Split erwähnt werden. Diese treten nur im Zusammenhang mit UPDATE-Operatoren auf und wurden daher ebenfalls in deren Gruppe eingeordnet. Funktional dienen sie dazu, das sogenannte Halloween Problem zu lösen, welches beim Update von aus einem Index gelesenen Daten aufkommen kann [10].

DBMS	Zugriffoperatoren				Zwischenverarbeitungsoperatoren				Manipulationsoperatoren				Rückgabeoperatoren						
	tableAccess	indexAccess	generatedRowAccess	remoteAccess	otherAccess	join	set	sort	aggregate	filter	bitmap	otherIntermediate		insert	update	delete	merge	remoteManipulation	otherManipulation
Oracle (12c R1)	MAT_VIEW ACCESS, MAT_VIEW REWRITE ACCESS, TABLE ACCESS	BITMAP, DOMAIN INDEX, INDEX	-	RE-MOTE	CUBE SCAN, SEQUENCE (Key Access), index_merge, Using index condition, Using join buffer (Block Nested loop)	CUBE JOIN, HASH JOIN, MERGE JOIN, NESTED LOOPS	CONCATENATION, INTERSECTION, MINUS, UNION	SORT	SORT HASH, SORT NATION	COUNT, FILTER, FIRST ROW	BITMAP	AND-EQUAL, CONNECT BY, FOR UPDATE, INLIST ITERATOR, PARTITION, PX (COORDINATOR PARTITION RECEIVE SEND), PIVOT, UNPIVOT, VIEW	INSERT STATEMENT	UPDATE STATEMENT	DELETE STATEMENT	-	-	-	(SELECT INSERT UPDATE DELETE) STATEMENT
	ALL const, index, FULL scan on NULL key, system, Using temporary	fulltext, index_subquery, range, index_ref, ref_or_null, no system, unique_subquery	DERIVED - no tables used	-	Using index condition, Using join buffer (Block Nested loop [Batched Key Access]), Using MRR	nested loop	DEPENDENT UNCACHABLE UNION	Using file-sorts	Using index for group-by, Using filesort	Range checked for each record, Using where [with pushed condition]	-	-	INSERT REPLACE, Using temporary Batch Hash, Clustered Index Insert, Index Spool, Online Index	Clustered Index Update, Collapse, Index Update, Online Index Update, Table Build, Table Split, Insert, Table Spool, Window Spool	Clustered Index Delete, Index Delete, Index Delete, Index Merge, Table Merge	Remote Delete, Remote Insert, Remote Update	-	-	SELECT, INSERT, UPDATE, DELETE, MERGE
MySQL (5.7)	Deleted Scan, Inserted Scan, Parameter Table Scan, RID Lookup, Table Scan, Table Spool, Table-valued Function, Window Spool	Clustered Index Scan, Clustered Index Seek, Index Scan, Index Seek, Index Spool	Remote Index Scan, Remote Index Seek, Remote Query, Remote Scan	-	Using index condition, Using join buffer (Block Nested loop)	Hash match, Merge Join, Nested Loops	Concatenation, Hash Match	Sort	Hash Match, Stream Aggregate, HashAggregate, WindowAgg, GroupAggregate, Sort, Unique	Filter, Top	Bitmap	Assert, Compute Scalar, Merge Interval, Parallelism, Row Count Spool, Switch, Segment, Sequence Project, UDX	Clustered Index Insert, Index Spool, Online Index Build, Table Split, Insert, Table Spool, Window Spool	Clustered Index Update, Collapse, Index Update, Online Index Update, Table Build, Table Split, Insert, Table Spool, Window Spool	Clustered Index Delete, Index Delete, Index Delete, Index Merge, Table Merge	Remote Delete, Remote Insert, Remote Update	-	-	SELECT, INSERT, UPDATE, DELETE, MERGE
Microsoft SQL-Server (2014)	Table-valued Function, Window Spool	Index Scan, Index Seek, Index Spool	Con-start Scan	Log Row Scan	Nested Loops	Concatenation, Hash Match	Sort	Hash Match, Stream Aggregate, HashAggregate, WindowAgg, GroupAggregate, Sort, Unique	Filter, Top	Bitmap	Project, UDX	Batch Hash, Clustered Index Insert, Index Spool, Online Index Build, Table Split, Insert, Table Spool, Window Spool	Clustered Index Update, Collapse, Index Update, Online Index Update, Table Build, Table Split, Insert, Table Spool, Window Spool	Clustered Index Delete, Index Delete, Index Delete, Index Merge, Table Merge	Remote Delete, Remote Insert, Remote Update	-	-	SELECT, INSERT, UPDATE, DELETE, MERGE	
PostgreSQL (9.3)	Materialize, Seq Scan	Bitmap (Index Heap) Scan, Index Scan, Only Scan, Materialize	Function Scan on series	Function Scan on dblink	Values Scan	Nested Loop, Hash Join, Merge Join	HashAggregate, HashSetOp, Append	Sort	Filter, Limit	-	-	Insert, Materialize	Update	Delete	-	-	Function Scan on dblink	-	
DB2 LUW (V10.5)	FETCH, TBSCAN	EISCAN, IXSCAN, XISCAN	GEN-ROW	RPD, SHIP	RIDSCAN, XSCAN	HSJOIN, MSJOIN, NLIJOIN, ZZJOIN	UNION	SORT	GRPBY	FILTER	XANDOR	CMPEXP, MGSTREAM, PIPE, REBAL, TQ	INSERT, TEMP	UPDATE	DELETE	-	-	RETURN	
DB2 z/OS (V11)	CORSUB ACCESS, FETCH, DFETCH, HSSCAN, TBSCAN, WFSKAN	DXSCAN, FEETCH, FIXSCAN, IXSCAN, RGLIST, SIXSCAN, XISSCAN	-	-	MIXSCAN	NLIJOIN, STARJOIN, MSJOIN, SEMJOIN, HBIJOIN	EXCEPT, INTERSECT, INTERSECTA, UNION	SORT	GRPBY	FILTER	XANDOR	INLIST, PARTITION, REPARTITION, RID FETCH	INSERT, WFILE	UPDATE	DELETE, TRUNCATE	MERGE	-	-	QUERY, QB

Abbildung 4: Kategorisierung der Ausführungsplanoperatoren

PostgreSQL [11]: Ausführungspläne in PostgreSQL sind gut strukturiert und bieten zu jedem Operator die für ihn entscheidenden Informationen, sei es der verwendete Schlüssel beim Sortieren oder welcher Index auf welcher Tabelle bei einem Indexscan zum Tragen kam. Für die Sortierung wird zudem angegeben, welche Sortiermethode benutzt wurde (quicksort bei genügend Hauptspeicherplatz, ansonsten external sort). Bei Unterabfragen wird der Plan zusätzlich unterteilt (sub plan, init plan). Soll auf Daten einer anderen (PostgreSQL-) Datenbank zugegriffen werden, so kommt das zusätzliche Modul dblink zum Tragen. Dieses leitet die entsprechende Anfrage einfach an die Zieldatenbank weiter, sodass aus dem Anfrageplan nicht ersichtlich wird, ob es sich um lesenden oder schreibenden Zugriff handelt. Auf einen expliziten Ausgabeoperator wurde in PostgreSQL verzichtet.

DB2 LUW [14]: Analog den zuvor betrachteten DBMS zeigen sich auch für die Planoperatoren von DB2 LUW verschiedene Besonderheiten. Mit XISCAN, XSCAN und XANDOR existieren spezielle Operatoren zur Verarbeitung von XML-Dokumenten. Für den Zugriff auf Objekte in fernen Quellen verfügt DB2 LUW über die Operatoren RFD (nicht relationale Quelle) und SHIP (relationale Quelle). Operatoren zur Manipulationen von Daten in fernen Quellen existieren nicht. Diese Prozesse werden im Ausführungsplan gänzlich vernachlässigt. Die Operatoren UNIQUE (einfach) und MGSTREAM (mehrfach) dienen zur Duplikateleminierung. Da sie dabei aber weder Sortier- noch Aggregationsfunktionalität leisten, wurden sie als otherIntermediate eingeordnet. Dort finden sich auch die Operatoren CMPEXP und PIPE, die lediglich für Debugging-Zecke von Bedeutung sind.

DB2 z/OS [15]: Obwohl DB2 z/OS und DB2 LUW gemeinsam zur DB2-Familie gehören, unterscheiden sich deren Ausführungsplanoperatoren nicht unerheblich voneinander. Lediglich etwa ein Drittel der DB2 LUW Operatoren findet sich namentlich und funktional annähernd identisch auch in DB2 z/OS wieder. Besonders für DB2 z/OS Planoperatoren sind vor allem die insgesamt 5 verschiedenen Join-Operatoren, die vielfältigen Indexzugriffsoperatoren und das Fehlen von Remote-, Aggregations- und Filteroperatoren. Remote-Operatoren sind nicht notwendig, da DB2 z/OS ferne Anfragen nur dann unterstützt, wenn diese ausschließlich Objekte einer DBMS-Instanz (Subsystem) referenzieren. Föderierte Anfragen sind nicht möglich. Aggregationen werden in DB2 z/OS entweder unmittelbar beim Zugriff oder über Materialisierung der Zwischenergebnisse in Form von temporären Workfiles (WKFILE) und anschließenden aggregierenden Workfile-Scans (WFSCAN) realisiert. Ein dedizierter Aggregationsoperator ist damit ebenfalls nicht nötig. Filteroperatoren existieren nicht, weil sämtliche Filterungen direkt in die vorausgehenden Zugriffsoperatoren eingebettet sind.

5. ZUSAMMENFASSUNG UND AUSBLICK

Der Beitrag verglich die Ausführungspläne und –planoperatoren der populärsten relationalen DBMS. Es konnte gezeigt werden, dass der grundlegende Aufbau von Ausführungsplänen sowie den dazu verwendeten Operatoren zu weiten Teilen systemübergreifend sehr ähnlich sind. Auf abstrakterer Ebene ist es sogar möglich, Grundoperatoren zu definieren, die von jedem DBMS in Abhängigkeit seines Funktionsumfangs gleichermaßen unterstützt werden. Der vorliegende Beitrag schlägt diesbezüglich eine Menge von Grundoperatoren und eine passende Kategorisierung der existierenden DBMS-spezifischen Operatoren vor. Darauf basierend ließe sich zukünftig ein Standardformat für Pläne definieren, mit dem die Entwicklung von Werkzeugen zur abstrakten DBMS-

unabhängigen Ausführungsplananalyse möglich wäre. Zusätzlich könnte es auch genutzt werden, um föderierte Pläne zwischen unterschiedlichen relationalen DBMS zu berechnen, die über bislang vorhandene abstrakte Remote-Operatoren hinausgehen.

6. LITERATUR

- [1] International Business Machines Corporation. *Solution Brief – IBM InfoSphere Optim Query Workload Tuner*, 2014. <ftp://ftp.boulder.ibm.com/common/ssi/ecm/en/ims14099usen/IMS14099USEN.PDF>
- [2] Dell Software Inc. *Toad World*, 2015. <https://www.toadworld.com>
- [3] AquaFold. *Aqua Data Studio – SQL Queries & Analysis Tool*, 2015. http://www.aquafold.com/aquadatastudio/query_analysis_tool.html
- [4] DB-Engines. *DB-Engines Ranking von Relational DBMS*, 2015. <http://db-engines.com/de/ranking/relational+dbms>
- [5] Oracle Corporation. *Oracle Database 12c Release 1 (12.1) – Database SQL Tuning Guide*, 2014. <https://docs.oracle.com/database/121/TGSQL.pdf>
- [6] Oracle Corporation. *Oracle Database 12c Release 1 (12.1) – PL/SQL Packages and Types Reference*, 2013. <https://docs.oracle.com/database/121/ARPLS.pdf>
- [7] Oracle Corporation. *MySQL 5.7 Reference Manual*, 2015. <http://downloads.mysql.com/docs/refman-5.7-en.a4.pdf>
- [8] Microsoft Corporation. *SQL Server 2014 – Transact-SQL Reference (Database Engine)*, 2015. <https://msdn.microsoft.com/de-de/library/bb510741.aspx>
- [9] Microsoft Corporation. *SQL Server 2014 – Showplan Logical and Physical Operators Reference*, 2015. <https://technet.microsoft.com/de-de/library/ms191158.aspx>
- [10] PostgreSQL Global Development Group. *PostgreSQL 9.3.6 Documentation – EXPLAIN*, 2015. <http://www.postgresql.org/docs/9.3/static/sql-explain.html>
- [11] EnterpriseDB. *Explaining EXPLAIN*, 2008. https://wiki.postgresql.org/images/4/45/Explaining_EXPLAIN.pdf
- [12] F. Amorim. *Complete Showplan Operators*, Simple Talk Publishing, 2014. [https://www.simple-talk.com/simplepod/Complete_Showplan_Operators_Fabiano_Amorim_\(without_video\).pdf](https://www.simple-talk.com/simplepod/Complete_Showplan_Operators_Fabiano_Amorim_(without_video).pdf)
- [13] International Business Machines Corporation. *DB2 10.5 for Linux, UNIX and Windows – Troubleshooting and Tuning Database Performance*, 2015. http://public.dhe.ibm.com/ps/products/db2/info/vr105/pdf/en_US/DB2PerfTuneTroubleshoot-db2d3e1051.pdf
- [14] International Business Machines Corporation. *Knowledge Center – DB2 10.5 for Linux, UNIX and Windows – Explain operators*, 2014. http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.explain.doc/doc/r0052023.html
- [15] International Business Machines Corporation. *Knowledge Center – Nodes for DB2 for z/OS*, 2014. https://www-304.ibm.com/support/knowledgecenter/SS7LB8_4.1.0/com.ibm.datatools.visualexplain.data.doc/topics/znodes.html
- [16] International Business Machines Corporation. *DB2 11 for z/OS – SQL Reference*, 2014. <http://publib.boulder.ibm.com/epubs/pdf/dsnsgn05.pdf>

Modularisierung leichtgewichtiger Kompressionsalgorithmen

Juliana Hildebrandt, Dirk Habich, Patrick Damme, Wolfgang Lehner
Technische Universität Dresden
Database Systems Group
01189 Dresden, Germany
firstname.lastname@tu-dresden.de

ABSTRACT

Im Kontext von In-Memory Datenbanksystemen nehmen leichtgewichtige Kompressionsalgorithmen eine entscheidende Rolle ein, um eine effiziente Speicherung und Verarbeitung großer Datenmengen im Hauptspeicher zu realisieren. Verglichen mit klassischen Komprimierungstechniken wie z.B. Huffman erzielen leichtgewichtige Kompressionsalgorithmen vergleichbare Kompressionsraten aufgrund der Einbeziehung von Kontextwissen und erlauben eine schnellere Kompression und Dekompression. Die Vielfalt der leichtgewichtigen Kompressionsalgorithmen hat in den letzten Jahren zugenommen, da ein großes Optimierungspotential über die Einbeziehung des Kontextwissens besteht. Um diese Vielfalt zu bewältigen, haben wir uns mit der Modularisierung von leichtgewichtigen Kompressionsalgorithmen beschäftigt und ein allgemeines Kompressionsschema entwickelt. Durch den Austausch einzelner Module oder auch nur eingehender Parameter lassen sich verschiedene Algorithmen einfach realisieren.

1. EINFÜHRUNG

Die Bedeutung von In-Memory Datenbanksystemen steigt zunehmend sowohl im wissenschaftlichen als auch im kommerziellen Kontext. In-Memory Datenbanksysteme verfolgen einen Hauptspeicherzentrischen Architekturansatz, der sich dadurch auszeichnet, dass alle performancekritischen Operationen und internen Datenstrukturen für den Zugriff der Hauptspeicherhierarchie (z.B. effiziente Nutzung der Cachehierarchie etc.) ausgelegt sind. Üblicherweise gehen In-Memory Datenbanksysteme davon aus, dass alle relevanten Datenbestände auch vollständig in den Hauptspeicher eines Rechners oder eines Rechnerverbundes abgelegt werden können. Die Optimierung der internen Datenrepräsentationen wird damit extrem wichtig, da jeder Zugriff auf ein Zwischenergebnis genau so teuer ist wie ein Zugriff auf die Basisdaten [13].

Für In-Memory Datenbanksysteme haben sich zwei orthogonale Optimierungstechniken für die effiziente Behandlung

der Zwischenergebnisse etabliert. Auf der einen Seite sollten Zwischenergebnisse nicht mehr zum Beispiel durch entsprechend angepasste Code-Generierung [19] oder durch den Einsatz zusammengefügter Operatoren [16] produziert werden. Auf der anderen Seite sollten Zwischenergebnisse (wenn sie beispielsweise nicht vermeidbar sind) so organisiert werden, dass eine effiziente Weiterverarbeitung ermöglicht wird. Im Rahmen unserer aktuellen Forschung greifen wir uns den optimierten Einsatz leichtgewichtiger Kompressionsverfahren für Zwischenergebnisse in Hauptspeicherzentrischen Datenbankarchitekturen heraus und haben zum Ziel, eine *ausgewogene Anfrageverarbeitung auf Basis komprimierter Zwischenergebnisse* zu entwickeln [13]. Mit der expliziten Kompression aller Zwischenergebnisse soll (i) die Effizienz einzelner Datenbankanfragen bzw. der Durchsatz einer Menge an Datenbankanfragen erhöht werden, da der Hauptspeicherbedarf für Zwischenergebnisse reduziert und der Mehraufwand zur Generierung der komprimierten Form möglichst gering gehalten wird und (ii) die durchgängige Betrachtung der Kompression von den Basisdaten bis hin zur Anfrageverarbeitung etabliert wird.

Im Forschungsbereich der klassischen Kompression existiert eine Vielzahl an wissenschaftlichen Publikationen. Klassische Kompressionsverfahren, wie zum Beispiel arithmetisches Kodieren [26], Huffman [15] und Lempel-Ziv [30], erzielen hohe Kompressionsraten, sind jedoch rechenintensiv und werden deshalb oft als schwergewichtige Kompressionsverfahren bezeichnet. Speziell für den Einsatz in In-Memory Datenbanksystemen wurden leichtgewichtige Kompressionsalgorithmen entwickelt, die verglichen mit klassischen Verfahren aufgrund der Einbeziehung von Kontextwissen ähnliche Kompressionsraten erzielen, aber sowohl eine viel schnellere Kompression als auch Dekompression erlauben. Beispiele für leichtgewichtige Kompressionsalgorithmen sind unter anderem Domain Kodierung (DC) [24], Wörterbuch-basierte Kompression (Dict) [5, 8, 17], reihenfolgeerhaltende Kodierungen [5, 28], Lauflängenkodierung (RLE) [7, 21], Frame-of-Reference (FOR) [12, 31] und verschiedene Arten von Nullkomprimierung [1, 20, 21, 23]. Die Anzahl der leichtgewichtigen Kompressionsalgorithmen hat in den letzten Jahren zugenommen, da ein großes Optimierungspotential über die Einbeziehung von Kontextwissen besteht.

Mit Blick auf unser Ziel der *ausgewogenen Anfrageverarbeitung auf Basis komprimierter Zwischenergebnisse* wollen wir eine breite Vielfalt an leichtgewichtigen Kompressionsalgorithmen unterstützen, um die jeweiligen Vorteile der Algorithmen effizient ausnutzen zu können. Um dieses Ziel zu erreichen, haben wir uns mit der Modularisierung von

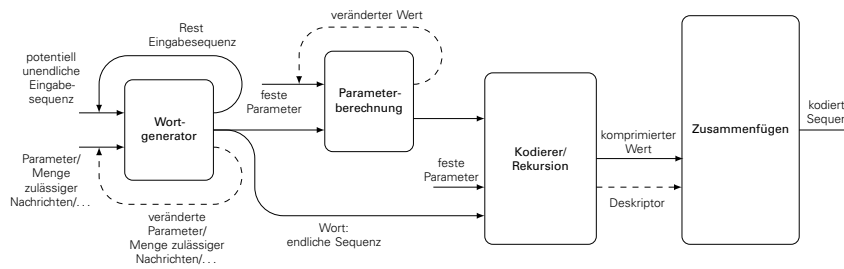


Abbildung 1: Allgemeines Schema für leichtgewichtige Komprimierungsalgorithmen.

Kompressionsalgorithmen beschäftigt. Für diese Modularisierung haben wir eine Vielzahl von leichtgewichtigen Kompressionsalgorithmen systematisch analysiert und ein allgemeines Kompressionsschema bestehend aus wohldefinierten Modulen abgeleitet. Durch den Austausch einzelner Module oder auch nur eingehender Parameter lassen sich häufig verschiedene Algorithmen einfach darstellen. Des Weiteren wird durch die Darstellung eines Algorithmus und durch die Unterteilung in verschiedene, möglichst unabhängige kleinere Module die Verständlichkeit erleichtert. Unsere entwickelte Strukturierung bildet eine gute Basis zur abstrakten und implementierungsunabhängigen Betrachtung von leichtgewichtigen Kompressionsalgorithmen.

Im Abschnitt 2 führen wir unser neues Kompressionsschema für leichtgewichtige Kompressionsverfahren bestehend aus vier Modulen ein. Dieses neue Kompressionsschema nutzen wir in Abschnitt 3, um bekannte Muster zu definieren. Im Anschluss daran gehen wir auf die Modularisierung konkreter Algorithmen exemplarisch im Abschnitt 4 ein. Der Artikel schließt dann mit einer Zusammenfassung und einem Ausblick im Abschnitt 5.

2. KOMPRESSIONSSCHEMA

Mit dem Paradigma der Datenkompression aus den 1980er Jahren [25] gibt es bereits eine eher allgemein gehaltene Modularisierung für die Kompression von Daten im Kontext der Datenübertragung. Diese unterteilt Kompressionsverfahren lediglich in ein *Datenmodell*, welches auf Grundlage bereits gelesener Daten erstellt und angepasst wird, und einen *Kodierer*, welcher eingehende Daten mithilfe des berechneten Datenmodells kodiert. Diese Modularisierung eignet sich für damals übliche adaptive Kompressionsmethoden; greift aber für viele Verfahren zu kurz. Die gesamte Zerlegung eines Eingabedatenstroms wird beispielsweise außen vor gelassen. Bei vielen aktuellen und gerade semiadaptiven Verfahren mit mehreren Pässen werden Daten mehrstufig zerlegt, um ein komplexes Datenmodell zu erzeugen. Auch das Zusammenfügen der Daten wird im Normalfall wesentlich diffiziler realisiert als mit einer einfachen Konkatenation komprimierter Daten. Unser aus vier Modulen bestehendes allgemeines Kompressionsschema in Abbildung 1 ist eine Erweiterung des bisherigen Paradigmas der Datenkompression, das eine wesentlich detailliertere und komplexere Abbildung einer Vielzahl von leichtgewichtigen Algorithmen erlaubt. Eingabe für ein Kompressionsverfahren ist hierbei immer eine potentiell unendliche Sequenz von Daten. Ausgabe ist ein Strom aus komprimierten Daten.

Der *Wortgenerator* als erstes Modul zerlegt die Sequenz in endliche Teilsequenzen resp. einzelne Werte. Mit einem Da-

tenstrom als Eingabe gibt ein Wortgenerator einen endlichen Anfang aus und verarbeitet den Rest der Eingabe ebenso, *rekursiv*. Ist die Eingabe des Wortgenerators endlich, kann ihre Zerlegung stattdessen auch *nicht rekursiv*, z.B. ein Optimierungsproblem sein. Wortgeneratoren erhalten als zweite Eingabe die Information, wie die Eingabesequenz zerlegt werden soll, als Berechnungsvorschrift. Entweder wird *datenunabhängig* eine Anzahl von Werten ausgegeben (z.B. immer 128 Werte oder immer ein Wert) oder *datenabhängig* aufgrund inhaltlicher Merkmale die Länge der auszugebenden Teilsequenz bestimmt. Möglich ist eine *adaptive* Zerlegung, so dass sich die Berechnungsvorschrift nach jeder Ausgabe einer Teilsequenz ändert. Als optionaler Datenfluss ist dies im Kompressionsschema durch eine unterbrochene Linie dargestellt.

Das Datenmodell des Paradigmas der Datenkompression wird durch das Modul der *Parameterberechnung* ersetzt. So können bei semiadaptiven Verfahren für endliche Sequenzen statistische Werte berechnet werden, wie zum Beispiel ein Referenzwert für Frame-of-Reference-Verfahren (FOR) oder eine gemeinsame Bitweite, mit der alle Werte der endlichen Sequenz kodiert werden können. Möglicherweise gibt es feste Eingabeparameter, beispielsweise eine Auswahl an erlaubten Bitweiten. Eine *adaptive* Parameterberechnung zeichnet sich durch einen Startwert als festen Eingabeparameter aus und eine Ausgabe, die im nächsten Schritt wieder als Eingabe und so dem Modul der Parameterberechnung als Gedächtnis dient. Beispielsweise benötigen Differenzkodierungsverfahren eine adaptive Parameterberechnung.

Der *Kodierer* erhält einen atomaren Eingabewert sowie möglicherweise berechnete oder feste Parameter, die für die Kodierung des Eingabewertes benötigt werden. Solche Parameter können z.B. Referenzwerte, Bitweiten oder gar Mappings sein, die die Abbildung einzelner Werte in einen Code definieren. Ein Kodierer bildet einen Eingabewert eindeutig auf einen anderen Wert ab, was für die Dekodierbarkeit notwendig ist. Ausgabe eines Kodierers ist ein komprimierter Wert, der sich möglicherweise durch einen Deskriptor wie einer Längenangabe bei einer Abbildung in einen variablen Code auszeichnet, um die Dekodierbarkeit zu gewährleisten. Soll eine endliche Sequenz, die der Wortgenerator ausgibt, noch weiter zerlegt werden, kann das gesamte Schema noch einmal mit einer *Rekursion* aufgerufen werden. Dabei geht eine endliche Sequenz wieder in einen Wortgenerator ein und wird dabei zerlegt, weiterverarbeitet und als komprimierte Sequenz wieder zusammengefügt. Diese komprimierte Sequenz ist Ausgabe der Rekursion.

Das letzte Modul des *Zusammenfügens* erhält als Eingabe einen komprimierten Datenstrom, nämlich die Ausga-

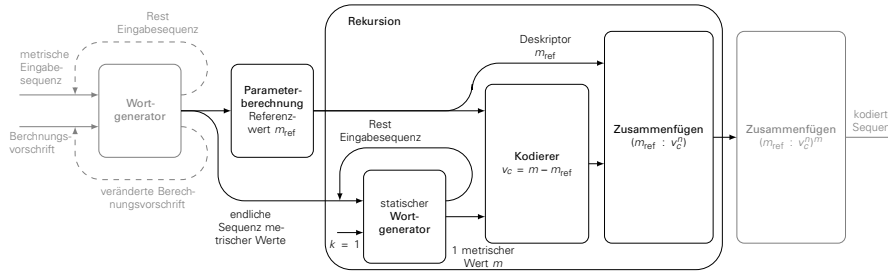


Abbildung 2: Modularisierung semiadaptiver Frame-of-Reference-Verfahren.

ben des Kodierers resp. der Rekursion. Es gibt verschiedene Möglichkeiten Daten zusammenzufügen. Im einfachsten Fall gibt es keine Deskriptoren, alle komprimierten Werte v_c werden nacheinander zusammengefügt (notiert als v_c^n). Gehört zu jedem komprimierten Wert ein Deskriptor d , so können beispielsweise immer Paare aus Deskriptoren und komprimierten Werten konkateniert werden (notiert als $(d : v_c)^n$) oder immer eine bestimmte Anzahl l von Deskriptoren, gefolgt von den zugehörigen komprimierten Werten (notiert als $(d^l : v_c^l)^n$). Gerade bei semiadaptiven Verfahren mit Rekursionen ist es möglich, dass gemeinsame Deskriptoren für mehrere Werte vom Modul der Parameterberechnung ausgehen und mit gespeichert werden müssen, so dass verschiedene Anordnungen bei der Konkatenation aller Werte denkbar sind.

3. KOMPRESSIIONSMUSTER

Bekannte Kompressionstechniken wie zum Beispiel Differenzkodierung, Frame-of-Reference (FOR), Wörterbuchkompression, Bitvektoren, Lauflängenkodierung (RLE) oder die Unterdrückung führender Nullen lassen sich mit dem allgemeinen Kompressionsschema als Muster ausdrücken. Das bedeutet, dass gewisse modulare Anordnungen und Inhalte einzelner Module durch die Begriffsdefinition der Techniken festgelegt, andere inhaltliche Aspekte sowie andere in Beziehung stehende Module hingegen nicht näher spezifiziert sind.

3.1 Muster Frame-of-Reference (FOR)

Für die allgemeine Definition des FOR muss die Eingabesequenz aus metrischen Werten bestehen, wie zum Beispiel aus natürlichen Zahlen. Diese werden als Differenz zum Referenzwert m_{ref} kodiert. Abbildung 3 zeigt das entsprechende Kompressionsschema. Der Wortgenerator gibt vom Anfang der Sequenz jeweils einen Integerwert m aus. Der Kodierer erhält neben den Eingabewerten den Referenzwert m_{ref} als

Parameter und berechnet die Differenz aus beiden Werten. Das Modul des Zusammenfügens konkateniert den Referenzwert mit allen kodierten Werten ($m_{ref} : v_c^n$). Notwendig ist die Speicherung des Referenzwertes aber nur, wenn dessen Kenntnis beim Dekodieren nicht vorausgesetzt werden kann. Dies ist durch einen optionalen Pfeil dargestellt. Im dargestellten Beispiel werden die Werte des Eingabedatenstroms als Differenz zum Referenzwert $m_{ref} = 273$ kodiert. Der Referenzwert sei beim Dekodieren aus dem Kontext bekannt.

Meist gehört es zum Selbstverständnis, dass der Referenzwert für eine endliche Sequenz wie in Abbildung 2 aus den gegebenen Daten berechnet und als Deskriptor gespeichert wird. Nach welchen Regeln der erste dargestellte Wortgenerator endliche Sequenzen ausgibt, ist dabei nicht spezifiziert. Das dargestellte Muster kann eine potentiell unendliche Sequenz als Eingabe erhalten. Es kann auch in einem größeren Zusammenhang mit anderen Modulen stehen und nur eine endliche Teilsequenz weiter zerlegen. Im Modul der Parameterberechnung wird aus der endlichen Sequenz, die der erste Wortgenerator ausgibt, der Referenzwert m_{ref} berechnet. Zum Beispiel kann als Referenzwert der kleinste Wert der endlichen Sequenz gewählt werden. Die endliche Sequenz geht in eine Rekursion ein. Arrangement und Inhalt der Module innerhalb der Rekursion entsprechen der Modularisierung statischer Frame-of-Reference-Verfahren (vgl. Abb. 3). Der Wortgenerator innerhalb der Rekursion gibt einzelne metrische Werte aus. Der Kodierer berechnet die Differenz aus Eingabe- und Referenzwert. Alle Werte werden gemeinsam mit dem Referenzwert konkateniert. Alle so komprimierten endlichen Sequenzen, die der erste dargestellte Wortgenerator ausgegeben hat, werden am Ende zusammengefügt, von Interesse sind für die allgemeinere Definition des FOR jedoch nur die Module innerhalb der Rekursion.

3.2 Muster Symbolunterdrückung

Unter dem Begriff Symbolunterdrückung werden sehr verschiedene Komprimierungsverfahren zusammengefasst, Präsenzbits sowie Lauflängenkodierung explizit für Nullen, die in potentiell unendliche Sequenzen auftauchen [6], Lauflängenkodierung von Nullen und Leerzeichen [21] oder auch die Eliminierung führender und damit redundanter Nullen bei binär kodierten Zahlen. Diese Methoden haben gemeinsam, dass es im Zeichenvorrat ein ausgezeichnetes Symbol s gibt, welches sich meist semantisch von allen anderen abhebt und öfter auftaucht als andere Werte. Das ausgezeichnete Symbol wird im Wortgenerator oder im Kodierer anders behandelt als andere Symbole. Im Falle von Präsenzbits ist dieses Symbol der NULL-Wert. Nullen sind das neutrale Element der Addition. Die genaue Anzahl führender Nullen beeinflusst

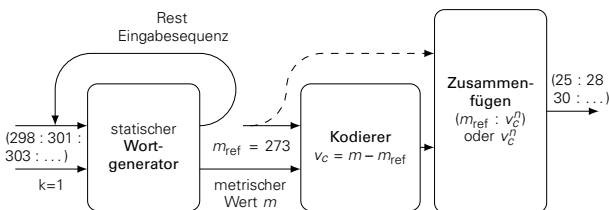


Abbildung 3: Modularisierung statischer Frame-of-Reference-Verfahren.

Additionsoperationen nicht und ist damit an sich schon eine redundante Information. Leerzeichen dienen in allen Sprachen dazu, Wörter voneinander zu separieren. Die Anzahl an Wiederholungen von Leerzeichen zwischen konkatenierten Wörtern besitzt auf semantischer Ebene keinerlei Bedeutung. Viele der Algorithmen, aber nicht alle, nutzen hierfür RLE-Kompressionen. Für Symbolunterdrückungen lässt sich allgemein keine Modularisierung darstellen, da es sich einfach nur durch die Sonderbehandlung eines Symbols auszeichnet. In Kombination mit einer Lauflängenkodierung gelingt aber eine Modularisierung mit unserem Schema.

Merkmal der Lauflängenkodierung ist das Vorhandensein von Läufen w^n , endlichen Sequenzen der Länge n aus ein und demselben Wert w . Werden wirklich einfach nur Läufe von Werten kodiert, so reicht das simple Kompressionsschema in Abbildung 4 aus. Der Wortgenerator unterteilt den

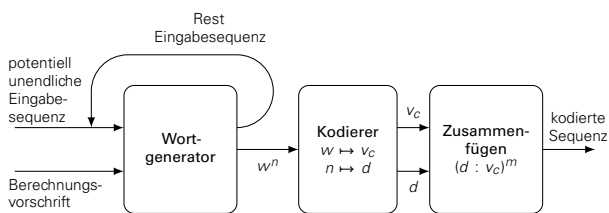


Abbildung 4: Modularisierung einer einfachen Lauflängenkodierung.

Eingabedatenstrom in Läufe. Im Kodierer werden dann der Wert w und die Lauflänge n kodiert. Läufe können auch in einer Sequenz zum Beispiel als führende Nullen eingebettet sein. Solche Fälle liegen im Schnittbereich zwischen Symbolunterdrückung und Lauflängenkodierung. Dies ist für statische Verfahren in Abbildung 5 dargestellt. Die Informationen über Sequenzlängen des ausgezeichneten Wertes s werden bei statischen Verfahren beim Zusammenfügen zum Beispiel in der Form $(d(n) : v_c)^m$ gespeichert. Dabei ist $d(n)$ eine eineindeutige Abbildung.

Sollen mehrere mit 32 Bits kodierte Werte mit geringerer, aber einheitlicher Bitweite gespeichert werden, wird für die Unterdrückung führender Nullen ein semiadaptives Schema benötigt (nicht dargestellt). Die gemeinsame Bitweite bw ist Ausgabe der Parameterberechnung und kann als Deskriptor angegeben werden, $bw = d(n)$ ist eine Funktion von n , der Anzahl der führenden Nullen, die entfernt wurden. Jeder

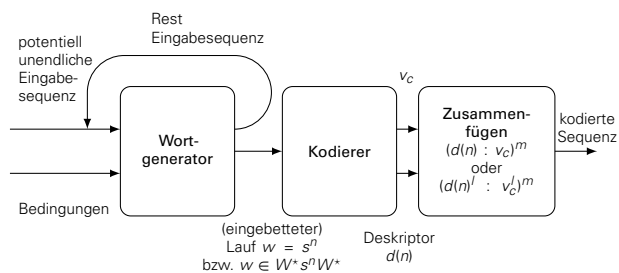


Abbildung 5: Modularisierung von symbolunterdrückenden Verfahren mit Lauflängenkodierung als statische Kompressionsverfahren.

Wert w einer endlichen Sequenz hat eine komprimierte Form v_c und eine Lauflänge n . Beide werden entweder zusammen oder in der Gruppe aus Deskriptoren und einer Gruppe aus komprimierten Werten gespeichert.

4. ALGORITHMEN-MODULARISIERUNG

Die vorgestellten und auch weitere Muster finden sich in verschiedenen Kompressionsalgorithmen, oft auch kombiniert oder auf mehreren Rekursionsebenen miteinander verwoben, wieder. Sich ähnelnde Algorithmen unterscheiden sich meist nur geringfügig in manchen Modulen oder sogar nur in Parametern, die in ein Modul eingehen. Beispielsweise ähneln sich die Algorithmen varint-PU und varint-SU [22] - letzterer ist besser bekannt als VByte [11, 10, 9] - sehr. VByte kodiert 32-Bit-Integerwerte mit ein bis 5 Bytes, wobei ein Byte aus einem Deskriptorbit und 7 Datenbits besteht. Ebenso ist dies bei varint-PU der Fall, beide Algorithmen unterscheiden sich nur in der Anordnung der Daten- und Deskriptorbits. Während bei VByte ein Bit pro zusammenhängendem Byte als Deskriptor dient, steht bei varint-PU der gesamte Deskriptor an einem Ende des komprimierten Integerwertes. Ein Beispiel zeigt Abbildung 6. Nicht belegte Bits bedeuten, dass diese im Beispiel nicht benötigt und weggelassen werden. Der Integerwert wird in komprimierter Form mit 3 statt 4 Bytes kodiert.

Beide Formate haben den gleichen modularen Aufbau (siehe Abbildung 7). Der rekursive statische Wortgenerator gibt immer eine Zahl aus. Da die Kodierung der Eingabe bei diesen Algorithmen soweit spezifiziert ist, dass die Zahlen als 32-Bit-Integerwerte kodiert sind, ist die ausgegebene Zahl ein eingebetteter Lauf von der Form $0^l 1 w_1 \dots w_{31-l}$ (bzw. 0^{32} für den Wert 0). Der Deskriptor $bw/7$ gibt die Anzahl der für die Datenbits benötigten 7-Bit-Einheiten an. Allein aus dem komprimierten Wert ohne Deskriptor ist die Lauflänge der bei der Kodierung unterschlagenen Nullen nicht ermittelbar, schon weil eine Folge von Werten nicht mehr dekodierbar ist. Die Lauflänge ist allein aus dem Deskriptor (und dem Wissen, dass es sich um 32-Bit-Integerwerte handelt) ersichtlich. Somit ist das Lauflängenmuster bei varint-SU und varint-PU begründbar. Da das Zeichen 0 eine Sonderstellung einnimmt, weil führende Nullen als Lauf betrachtet werden, findet sich hier, wie bei allen varint-Algorithmen, auch eine Symbolunterdrückung. Beide Algorithmen unterscheiden sich im Kompressionsschema nur im Modul des Zusammenfügens. Das Symbol \cdot wird hier als Konkatenationssymbol für abzählbar viele Werte verwendet.

Ein weiteres Beispiel für einen modularisierten Algorithmus ist FOR mit Binary Packing (nicht dargestellt), der sich durch marginale Veränderungen und weitere Definitionen aus dem Kompressionsschema für semiadaptive FOR-Verfahren (Abb. 2) ergibt. Beim Binary Packing wird für eine endliche Sequenz von n binär kodierten Integerwerten z.B. zu 32 Bits eine gemeinsame Bitweite bw berechnet, mit der alle n Werte kodiert werden können. Die erste Änderung im Kompressionsschema betrifft die Parameterberechnung. Zusätzlich zum Referenzwert für eine endliche Sequenz, die der erste Wortgenerator ausgibt, muss die gemeinsame Bitweite bw berechnet und ausgegeben werden. Die zweite Änderung betrifft den Kodierer innerhalb der Rekursion. Nach der Berechnung der Differenz aus Eingabe- und Referenzwert wird der so erhaltene Werte mit Bitweite bw binär kodiert. Im Modul des Zusammenfügens muss dann bw als ein weiterer gemeinsamer Deskriptor zum Beispiel in der Form

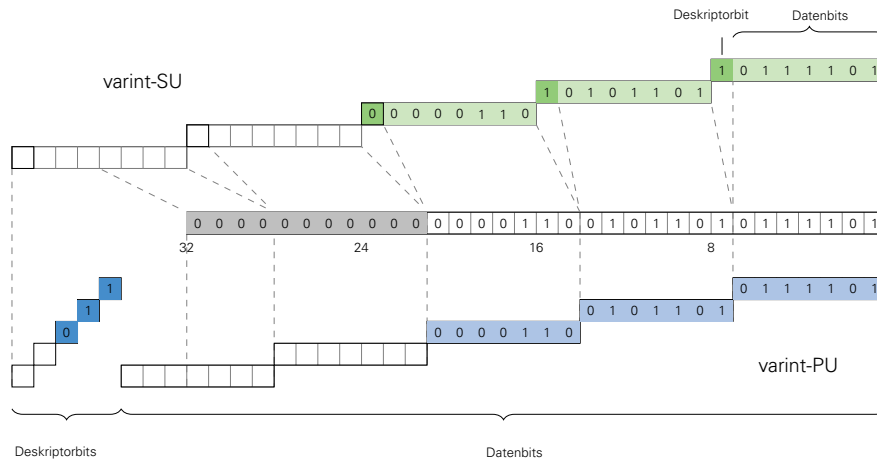


Abbildung 6: Datenformat varint-SU und varint-PU.

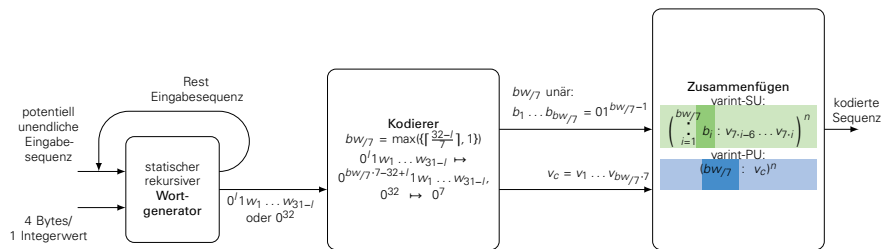


Abbildung 7: Modularisierung der Algorithmen varint-SU und varint-PU.

($m_{\text{ref}} : bw : v_c^n$) gespeichert werden. Die Modularisierung dieses Algorithmus zeichnet sich durch die Muster FOR, Lauflängenkodierung und Symbolunterdrückung aus.

Nicht für alle Algorithmen ist diese recht einfache Modularisierung ausreichend. Auf PFOR basierende Algorithmen [31, 29, 27, 18] kodieren die meisten Eingabewerte aus natürlichen Zahlen mit der gleichen Bitweite bw . Die größeren, die nicht mit der Bitweite bw kodierbar sind, werden allerdings als Ausnahme deklariert und auf andere Weise kodiert und an anderer Stelle gespeichert. Dafür benötigt das erweiterte Schema ein Splitmodul, welches Daten aufgrund inhaltlicher Merkmale in verschiedene Gruppen aufteilt und ausgibt. Für jede dieser Gruppen muss ein separater Kodierer verfügbar sein, wobei die kodierten Werte aller Gruppen am Ende gemeinsam zusammengefügt werden.

5. ZUSAMMENFASSUNG UND AUSBLICK

Unser entwickeltes Kompressionsschema bestehend aus vier Modulen ist durchaus geeignet, um eine Vielzahl verschiedener leichtgewichtiger Kompressionsalgorithmen gut zu modularisieren und systematisch darzustellen. Durch den Austausch einzelner Module oder auch nur eingehender Parameter lassen sich verschiedene Algorithmen mit dem gleichen Kompressionsschema darstellen. Einige Module und Modulgruppen tauchen in verschiedenen Algorithmen immer wieder auf, wie zum Beispiel die gesamte Rekursion, die das Binary Packing ausmacht, die sich in allen PFOR- und Simple-Algorithmen [2, 3, 29, 27, 4] findet. Die Verständlichkeit eines Algorithmus wird durch die Unterteilung in verschie-

dene, möglichst unabhängige kleinere Module, welche überschaubare Operationen ausführen, verbessert. Die Strukturierung durch das entwickelte Schema bildet aus unserer Sicht eine gute Basis zur abstrakten Betrachtung von leichtgewichtigen Kompressionsalgorithmen. Als Muster können nicht nur bestimmte Techniken, sondern auch andere Eigenschaften von Kompressionsalgorithmen dargestellt werden. Statische Verfahren wie z.B. varint-SU und varint-PU bestehen nur aus Wortgenerator, Kodierer und dem Modul des Zusammenfügens. Adaptive Verfahren haben einen adaptiven Wortgenerator, eine adaptive Parameterberechnung oder beides. Semiadaptive Verfahren zeichnen sich durch eine Parameterberechnung und eine Rekursion aus, in deren Wortgenerator oder Kodierer die Ausgabe der Parameterberechnung eingeht.

Durch die Möglichkeit Module sehr passend zusammenzustellen und mit Inhalt zu füllen, ergibt sich ein mächtiges Werkzeug für den automatisierten Bau von Algorithmen. Das Kompressionsschema bietet eine aus unserer Sicht fundierte Grundlage und eröffnet die Möglichkeit, für einen gegebenen Kontext sehr gezielt speziell zugeschnittene Algorithmen mit bestimmten Eigenschaften wie zum Beispiel der Art der Anpassbarkeit zusammenzubauen. Weiterhin können verschiedene Muster wie FOR, Differenzkodierung, Symbolunterdrückung oder Lauflängenkodierung an den Kontext angepasst eingesetzt werden und das auf verschiedensten Ebenen miteinander kombiniert.

Für die Fortführung dieses Gedankens ist es notwendig, einen noch stärkeren Zusammenhang zwischen Kontextwissen und passender Schemazusammenstellung sowie passen-

den Parametereingaben herzustellen. Des Weiteren wird gerade für das theoretische Grundkonzept eine passende praktische Umsetzung angegangen. Für die praktische Umsetzung wird ein Framework bestehend aus den eingeführten Modulen anvisiert, so dass der Zusammenbau leichtgewichtiger Kompressionsalgorithmen wie beschrieben realisiert werden kann. Die größte Herausforderung bei der praktischen Umsetzung wird die Effizienz der Algorithmen sein. Um eine vergleichbare Effizienz zu den bisherigen Implementierungen erzielen zu können, sind unterschiedliche Ansätze notwendig. Ein vielversprechender Ansatz dabei ist die Spezialisierung von generischen Code mit dem Einsatz spezieller Compiler-Techniken, wie wir es in [14] angesprochen haben. Über die Spezialisierung kann hochoptimierter Ausführungscode erzeugt werden, wobei das vorhandene Hintergrundwissen zur Codeoptimierung dem Compiler beigebracht werden muss.

Acknowledgments

Diese Arbeit ist im Rahmen des DFG-finanzierten Projektes "Leichtgewichtige Kompressionsverfahren zur Optimierung komplexer Datenbankanfragen"(LE-1416/26-1) entstanden.

6. LITERATUR

- [1] D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD*, pages 671–682, 2006.
- [2] V. N. Anh and A. Moffat. Inverted index compression using word-aligned binary codes. *Inf. Retr.*, 8(1):151–166, Jan. 2005.
- [3] V. N. Anh and A. Moffat. Improved word-aligned binary compression for text indexing. *IEEE Trans. on Knowl. and Data Eng.*, 18(6):857–861, June 2006.
- [4] V. N. Anh and A. Moffat. Index compression using 64-bit words. *Softw. Pract. Exper.*, 40(2):131–147, Feb. 2010.
- [5] G. Antoshenkov, D. B. Lomet, and J. Murray. Order preserving compression. In *ICDE*, pages 655–663, 1996.
- [6] J. Aronson. Computer science and technology: data compression — a comparison of methods. NBS special publication 500-12, Department of Commerce, National Bureau of Standards, Institute for Computer Sciences and Technology, Washington, DC, USA, June 1977. ERIC Document Number: ED149732.
- [7] M. A. Bassiouni. Data compression in scientific and statistical databases. *IEEE Transactions on Software Engineering*, 11(10):1047–1058, 1985.
- [8] P. A. Boncz, S. Manegold, and M. L. Kersten. Database architecture optimized for the new bottleneck: Memory access. In *VLDB*, pages 54–65, 1999.
- [9] S. Büttcher, C. Clarke, and G. V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. The MIT Press, 2010.
- [10] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.
- [11] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In R. A. Baeza-Yates, P. Boldi, B. A. Ribeiro-Neto, and B. B. Cambazoglu, editors, *WSDM*, page 1. ACM, 2009.
- [12] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In *ICDE Conference*, pages 370–379, 1998.
- [13] D. Habich, P. Damme, and W. Lehner. Optimierung der Anfrageverarbeitung mittels Kompression der Zwischenergebnisse. In *BTW 2015*, pages 259–278, 2015.
- [14] C. Hänsch, T. Kissinger, D. Habich, and W. Lehner. Plan operator specialization using reflective compiler techniques. In *BTW 2015*, pages 363–382, 2015.
- [15] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.
- [16] T. Kissinger, B. Schlegel, D. Habich, and W. Lehner. QPPT: query processing on prefix trees. In *CIDR 2013*, 2013.
- [17] T. J. Lehman and M. J. Carey. Query processing in main memory database management systems. In *SIGMOD Conference*, pages 239–250, 1986.
- [18] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *CoRR*, abs/1209.2137, 2012.
- [19] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.
- [20] H. K. Reghbati. An overview of data compression techniques. *IEEE Computer*, 14(4):71–75, 1981.
- [21] M. A. Roth and S. J. V. Horn. Database compression. *SIGMOD Record*, 22(3):31–39, 1993.
- [22] A. A. Stepanov, A. R. Gangolli, D. E. Rose, R. J. Ernst, and P. S. Oberoi. Simd-based decoding of posting lists. In *CIKM*, pages 317–326, 2011.
- [23] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *SIGMOD Record*, 29(3):55–67, 2000.
- [24] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. *PVLDB*, 2(1):385–394, 2009.
- [25] R. N. Williams. *Adaptive Data Compression*. 1991.
- [26] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications ACM*, 30(6):520–540, 1987.
- [27] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *WWW*, pages 401–410, 2009.
- [28] A. Zandi, B. Iyer, and G. Langdon. Sort order preserving data compression for extended alphabets. In *Data Compression Conference*, pages 330–339, 1993.
- [29] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *WWW*, pages 387–396, 2008.
- [30] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.
- [31] M. Zukowski, S. Heman, N. Nes, and P. Boncz. Super-scalar ram-cpu cache compression. In *ICDE*, page 59, 2006.

Annotation und Management heterogener medizinischer Studienformulare

Victor Christen
Institut für Informatik, Universität Leipzig
christen@informatik.uni-leipzig.de

ABSTRACT

Medizinische Formulare werden für die Dokumentation innerhalb der klinischen Forschung oder der Dokumentation von Patientendaten verwendet. Es existiert eine Vielzahl verschiedener Formulare, die für verschiedene Nutzungszwecke bzw. Anwendungen erstellt werden. Aufgrund der resultierenden Heterogenität ist eine Vergleichbarkeit, eine studienübergreifende Analyse oder eine effiziente Suche nicht ohne weiteres möglich. Um die Interoperabilität der Anwendungen, die auf der Auswertung von Formularen basieren, zu erhöhen, ist eine einheitliche Annotation von medizinischen Formularen mittels einer medizinischen Wissensbasis hilfreich. Eine solche Wissensbasis ist das Unified Medical Language System (UMLS), welches biomedizinisch relevante Konzepte umfasst. Diese Arbeit befasst sich mit der semi-automatischen Annotation von Studienformularen. Basierend auf einem allgemeinen Matching-Workflow, werden weitere Lösungsansätze präsentiert, um die Besonderheiten der Annotation von Studienformularen zu behandeln.

Keywords: semantische Annotationen, medizinische Formulare, klinische Studien, UMLS

1. EINLEITUNG

Medizinische Formulare werden verwendet, um Patientendaten und resultierende Daten innerhalb einer klinischen Studie zu dokumentieren. So werden Studienformulare für die Rekrutierung der Probanden der jeweiligen Studien verwendet, indem die Ein- und Ausschlusskriterien definiert werden. Momentan sind ~ 180000 Studien auf <http://clinicaltrials.gov> registriert, wobei jede Studie eine Menge von Case Report Forms (CRF) umfasst, um die notwendigen Daten zu dokumentieren. Im Allgemeinen werden Formulare einer Studie neu erstellt ohne bereits existierende Formulare wieder zu verwenden.

Aufgrund der hohen Anzahl heterogener Formulare ist eine studienübergreifende Analyse oder der Datenaustausch komplex und nicht ohne weiteres effizient realisierbar. Um

eine einheitliche und strukturierte Repräsentation zu ermöglichen werden die Formulare mit Konzepten von standardisierten Vokabularen wie z.B. Ontologien annotiert [4]. Ontologien sind in der Biomedizin für die Anreicherung von Realweltobjekten weit verbreitet. Die Gene Ontology (GO) wird verwendet, um die Funktionen von Genen und Proteinen zu beschreiben, mithilfe der Medical Subject Headings (MeSH) [8] Ontologie werden wissenschaftliche Publikationen annotiert, und durch die Annotation mit Konzepten der SNOMED CT Ontologie [3] ist eine strukturierte und einheitliche Verwaltung von Patientendaten möglich. Das UMLS [1] repräsentiert eine biomedizinische Wissensbasis, die mehr als 100 biomedizinische Ontologien integriert, wie z.B. SNOMED CT, National Cancer Institute Thesaurus (NCIT) oder MeSH und umfasst ~ 2.8 Millionen Konzepte. Die verschiedenen Anwendungsfälle zeigen das Potential für die Vereinfachung der semantischen Suche und der Datenintegration durch die Annotation von Realweltobjekten mittels der Konzepte von Ontologien. Die Annotation von Formularen hat folgenden Mehrwert:

- **Studienübergreifende Analysen** Eine studienübergreifende Analyse umfasst Studien mit einer ähnlichen Thematik. Die Identifikation ähnlicher Studien ist mithilfe der annotierten Formulare bzgl. der Studien effizient und effektiv durchführbar. Ein Beispiel für eine studienübergreifende Analyse ist der Vergleich der Wirksamkeit und Sicherheit von medikamentbeschichteten Stents und unbeschichteten Stents für Herzkranzgefäße [7]. Bei dieser Analyse wurden 9470 Patienten von 22 randomisierten kontrollierten Studien und 182901 Patienten von 34 Beobachtungsstudien betrachtet. Bei der Auswertung der Daten müssen die Antworten der Fragen der Formulare integriert werden. Die Annotationen der Formulare können für den Integrationsprozess verwendet werden, indem initial durch die Annotationen ähnliche Items identifiziert werden. Die Daten, die die ähnlichen Items betreffen, werden durch weitere Integrationsschritte vereinheitlicht, so dass eine Analyse möglich ist.
- **Erstellung von Formularen** Bisher werden Formulare mit ihren Items für eine durchzuführende Studie neu erstellt. Die Erstellung eines Formulars ist ein aufwändiger Prozess, da z.B. eine unscharfe Formulierung der Ein- und Ausschlusskriterien zu einer mögl. Menge an Probanden führt, die für die Studie nicht vorgesehen waren. Durch die Identifikation bereits annotierter Formulare, die der Thematik der durchzuführenden

Items	Assoziierte UMLS Konzepte		
Patients with established CRF (1) as an indication for the treatment (2) of anemia (3)	<input type="radio"/> yes	1	C0022661 Kidney Failure, Chronic
	<input type="radio"/> no	2	C0039798 therapeutic aspects
	<input type="radio"/> no	3	C0002871 Anemia
Patients who have had prior recombinant erythropoietin (1) treatment whose anemia (2) had never responded (3)	<input type="radio"/> yes	1	C0376541 Recombinant Erythropoietin
	<input type="radio"/> no	2	C0002871 Anemia
	<input type="radio"/> no	3	C0438286 Absent response to treatment
Ulcerating plaque (1)	<input type="checkbox"/> yes	1	C0751634 Carotid Ulcer

Figure 1: Beispiel für die Annotation der Items eines Formulars mit Konzepten des UMLS

Studie entsprechen oder ähneln, können ähnliche Items bei der Erstellung des neuen Formulars wiederverwendet werden.

Ein Formular besteht aus einer Menge von *Items*. Ein *Item* umfasst eine Frage und die dazugehörigen Antwortmöglichkeiten. Eine Antwort hat einen Datentyp wie z.B. Boolean oder String, bei Freitextantworten, oder kann durch einen vordefinierten Bereich wie z.B. das Alter von 0 bis 140 oder eine vorgegebene Menge, die z.B. die möglichen Symptome definiert, eingeschränkt werden. Bei der Annotation eines medizinischen Formulars wird jedem Item eine Menge von Konzepten des UMLS zugeordnet, so dass diese semantisch beschrieben sind. Ein Beispiel für die Annotation eines Formulars für die Ein- und Ausschlusskriterien einer Studie bzgl. Blutarmut ist in Abb. 1 dargestellt. Das Beispiel verdeutlicht die Komplexität der automatischen Identifikation von Annotationen, da z.B. wie in Frage 1 signifikante Wortgruppen zu einem Konzept korrespondieren oder die Frage 3 ein Synonym enthält bzgl. des korrespondierenden Konzepts.

Die *Medical Data Models* Plattform bietet bereits Möglichkeiten für die Erstellung, die Analyse, den Austausch und die Wiederverwendung von Formularen in einem zentralen Metadaten Repository [2]. Aktuell umfasst das Repository mehr als 9000 Versionen von medizinischen Formularen und über 300000 Items. Um die semantische Heterogenität zu reduzieren, werden die Formulare mit Konzepten des UMLS annotiert. Die Annotation der Formulare ist im MDM bisher nur manuell durchführbar und somit sind viele Formulare nicht bzw. unvollständig annotiert, da dieser Prozess sehr zeitintensiv ist.

Die automatische Annotation von Formularen ist thematisch verwandt mit dem Ontologie-Matching, das eine Menge von Korrespondenzen, Mapping genannt, zwischen den Konzepten von zwei oder mehreren Ontologien generiert. Dabei repräsentiert eine Korrespondenz eine semantische Ähnlichkeit zwischen zwei Konzepten. Bei der Annotation von Formularen werden ebenfalls Korrespondenzen ermittelt, wobei eine Korrespondenz zwischen einem Item und einem Konzept ist, welches das Item semantisch beschreibt. Auf dem Gebiet des Ontologie-Matchings existieren eine Vielzahl von Verfahren [11], die eine effiziente und effektive Generierung eines Ontologie-Mappings realisieren, wie z.B. GOMMA [6]. Aufgrund dessen werden Ansätze des Ontologie-Matchings für die Annotation von Studien Formularen verwendet, wie z.B. diverse String-Matchverfahren oder Blocking-Techniken.

Jedoch unterscheiden sich Formulare und Ontologien dahingehend, dass Formulare nicht formal strukturiert sind und aufgrund der besseren Verständlichkeit einen höheren Freitextanteil beinhalten. Die bisherigen Ontologie-Matching Verfahren unterstützen nur unzureichend das Matching von Entitäten mit einem hohen Freitextanteil sowie die Erkennung von n:m Korrespondenzen.

Das Ziel unserer Forschung ist die Verbesserung der Qualität der Annotationen. Des Weiteren soll ein Formular Management System (FMS) realisiert werden, das die Verwaltung der Formulare, Ontologien und der berechneten Annotationen ermöglicht. Das FMS soll zusätzlich das Annotationsverfahren beinhalten sowie Funktionalitäten für die Suche, Analyse und Verifikation der Annotationen von Formularen bereitstellen. Für die Verbesserung der Qualität der Annotationen und der Effizienz der Verfahren sollen folgende Aspekte betrachtet werden.

- **Identifikation von signifikanten Termen und zusammengehörigen Einheiten** Die Fragen innerhalb eines Formulars sind in natürlicher Sprache formuliert. Jedoch sind die Konzepte von Ontologien in einer kompakten Form beschrieben und auf die relevanten Terme beschränkt. Somit ist es notwendig innerhalb einer Frage die signifikanten Terme zu identifizieren. Des Weiteren kann eine Frage aus mehreren semantischen Einheiten bestehen, die jeweils durch ein Konzept beschrieben werden. Aufgrund dessen ist es notwendig diese Wortgruppen zu identifizieren.
- **Wiederverwendung von annotierten Formularen** Da das UMLS eine hohe Anzahl von Konzepten umfasst, ist die vollständige Berechnung des kartesischen Produkts bzgl. aller Fragen eines Formulars sehr zeitintensiv. Durch die Verwendung bereits annotierter Items ist es möglich, die zeitliche Komplexität zu reduzieren, indem zu dem unannotierten Item ähnliche, bereits annotierte Items ermittelt werden. Die assoziierten Konzepte der annotierten Items sind Kandidaten für die Annotation des unannotierten Items.
- **Erweiterte Selektionsstrategien** Beim Ontologie-Matching wird ein Mapping generiert, wobei durch Top-k Selektionsstrategien die Korrespondenzen basierend auf einer berechneten Ähnlichkeit selektiert werden. Da eine Frage durch mehrere Konzepte beschrieben werden kann, die Konzepte jedoch nicht ähnlich sind, sind solche Selektionsstrategien nicht effektiv. Aufgrund dessen sind komplexere Selektionsstrategien erforderlich, die n:m Korrespondenzen berücksichtigen.
- **Verifikationsverfahren** Mithilfe eines Expertenkonsortiums soll die Qualität der Annotationen innerhalb des FMS durch die unterstützte manuelle Verifikation der ermittelten Annotationen erhöht werden. Des Weiteren ist es möglich, dass ein Experte weitere Annotationen vorschlagen kann. Zusätzlich soll ein Verifizierungsverfahren realisiert werden, welches die Widerspruchsfreiheit und die Minimalität der assoziierten Konzepte mit berücksichtigt. So ist z.B. eine Menge von Annotationen nicht korrekt, wenn zwei Konzepte innerhalb dieser Menge als disjunkt definiert sind, dass heißt diese zwei Konzepte besitzen keine gemeinsamen

Instanz. Ein Annotations-Mapping ist nicht minimal, wenn zwei Konzepte dieselbe Thematik beschreiben. Mithilfe der *is_a*-Hierarchie und den Disjunktheitsbeziehungen innerhalb einer Ontologie sind solche Konflikte identifizierbar und durch die Anwendung von Auflösungsstrategien zu beheben.

- **Reduktion der Vergleiche im Annotationsprozess** Aufgrund der hohen Anzahl der Konzepte bei Ontologien ist es sinnvoll die Anzahl der Vergleiche im Annotationsprozess einzuschränken, um eine hohe Effizienz zu erzielen. Es existieren bereits Verfahren, die eine Reduktion der Vergleiche ermöglichen wie z.B. Längenfilter, PPJoin[12] oder Locality Sensitive Hashing (LSH) [5]. Als Ziel unserer Forschung sollen ähnliche Verfahren in den Annotationsprozess integriert werden bzw. neue Verfahren realisiert werden.

Es wurde begonnen ein automatisches Verfahren für die Annotation von Formularen zu implementieren. Die Realisierung eines basalen Workflows und erste Erweiterungen wurden in einer eingereichten Publikation „Annotating Medical Forms using UMLS“ beschrieben. Die Ergebnisse verdeutlichen die Schwierigkeiten der automatischen Annotation und die Vielfalt der Arten von Formularen. So werden für Formulare bzgl. der Qualitätssicherung von medizinischen Geräten gute Resultate erzielt, wohingegen die Qualität der Annotationen für Formulare bzgl. der Ein- und Ausschlusskriterien von Studien ausbaufähig ist.

Der Aufbau dieser Arbeit ist wie folgt gegliedert. In Abschnitt 2 wird das Problem der Annotation von Formularen formal definiert. Der basale Workflow für die Identifikation der Annotationen ist in Abschnitt 3 erläutert. In Abschnitt 4 werden die zu realisierenden Erweiterungen für den definierten Workflow vorgestellt, um die Qualität der Annotationen zu verbessern und die Effizienz des Verfahrens zu erhöhen. In Abschnitt 5 wird konzeptionell die Architektur eines FMS für medizinische Formulare und ihre Annotationen vorgestellt. In Abschnitt 6 wird die Arbeit zusammengefasst.

2. PROBLEMDEFINITION

Das Ziel der semi-automatischen Annotation eines Formulars F ist die Bestimmung eines Annotations-Mappings \mathcal{M} zwischen den Fragen $F = \{q_1, q_2, \dots, q_k\}$ des Formulars und den Konzepten $UMLS = \{cui_1, cui_2, \dots, cui_n\}$ des UMLS. Eine Annotation stellt eine Assoziation zwischen einer Frage und einem Konzept des UMLS dar, wobei eine Frage mit mehreren Konzepten annotiert sein kann. Dabei ist ein Konzept durch einen *Concept Unique Identifier* CUI eindeutig identifizierbar und wird durch Attribute wie z.B. einen Namen oder Synonyme beschrieben. Ein Annotations-Mapping $\mathcal{M}_{F,UMLS}$ ist formal definiert als:

$$\mathcal{M}_{F,UMLS} = \{(q, cui, sim) | q \in F \wedge cui \in UMLS \wedge sim \in [0, 1]\}.$$

Dabei ist sim ein numerischer Wert, der die Ähnlichkeit zwischen einer Frage q und einem Konzept cui repräsentiert.

3. BASIS-WORKFLOW

Unser Ansatz für die Identifikation von Korrespondenzen basiert auf der Berechnung von Stringähnlichkeitsmaßen zwischen den Fragen der *Items* und den Attributen, wie z.B.

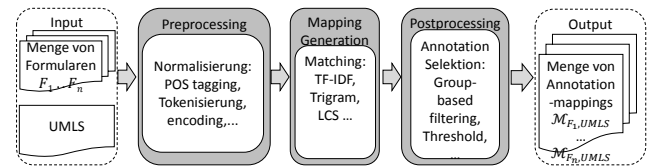


Figure 2: Annotations-Workflow

den Namen und den Synonymen der Konzepte. Der generelle Workflow für die automatische Annotation ist in Abb. 2 dargestellt. Die Eingabe ist eine Menge von Formularen $\{F_1, F_2, \dots, F_n\}$, das *UMLS* und die Ausgabe ist eine Menge von Annotations-mappings

$\{\mathcal{M}_{F_1,UMLS}, \mathcal{M}_{F_2,UMLS}, \dots, \mathcal{M}_{F_n,UMLS}\}$. Zu Beginn werden im *Preprocessing* Schritt die Fragen bzw. Attribute der Konzepte normalisiert. Konkret, werden alle nicht relevanten Wörter entfernt, dazu gehören Präpositionen, Verben und Stoppwörter, die mithilfe eines Part-of-speech Taggers ermittelt werden. Des Weiteren werden alle Tokens klein geschrieben. Um eine effiziente Mapping-Generierung zu ermöglichen werden alle Tokens und Trigramme der Attribute der Fragen eines Formulars bzw. eines Konzepts enkodiert.

Im Schritt *Mapping-Generation* wird eine Menge von Tupeln der Form (q, cui, sim) durch den Vergleich der Fragen mit den Attributen der UMLS Konzepten generiert. Der Vergleich kann durch verschiedene Match-Verfahren realisiert werden wie z.B. Trigramm, TF/IDF oder Longest Common Substring (LCS). Bei einem naiven Ansatz wird das kartesische Produkt bzgl. der Menge der Fragen und der Menge der Konzepte berechnet, jedoch kann durch Pruning-Techniken oder partitionsbasiertes Matching die Anzahl der durchzuführenden Vergleiche reduziert werden [10].

In der *Postprocessing* Phase wird das Mapping durch die Anwendung von Aggregations- und Selektionsstrategien generiert. Im Allgemeinen wird eine Mindestähnlichkeit δ für eine Korrespondenz gefordert, damit diese als korrekt angesehen wird. Da es sich um einen semi-automatischen Prozess handelt, werden die identifizierten Annotationen durch einen Experten verifiziert.

4. ANSÄTZE ZUR ERWEITERUNG DES BASIS-WORKFLOWS

Aufgrund der Besonderheiten bzgl. der Annotation von Formularen, werden im Folgenden die Schwierigkeiten bzgl. des Annotationsprozesses beschrieben und mögliche Lösungsansätze erläutert.

Vorkommen natürlicher Sprache Im Gegensatz zu Ontologien, bei denen die Attribute der Konzepte in einer kompakten Repräsentation dargestellt sind, enthält eine Frage einen hohen Anteil an Freitext.

Ein möglicher Ansatz ist die Identifikation der Schlüsselwörter, die die Frage charakterisieren und ein Konzept des UMLS darstellen. Aufgrund des Vorkommens von Synonymen innerhalb einer Frage, die nicht in einem Konzept des UMLS erfasst sind, ist es nicht möglich durch Stringähnlichkeiten solche Korrespondenzen zu identifizieren. Ein Ansatz ist die Verwendung eines Synonymwörterbuchs, das es er-

laubt alle Tokens innerhalb einer Frage und eines Konzepts durch einen Identifier zu ersetzen. Mithilfe des Identifiers werden Synonyme als gleich angesehen, obwohl die String-ähnlichkeit gering ist. Das Synonymwörterbuch kann entweder durch externe Web-services generiert werden oder durch bereits verifizierte Annotationen erstellt werden.

Komplexe Mappings: Im Gegensatz zu Ontologie-Mappings, die im Allgemeinen aus 1:1 Korrespondenzen zwischen den Konzepten bestehen, werden komplexe Fragen in Formularen durch mehrere Konzepte inhaltlich beschrieben. Um solche komplexen Korrespondenzen zu identifizieren, sind die herkömmlichen Selektionsstrategien wie z.B. die Selektion der Korrespondenz mit der maximalen Ähnlichkeit oder Top-k nicht ausreichend. Für die Bestimmung dieser Korrespondenzen sind komplexe Selektionsstrategien oder entsprechende Vorverarbeitungsschritte sinnvoll. Im Folgenden wird eine Selektionsstrategie und eine mögliche Vorverarbeitung erläutert.

Bei der komplexen Selektionsstrategie werden die Korrespondenzen eines berechneten Mappings gefiltert, indem alle berechneten korrespondierenden Konzepte zu einer Frage bzgl. ihrer Ähnlichkeit gruppiert werden und pro Gruppe das Konzept als korrekt angesehen wird, welches die höchste Ähnlichkeit *sim* zu der Frage aufweist. Alle anderen Konzepte der Gruppe werden aus dem Mapping $\mathcal{M}_{F,UMLS}$ entfernt. Dieser Ansatz ist bereits realisiert und in der eingereichten Publikation vorgestellt.

Des Weiteren sind komplexe Korrespondenzen identifizierbar, wenn die Frage bzgl. ihres Inhalts separiert wird. Eine Wortgruppe oder Teilmenge der Frage repräsentiert dabei eine semantische Einheit und wird zu einem Konzept gemacht. Die Identifikation solcher Gruppen ist beispielsweise durch Named Entity Recognition (NER) Verfahren realisierbar oder durch eine statistische Erhebung von häufig auftretenden Kookkurrenzen innerhalb einer Menge von Formularen.

Größe der Datenquellen Das UMLS umfasst ~ 2.8 Mio. Konzepte, wohingegen ein Formular im Schnitt 50 Fragen enthält. Wenn man 100 Formulare annotiert, bedeutet dies, dass 14 Milliarden Vergleiche durchzuführen sind. Um einen effizienten automatischen Annotationsprozess zu realisieren ist es deshalb notwendig unnötige Vergleiche zu vermeiden.

Ein Ansatz zur Reduktion der Vergleiche ist die Verwendung von Bitlisten. Dabei wird das UMLS in Partitionen aufgeteilt. Bei der Partitionierung werden alle Konzepte bzgl. ihres Namens sortiert und einer Partition mit einer fixen Partitionsgröße (z.B. 100) zugeordnet. Alle Trigramme des Namens und der Synonyme eines Konzepts werden mittels einer Hashfunktion h auf eine Bitposition einer Bitliste der Länge l abgebildet. Die Trigramme werden tokenweise für das jeweilige Attribut erzeugt. Eine Bitlistenlänge $l = 27000$ ist ausreichend, wenn man ausschließlich kleingeschriebene Buchstaben berücksichtigt. Alle Bitlisten der Konzepte einer Partition werden durch die OR-Bitoperation zu einer Bitliste aggregiert. Die resultierende Bitliste ist ein Repräsentant der jeweiligen Partition. Ein Vergleich zwischen einer Frage und den Konzepten einer Partition wird durchgeführt, wenn der Bitlistenvektor der Frage, der ebenfalls durch die Hashfunk-

Bitposition	0	1	2	3	4	5	6	7	8	9	10	11	12	13
P_0 <i>cui1,cui2,cui3</i>	0	0	1	0	0	1	0	1	0	0	1	0	1	0
P_1 <i>cui4,cui5,cui6</i>	1	0	1	1	0	0	0	0	0	1	0	1	0	0
Question <i>q</i>	0	1	1	0	0	0	0	0	0	0	1	0	0	0
$q \wedge P_0$	0	0	1	0	0	0	0	0	0	0	1	0	0	0
$q \wedge P_1$	0	0	1	0	0	0	0	0	0	0	0	0	0	0

$\frac{|q \wedge P_0|}{|q|} = 2/3$

$\frac{|q \wedge P_1|}{|q|} = 1/3$

Figure 3: Beispiel für die Reduktion der Vergleiche mittels Partitionierung und der Repräsentation als Bitliste von Trigrammen

tion h erstellt wird, eine geforderte relative Überlappung $min_overlap[0, 1]$ erzielt. Die Berechnung der Überlappung entspricht der AND-Bitoperation. Die relative Überlappung $rel_overlap$ ist der Quotient aus der Anzahl der Überlappung und der Anzahl der gesetzten Bits der Frage. Somit wird die Anzahl der Vergleiche für eine Frage auf die Anzahl der Konzepte beschränkt, die eine Mindestähnlichkeit bzgl. der Trigramme aufweisen.

Ein Beispiel ist in Abb. 3 dargestellt, dabei wird die Menge der Konzepte $UMLS_example = \{cui1, cui2, \dots, cui6\}$ und eine Frage q betrachtet. Die gegebene Menge wird auf die Partitionen P_0 und P_1 aufgeteilt. Dabei bilden die Trigramme der Konzepte *cui1*, *cui2* und *cui3* mittels einer Hashfunktion h auf die Bitpositionen 2, 5, 7, 10, 12 ab. Analog wird der Bitlistenvektor für die Partition P_1 und die Frage q erstellt. Die relative Überlappung der Bitlisten der Frage q und der Partition P_0 ist $\frac{2}{3}$ und für P_1 $\frac{1}{3}$. Bei einer geforderten relativen Überlappung $min_overlap = 0.5$ wird der Vergleich zwischen der Frage und den Konzepten *cui4*, *cui5* und *cui6* nicht durchgeführt, da die relative Überlappung $rel_overlap = \frac{1}{3}$ ist.

Jedoch ist die Reduktion abhängig von der Effektivität der Partitionierung, so dass im ungünstigen Fall die Konzepte aller Partitionen verglichen werden müssen, wenn die Bitlisten eine hohe Überlappung untereinander aufweisen. Aufgrund dessen, ist eine qualitative Partitionierung bzgl. der Ähnlichkeit der Konzepte essentiell. Eine qualitativhochwertige berechnete Partitionierung ist unabhängig von den zu annotierenden Formularen, so dass diese für eine Vielzahl von Formularen einsetzbar ist.

5. ARCHITEKTUR EINES FORMULAR MANAGEMENT SYSTEMS (FMS)

Es ist geplant, ein Managementsystem zu realisieren, das die Formulare, Ontologien und die dazugehörigen Annotationen verwaltet. Das FMS soll die Möglichkeit bieten Formulare strukturiert zu suchen, ermittelte Annotationen zu verifizieren und neue Formulare zu annotieren. Das Managementsystem soll Wissenschaftlern die Möglichkeit bieten, effizient Formulare zu analysieren und passende Formulare wiederzuverwenden. Die Architektur umfasst eine Datenhaltungsschicht, eine Service-Schicht und eine Frontend-Schicht in Form einer Webanwendung (siehe Abb. 4).

Die Datenhaltungsschicht umfasst die Persistierung der Formulare, Ontologien und der berechneten sowie vorgeschla-

genen Annotationen durch eine relationale Datenbank. Die Service-Schicht umfasst folgende Module: *Import*, *Annotating*, *Search*, *Clustering* und *Verification*.

- **Import** Mithilfe des *Import* Moduls sollen Formulare in das Repository eingepflegt werden, so dass eine effiziente Suche bzw. Annotation möglich ist.
- **Annotating** Das *Annotating-Modul* ermöglicht die Annotation der Formulare des Repositories mit gewählten Ontologien. Des Weiteren sollen bereits annotierte Fragen verwendet werden, um unbekannte Fragen zu annotieren. Diesbezüglich ist ein Suchverfahren innerhalb des *Search-Moduls* notwendig, welches ähnliche Fragen oder Fragmente zu einer gegebenen Frage bzw. Fragments identifiziert. Die Annotationen der identifizierten Fragen sind mit hoher Wahrscheinlichkeit ebenfalls Annotationen für die gegebene Frage. Mithilfe der Wiederverwendung bereits existierender Annotationen wird der Vergleich mit dem kompletten UMLS vermieden.
- **Search** Um eine strukturierte Suche nach ähnlichen Formularen oder Fragen zu ermöglichen, umfasst das *Search-modul* eine Komponente, die basierend auf den Annotationen und der Eingabe einer Menge von Schlüsselwörtern eine explorative Suche nach den gewünschten Formularen bzw. Fragen ermöglicht. Des Weiteren soll dieses Modul eine Komponente umfassen, die eine effiziente Suche nach ähnlichen Fragen ermöglicht. Ein naiver Ansatz wäre die Erstellung einer invertierten Liste bzgl. der Token oder Wortgruppen einer Frage, um für eine unbekannte Frage, die ähnlichsten Fragen zu ermitteln.
- **Clustering** Des Weiteren kann die Effizienz der Suche durch eine Clustering der Formulare bzw. Fragen erhöht werden. In diesem Modul sollen Clustering-Verfahren bereitgestellt werden, die basierend auf den Annotationen eine Gruppierung der Formulare und Fragen ermöglichen.
- **Verification** Da ein automatisches Verfahren keine vollständige Korrektheit gewährleisten kann, soll dieses Modul die Bewertung von Experten in den Qualitätssicherungsprozess bzgl. der Annotationen mit einbeziehen. Ein Experte soll in der Lage sein berechnete Annotationen zu bewerten oder zu ergänzen. Somit soll eine stetige Verbesserung der Qualität der Annotationen im System erzielt werden. Des Weiteren soll mithilfe der verifizierten Annotationen die Effektivität und Effizienz des Annotationsprozesses mittels der Wiederverwendung erhöht werden.

Die Frontend-Schicht wird durch eine Webanwendung repräsentiert, so dass der Anwender die Möglichkeit hat neue Formulare zu importieren, ähnliche Formulare oder Teilfragmente mithilfe einer explorativen Suchfunktion zu ermitteln. Der Anwender soll durch die Eingabe eines Suchterms die Möglichkeit haben, die Menge der Formulare mittels der Annotationen weiter einzugrenzen. Ein Ansatz für eine explorative Suche mittels einer Tag-Cloud ist in eTACTS [9] realisiert. Des Weiteren soll eine Sicht für die Verifikation

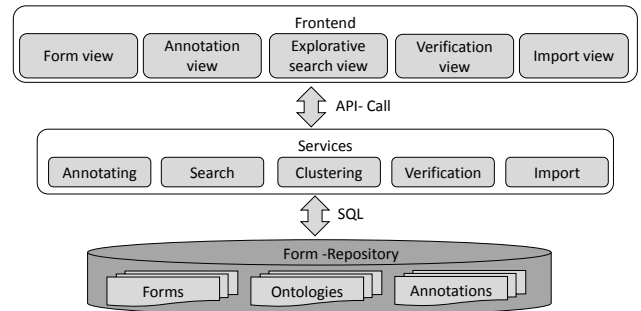


Figure 4: Architektur eines FMS

bereitgestellt werden, die Experten erlaubt einzelne Annotationen zu bewerten.

6. ZUSAMMENFASSUNG

Annotationen sind für die Beschreibung und einheitliche Repräsentation von Formularen essentiell. Durch die Verwendung von Annotationen wird der Datenaustausch, die Integration von Daten der zugrundeliegenden Formulare und die Suche vereinfacht. Um einen effektiven und effizienten Annotationsprozess zu realisieren, sind die bisherigen Methoden des Ontologie-Matching nicht ausreichend. In dieser Arbeit wurde der generelle Workflow für die semi-automatische Annotation vorgestellt sowie Lösungsansätze präsentiert, die die Besonderheiten der Annotation von Formularen behandeln. Um den Nutzen der Allgemeinheit zur Verfügung zu stellen, wurde konzeptionell die Architektur eines Formular Management Systems dargestellt, welches die Möglichkeit bietet neben der Annotation, Formulare oder Fragen basierend auf den Annotationen zu suchen oder zu analysieren. Aufgrund des automatischen Annotationsprozesses soll im Gegensatz zur MDM-Plattform die Vielzahl der Formulare annotiert sein. Da jedoch ein automatisches Verfahren keine vollständige Korrektheit gewährleisten kann, soll mithilfe einer Verification-Komponente ein Expertenkonsortium für die Verifikation mit einbezogen werden.

7. REFERENCES

- [1] O. Bodenreider. The Unified Medical Language System (UMLS): Integrating biomedical terminology. *Nucleic Acids Research*, 32(suppl 1):D267–D270, 2004.
- [2] B. Breil, J. Kenneweg, F. Fritz, et al. Multilingual medical data models in ODM format—a novel form-based approach to semantic interoperability between routine health-care and clinical research. *Appl Clin Inf*, 3:276–289, 2012.
- [3] K. Donnelly. SNOMED-CT: The Advanced Terminology and Coding System for eHealth. *Studies in Health Technology and Informatics—Medical and Care Computetics* 3, 121:279–290, 2006.
- [4] M. Dugas. Missing Semantic Annotation in Databases. The Root Cause for Data Integration and Migration Problems in Information Systems. *Methods of Information in Medicine*, 53(6):516–517, 2014.
- [5] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98,

pages 604–613, New York, NY, USA, 1998. ACM.

- [6] T. Kirsten, A. Gross, M. Hartung, and E. Rahm. GOMMA: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *Journal of Biomedical Semantics*, 2(6), 2011.
- [7] A. J. Kirtane, A. Gupta, S. Iyengar, J. W. Moses, M. B. Leon, R. Applegate, B. Brodie, E. Hannan, K. Harjai, L. O. Jensen, et al. Safety and efficacy of drug-eluting and bare metal stents comprehensive meta-analysis of randomized trials and observational studies. *Circulation*, 119(25):3198–3206, 2009.
- [8] H. J. Lowe and G. O. Barnett. Understanding and using the medical subject headings (MeSH) vocabulary to perform literature searches. *Journal of the American Medical Association (JAMA)*, 271(14):1103–1108, 1994.
- [9] R. Miotto, S. Jiang, and C. Weng. eTACTS: A method for dynamically filtering clinical trial search results. *Journal of Biomedical Informatics*, 46(6):1060–1067, 2013.
- [10] E. Rahm. Towards Large-Scale Schema and Ontology Matching. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping, Data-Centric Systems and Applications*, pages 3–27. Springer Berlin Heidelberg, 2011.
- [11] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, pages 146–171. Springer, 2005.
- [12] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th International Conference on World Wide Web, WWW '08*, pages 131–140, New York, NY, USA, 2008. ACM.

Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data

Muhammad Saqib Niaz
Dept. of Computer Science
Otto von Guericke University
Magdeburg, Germany
saqib@iti.cs.uni-magdeburg.de

Gunter Saake
Dept. of Computer Science
Otto von Guericke University
Magdeburg, Germany
gunter.saake@ovgu.de

ABSTRACT

One of the problems associated with outsourcing data to cloud service providers is the data integrity of outsourced data. In this paper we present data integrity techniques for the outsourced data. Data integrity encompasses the completeness, correctness and freshness of the data. This paper focuses on the Merkle Hash Tree based data integrity techniques. It also presents the techniques for storage and retrieval of Merkle Hash Tree based authentication data to and from cloud data service provider. Critical analysis of the Radix Path Identifiers, a technique for storage of Merkle Hash Trees in the databases, is presented in this paper.

General Terms

Cloud Databases, Security

Keywords

Database Security, Data Integrity, Outsourced Data

1. INTRODUCTION

Data outsourcing means to store your data on third party cloud data service providers. It is cheaper and easier to maintain the data on a cloud data service instead of maintaining it in data owner's own premises. Besides all the benefits, data outsourcing poses numerous security threats to the outsourced data. The list includes but is not limited to data integrity, access privacy and unauthorized access of data. The focus of this paper is data integrity that encompasses completeness, correctness and freshness.

There are three parties involved in these schemes. Data owner (DO), data clients and data service provider (DSP). A DSP provides all the data services and can be trusted with the server availability, timely backups, replications and disaster recovery. But the DSP cannot be trusted with the integrity of outsourced data. A DSP has unlimited access to the data to make it possible for the DSP to forge the data in anyway. It is assumed that the link between the DO

and DSP and the links between clients and DSP are secure using some technique like SSL and the forgery of the data over these links can be easily detectable.

Following are some the features that need to be considered in designing data integrity techniques for outsourced data:

- Computation overhead for the DO
- Computation overhead of the DSP
- Storage overhead of DSP
- Computation overhead of the client
- Storage overhead of the client

The rest of the paper is organized as follows. A basic data integrity technique is presented in Section 2. The Merkle Hash Tree based data integrity scheme is presented in Section 3. Section 4 explains the storage and retrieval technique for Merkle Hash Tree based authentication data. Section 5 presents our analysis of the Radix Path Identifiers technique and the ongoing work on a new technique. Finally, the conclusions follow in Section 6.

2. BASIC TECHNIQUE

For the rest of the paper, we assume that there is a mechanism in place to securely share some data between DO and clients. This data could be the public key of the DO or some hash data. Only the DO can modify the data and the clients have read-only access of the data at the DSP.

The simplest data integrity technique could be to individually sign all the tuples in a data table and storing the signatures in a separate column in the same data table. Afterwards, on query of client, this signature can be sent to a client along with the tuple data. Clients can then check the integrity of the data by verifying the signature of the DO for the associated tuple. This scheme poses a huge computation overhead for the DO and the clients. Despite the computational overhead, it has a linear storage overhead as a distinct signature needs to be stored with each tuple. Still, attacks are possible on this scheme. DSP can delete some valid tuples from the data and the client would never be able to establish this fact. DSP can send an incomplete data set to the client and this forgery will also go undetected at client's end.

Data integrity schemes can be divided into two main categories, i.e., probabilistic and deterministic. Probabilistic approaches for data integrity have been suggested in [7, 11, 12]. The proposed techniques do not require any changes

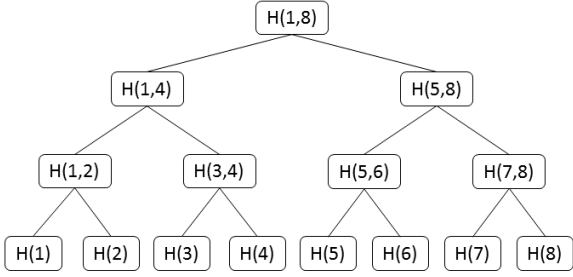


Figure 1: Merkle Hash Tree

at the DSP end, but sometimes the integrity results can be doubtful, as evident from the name.

The second category consists of deterministic approaches that generally base on Authenticated Data Structures (ADS) [6, 5, 10, 1, 2]. ADS based schemes will be the focus of the rest of the paper.

3. MERKLE HASH TREES

Authenticated Data Structures is a technique in which some kind of authentication data is stored on the DSP. On the client's query, a DSP returns the queried data along with some extra authentication data that is then used by the client to verify the authenticity of returned data.

Numerous techniques have been proposed that utilizes ADS for checking the data integrity. Signature aggregation based schemes have been proposed in [6, 5]. These approaches require to modify signatures of all the records, which renders it impractical considering the number of signatures [10]. The authenticated skip lists based approach has been proposed in [1]. A modified Merkle Hash Tree (MHT) based scheme has been proposed in [2] named super-efficient data integrity scheme. In this scheme the main MHT is divided into smaller MHTs and the root hashes of these sub-trees are signed. The purpose of the division in smaller MHTs is to avoid the unnecessary calculation up to the root of the main hash tree.

The Merkle Hash Tree based data integrity techniques for outsourced data are based on a signature scheme proposed by Merkle in [4]. This scheme eliminated the need of digital signatures for the data integrity purposes. MHT based data integrity techniques are based on two sub-components, i.e., Merkle's Signature Scheme and $B+$ Trees.

3.1 Merkle's Signature Scheme

Merkle proposed a Signature Scheme based on a binary tree of hashes in [4]. Figure 1 shows a typical example of an MHT. Each leaf node holds the hash of a data block, e.g., $H(1)$ holds the hash of the data block 1. Internal nodes hold the hash of the concatenated hashes of their children e.g. $H(1,2) = H(H(1) | H(2))$ where '|' indicates concatenation. This scheme is based on the assumption that a safe/trusted way exists to share the root of the tree between the signer and the verifier. To verify the integrity of any data block, the whole tree of hashes does not need to be transmitted to the verifier. A signer transmits the hashes of only those nodes which are involved in the authentication path of the data block under consideration. For example, if the receiver needs to verify the integrity of data block 2 then only $H(1)$, $H(3,4)$ and $H(5,8)$ need to be transferred to

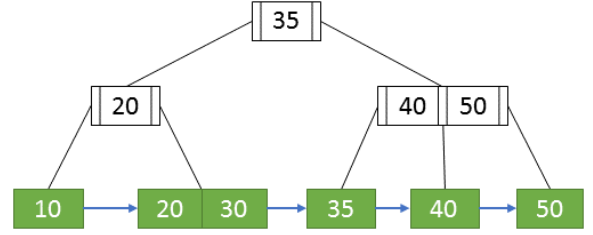


Figure 2: $B+$ Tree of order 3

the receiver. The receiver can calculate the $H(2)$ from data block 2. $H(1,2)$ can then be calculated by using the received $H(1)$ and calculated $H(2)$. In the same way, $H(1,4)$ can be calculated and then $H(1,8)$. The receiver then can compare the calculated $H(1,8)$ with the already shared $H'(1,8)$ and if both the hashes match then the integrity of data block 2 is confirmed.

Some important facts regarding Merkle's Signature Scheme are as follows:

- Security of this signature scheme depends on the security of the hash function.
- Only one hash needs to be maintained/shared securely.
- To authenticate any data block only $\log_2 n$ hashes need to be transferred, where n denotes total number of data blocks.
- In case of integrity checking of a continuous range of blocks, even less than $\log_2 n$ hashes need to be transferred.

3.2 $B+$ Trees

$B+$ trees are a special case of B trees as shown in Figure 2. They are n -ary trees. The root node can be a leaf node or it can be an internal node. Internal nodes only hold keys, they do not hold data. Data always stays in the leaf nodes. Leaf nodes are connected through pointers to form a kind of linked list. This linkage helps in sequential traversal of the data.

Let n be the order of a $B+$ tree. The root node can hold 1 to $n-1$ keys when root node is the only node in the tree. If root node is an internal node then it can have 2 to n child nodes. Internal nodes can have $\lceil n/2 \rceil$ to n child nodes. Leaf nodes can hold $\lceil n/2 \rceil$ to $n-1$ keys [8].

3.3 Data Integrity based on Merkle Hash Tree

Data integrity schemes based on MHT have been designed by replacing binary trees with $B+$ trees in original Merkle's Signature Scheme. The $B+$ tree presented in Figure 2 is used with some modifications. Leaf nodes are linked with direct pointers. Besides keys, leaf nodes also hold the hashes of the data records pointed by corresponding keys. As an example, the leaf node 20, 30 also holds the hashes of the data records of the keys 20 and 30. Internal nodes' pointers also hold the hashes of the concatenated hashes of its child nodes. Like right pointer of the internal node 20 holds the hash of the concatenated hashes of the data records pointed by keys 20 and 30 i.e. $H(20,30) = H(H(20) | H(30))$.

Security of Merkle Hash Tree based data integrity schemes depend on the security of the hash function as in original

Table 1: Employee Data Table

ID	Name	Salary
10	Alice	1000
20	Bob	2000
30	Cindy	3000
40	Dan	3000
50	Eva	2000
60	Felix	1000

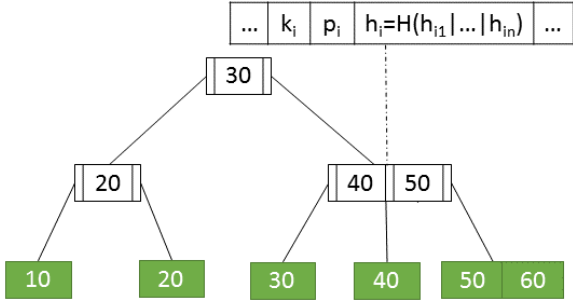


Figure 3: Merkle Hash Tree based on Table 1

Merkle’s signature scheme. This scheme resolves the freshness issue of the query results, too. Each time a DO updates the data in the DSP, a new root hash is calculated based on the newly updated state of the data. By sharing the new root hash with the clients, freshness can be ensured.

3.4 Implementation Issues

A problem associated with the MHTs based data integrity schemes is the efficient storage and retrieval of MHTs in the DSP’s database. Numerous approaches exist to store hierarchical or tree like data in a database, e.g., adjacency list, nested set, nested interval and closure table etc. Each approach has its pros and cons. For this specific problem of storing MHTs, a new technique named Radix Path Identifiers has been proposed in [10].

4. RADIX PATH IDENTIFIER

Consider a data table named *Employee* as shown in Table 1. A Merkle Hash Tree is created based on the data in *Employee* table as shown in Figure 3. Data is inserted in the MHT in an ascending order. Fanout of this $B+$ tree is three that’s why every node holds either one or two keys.

The basic idea is to assign numbers based on a radix to each pointer of the internal node and each key of the leaf node in order to uniquely identify them in a MHT. Radix could be any number equal to or greater than fanout of the MHT. We take 3 as the radix for the MHT created for *Employee* table.

Radix path identifiers have been added to the MHT shown in Figure 3 and the modified MHT is shown in 4. Radix Path Identifier of a pointer or key (in leaf node) depends upon its level in MHT and position in a node. Let l be the level of the MHT. The level of root node is 0 and the level of leaf nodes is the maximum. r_b is the radix base. f denotes the fanout of the MHT. i denotes the index of a pointer or a key in a node, ranging from 0 to f . Radix Path Identifier rpi can be computed using the following equation:

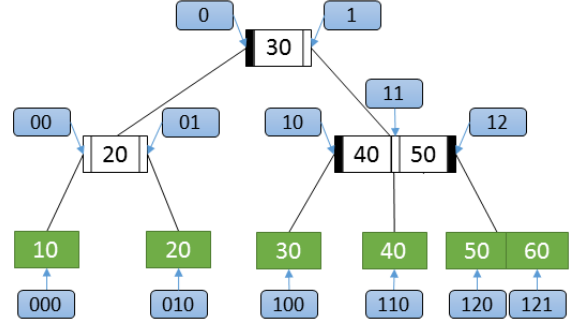


Figure 4: MHT with Radix Path Identifiers

$$rpi = \begin{cases} l & \text{if } l == 0 \\ rpi_{parent} * r_b + i & \text{if } l > 0 \end{cases} \quad (1)$$

For example, to calculate the *RPI* of the key 60 in the leaf node, the level of the key is determined. The level is not zero so the lower part of the equation is applicable and also note that all the calculations done are based on ternary number system. i in this case is 1 as 60 is the second key in the leaf node. *RPI* of the parent is 12 and the r_b is 3. Multiplying rpi_{parent} with r_b gives 120 and adding i into it gives 121, so the *RPI* of the key 60 in leaf node is 121.

The proposed Radix Path Identifier scheme has several important properties:

1. *RPIs* are continuous in nodes, but not continuous among two consecutive nodes. For example, the base-3 numbers 10, 11, 12 are continuous but 110 and 120 are not continuous as shown in Figure 4.
2. From an *RPI*, we can easily find the *RPI* of its parent pointer based on the fact that rpi_{parent} equals to $\lfloor rpi/r_b \rfloor$.
3. From the *RPI* in a node, we can easily calculate the min and max *RPIs* in the node, which are $(\lfloor rpi/r_b \rfloor) * r_b$ and $(\lfloor rpi/r_b \rfloor) * r_b + (r_b - 1)$.
4. From an *RPI* in a node, we can easily compute the index i of the pointer or key in the node, which is $rpi \bmod r_b$.

4.1 MHT Storage in the Database

Two models have been suggested in [10] for storage of Radix Path Identifiers based MHTs in the database. The first method is to store the whole data in one authentication table called Single Authentication Table (SAT). The second method is to store each level of MHT in an individual table called Level Based Authentication Table (LBAT).

4.1.1 Single Authentication Table

In this technique, one table holds the entire authentication data as shown in Table 2. A tuple in this table represents either a pointer in an internal node or a key in a leaf node of the MHT. The authentication table has four columns named as *ID*, *RPI*, *Hash* and *Level*. The *ID* column in authentication table corresponds to the values in *ID* column in *Employee* table. In case of leaf nodes, each key corresponds to a tuple in the *Employee* table so the mapping is straight forward. However in case of internal nodes,

Table 2: Single Authentication Table (SAT)

ID	RPI	Hash	Level
-1	0	hash	2
30	1	hash	2
-1	0	hash	1
20	1	hash	1
-1	3	hash	1
40	4	hash	1
50	5	hash	1
10	0	hash	0
20	3	hash	0
30	9	hash	0
40	12	hash	0
50	15	hash	0
60	16	hash	0

the number of pointers is always 1 more than the number of keys. Because of that one pointer is stored in the table with -1 as the ID. Rest of the pointers are saved with the IDs of the corresponding keys in the node. Considering the left most pointer in each internal node as an extra pointer, -1 is assigned in the ID column of these pointers. The Hash column holds the hashes associated with the pointers in internal nodes and keys in leaf nodes.

The RPI holds the Radix Path Identifier of the pointer in internal node or key in leaf node. RPIs shown in Figure 4 are unique because these numbers are written in base 3 with their preceding zeros. However, for storing RPIs in authentication table, preceding zeros are ignored and the RPIs are converted into base 10 numbers. This results in mapping of different base 3 numbers to the same base 10 numbers. For example, 011 in third level and 11 in second level transform to the same base 10 number i.e. 4. Consequently, the transformed RPIs are unique in a level but they can be repeated among different levels of the tree. In order to distinguish between the same RPIs, a Level column is added to the authentication table.

4.1.2 Level Based Authentication Table

In the LBAT technique, authentication data for each level of an MHT is stored in an individual table. Besides this, an extra table is created that holds the data about the authentication tables i.e. name of the table and its associated level. LBATs for the MHT shown in Figure 4 are shown in Table 3. As every LBAT table represents one level in the tree, there is no need to have a column Level in the authentication tables. Level 0 authentication table has exactly the same number of records as the Employee table so both tables can be merged to form one. RPI and Hash columns have been added at the end of Employee table to hold the authentication data for Level 0.

4.1.3 Performance comparison between both schemes

Considering the table level locks during updates and inserts, it is easier/faster to update authentication data in LBAT than SAT. In the LBAT, as authentication data is stored along with the data record, it makes it straight forward to retrieve authentication data for the leaf level along with the required table data.

4.2 Authentication Data Extraction

Table 3: Level Based Authentication Tables (LBAT)

Emp_2 (Root)			Emp_1		
ID	RPI	Hash	ID	RPI	Hash
-1	0	hash	-1	0	hash
30	1	hash	20	1	hash
-1	0	hash	-1	3	hash
30	1	hash	40	4	hash
			50	5	hash

Employee (Leaf Nodes)				
ID	Name	Salary	RPI	Hash
10	Alice	1000	0	hash
20	Bob	2000	3	hash
30	Cindy	3000	9	hash
40	Dan	3000	12	hash
50	Eva	2000	15	hash
60	Felix	1000	16	hash

Authentication data extracted from LBAT is used to compute the root hash of the MHT. For data extraction from LBAT table, four different ways have been presented i.e. Multi-Join, Single-Join, Zero-Join and Range-Condition in [9]. In all the following methods, data will be extracted from LBATs to verify the authenticity of data record with ID 40. All the required data involved in the authentication path of ID 40 also needs to be extracted from the LBAT and the pointers involved in authentication path are marked with black color in Figure 4.

4.2.1 Multi-Join

In the Multi-Join approach, all the authentication data from respective LBATs are retrieved in a single query. Following SQL statement retrieves the authentication data of record with ID 40. In order to fetch all the authentication data in one query, multiple left outer joins have been used which introduces redundancy in the result.

```
select a0.RPI as RPI0, a0.hash as hash0,
a1.RPI as RPI1, a1.hash as hash1,
a2.RPI as RPI2, a2.hash as hash2
from Employee emp
left join Employee a0 on a0.RPI/3 = emp.RPI/3
left join Emp_1 a1 on a1.RPI/3 = emp.RPI/(3*3)
left join Emp_2 a2 on a2.RPI/3 = emp.RPI/(3*3*3)
where emp.ID = 40;
```

4.2.2 Single-Join

In the Single-Join approach, data from each authentication table is retrieved separately. As ID column does not exist in authentication tables that's why in each query, the authentication table has been joined with the Employee table on column RPI.

```
select e0.RPI, e0.hash
from Employee emp
left outer join Employee e0 on e0.RPI/3 = emp.RPI/3
where ID = 40;

select e1.RPI, e1.hash
from Employee emp
left outer join Emp_1 e1 on e1.RPI/3 = emp.RPI/(3*3)
```

```

where emp.ID = 40;

select e2.RPI, e2.hash
from Employee emp
left outer join Emp_2 e2 on e2.RPI/3 = emp.RPI/(3*3*3)
where emp.ID = 40;

```

4.2.3 Zero-Join

As evident from the name, no tables are joined for querying authentication data. Each table is queried individually. To query each table without any joins, the *RPI* of the record under consideration have to be extracted first and stored in some variable. Afterwards this stored *RPI* is used to extract authentication data from the LBATs.

```

declare @rpid as int;
select @rpid = RPI from Employee where ID = 40;

select RPI, hash from Employee where RPI/3=@rpid/3;

select RPI, hash from Emp_1 where RPI/3=@rpid/(3*3);

select RPI, hash from Emp_2 where RPI/3=@rpid/(3*3*3);

```

4.2.4 Range-Condition

Execution of queries presented in Zero-Join section scans *RPI* for each query. This scan can be replaced with index seek by creating an index on *RPI* and replacing the Zero-Join queries with Range-Conditions. Following queries show how to utilize an index created on *RPI* and efficiently query data from LBAT for authentication of data record *ID 40*.

```

declare @rpid as int;
select @rpid = RPI from Employee where ID = 40;

select RPI, hash from Employee
where RPI >= (@rpid/3)*3
and RPI < (@rpid/3)*3+3;

select RPI, hash from Emp_1
where RPI >= (@rpid/(3*3))*3
and RPI < (@rpid/(3*3))*3+3;

select RPI, hash from Emp_2
where RPI >= (@rpid/(3*3*3))*3
and RPI < (@rpid/(3*3*3))*3+3;

```

Each of the above given queries retrieve authentication data from a specific level of the MHT. The range condition specified in the above queries encompasses all the *RPIs* of the elements present in a node. For example, at level three, a node can have following *RPIs* i.e. *120*, *121* and *122*. In the *RPIs*, the last digit always stays less than the fanout of the tree that is why *3* is mentioned as the upper bound in the query.

4.3 Data Operations

Four data operations, i.e. select, insert, update and delete, will be discussed.

4.3.1 Select

Selection of a single record along with its associated authentication data has already been discussed in detail. Authentication of a continuous range of records is a bit different

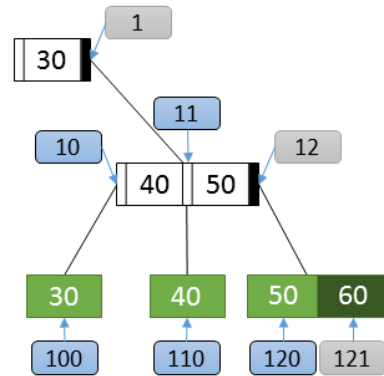


Figure 5: Updating a Record

than the authentication of a single record. Let suppose, our range query returns a set of records from *ID 20* to *40*. User needs to find the two bounding values of the range under consideration. In this case, the two bounding values would be *10* and *50*. The user just needs to fetch the range of records from *ID 20* to *40* without their authentication data. In order to authenticate this range, only the authentication data of the two bounding values is required, i.e., *10* and *50*. By verifying the data integrity of the bounding values, user can be assured of the data integrity of the whole range.

4.3.2 Update

Update is a more complicated operation than Select. In updating a record, along with updating the data table, the user also needs to update the hashes in all the records involved in the authentication path of the updated record. For example, if user updates the data record with *ID 60*, hash of this record will be updated in the *Employee* table along with the data update. In addition user needs to update the hash data in the pointers marked as *12* and *1* as shown in Figure 5.

4.3.3 Insert & Delete

Insert & Delete are more complicated operations than updating a record. Insert & Delete could affect the associated MHT in three different ways. Simplest case could be that the insertion or deletion of a record effects only a single leaf i.e. a key can be added or deleted from a leaf node, in this case only the data involved in the authentication path of the affected leaf node needs to be updated. A little more complicated case could be the change in a subtree of the MHT, in this case all the authentication records of that subtree needs to be updated. In addition the authentication path of the root of the updated subtree also needs to be updated. The most complex case could be the addition or deletion of a new level in the MHT. In case of addition of a new level following needs to be done:

1. Addition of a new LBAT table in the database for the newly inserted level in the MHT.
2. Information regarding the new LBAT table needs to be inserted in the table that holds the data about the LBATs.
3. Update of data in all the LBATs.

5. ANALYSIS & ONGOING WORK

5.1 Analysis

In this section, we analyze the properties of Radix Path Identifiers and identify our next steps based on it. Merkle Hash Tree based data integrity technique guarantees all three aspects of data integrity, i.e., completeness, correctness and freshness [3]. And in doing so, it completely avoids digital signatures which pose a lot of computation overhead. Analysis of the Radix Path Identifier technique is as follows:

- It is assumed that only the DO can change the data.
- All the tests are performed on traditional DBMS i.e. SQL Server [10]. NoSQL databases may perform differently.
- Cached technique for Update results in the lowest overhead. Without caching of data at DO's end, the overhead can go up to 100%.
- Insert at the end of the table gives better result than the Insert at the beginning of the table. Both cases poses significant overhead. No results have been published for Delete.
- In order to modify the data, DO either has to download the whole copy of the table along with authentication data or has to keep the whole data cached at DO's premises.

5.2 Ongoing Work

We are currently starting to work on a Data Integrity Technique that is based on Merkle Hash Trees and Radix Path Identifiers. We want to achieve following goals in this new technique:

- Multiple users should be able to manipulate data.
- A user should not be required to keep a copy of the data in order to modify the data because keeping a copy of data eliminates the purpose of data outsourcing.
- Communication overhead should be minimized to a level near to performing operations in a normal database. For instance, to insert a row of data, a single insert statement should be sent to the database.
- The technique is being designed keeping in view the NoSQL database concepts, too.

6. CONCLUSIONS

Numerous techniques have been proposed in the literature to check the data integrity of the outsourced data to untrusted clouds. Focus of our paper was on Merkle Hash Tree based techniques. Despite a lot of research on MHT based techniques, still insertion and deletion pose a lot of communication and computation overhead. We have also discussed a technique named Radix Path Identifiers to store and retrieve authentication data in the DSP's database. We plan to design a new technique to eliminate the shortcomings of the current data integrity techniques. The main purpose of our technique is to avoid keeping or fetching the copy of whole data in order to run an insert or update statement. We are also going to simplify the process of inserting and deleting the data the way we are used to do in traditional DBMSs.

7. REFERENCES

- [1] G. Di Battista and B. Palazzi. Authenticated relational tables and authenticated skip lists. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 31–46, Berlin, Heidelberg, 2007. Springer-Verlag.
- [2] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *Proceedings of the 2008 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'08, pages 407–424, Berlin, Heidelberg, 2008. Springer-Verlag.
- [3] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 121–132, New York, NY, USA, 2006. ACM.
- [4] R. C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 218–238, 1989.
- [5] M. Narasimha and G. Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In *In International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 420–436. DASFAA, 2006.
- [6] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 407–418, New York, NY, USA, 2005. ACM.
- [7] R. Sion. Query execution assurance for outsourced databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 601–612. VLDB Endowment, 2005.
- [8] A. L. Tharp. *File Organization and Processing*. John Wiley & Sons, Inc., New York, NY, USA, 1988.
- [9] W. Wei and T. Yu. Practical integrity assurance for big data processing deployed over open cloud, 2013.
- [10] W. Wei and T. Yu. Integrity assurance for outsourced databases without DBMS modification. In *Data and Applications Security and Privacy XXVIII - 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*, pages 1–16, 2014.
- [11] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 782–793. VLDB Endowment, 2007.
- [12] M. Xie, H. Wang, J. Yin, and X. Meng. Providing freshness guarantees for outsourced databases. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '08, pages 323–332, New York, NY, USA, 2008. ACM.

Crowdsourcing Entity Resolution: a Short Overview and Open Issues

Xiao Chen
Otto-von-Gueriecke University
Magdeburg
xiao.chen@ovgu.de

ABSTRACT

Entity resolution (ER) is a process to identify records that stand for the same real-world entity. Although automatic algorithms aiming at solving this problem have been developed for many years, their accuracy remains far from perfect. Crowdsourcing is a technology currently investigated, which leverages the crowd to solicit contributions to complete certain tasks via crowdsourced marketplaces. One of its advantages is to inject human reasoning to problems that are still hard to process for computers, which makes it suitable for ER and provides an opportunity to achieve a higher accuracy. As crowdsourcing ER is still a relatively new area in data processing, this paper provides an overview and a brief classification of current research state in crowdsourcing ER. Besides, some open issues are revealed that will be a starting point for our future research.

General Terms

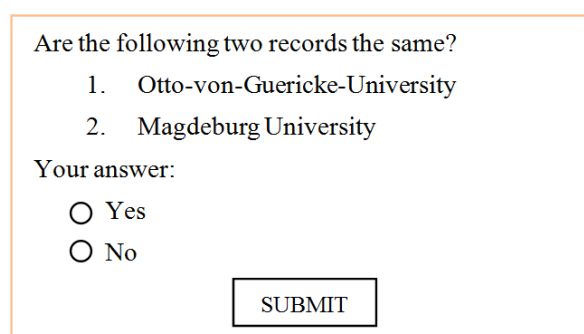
Theory

Keywords

Crowdsourcing, Entity Resolution, Record Linkage

1. INTRODUCTION

Entity resolution (ER) is a process to identify records that refer to the same real-world entity. It plays a vital role not only in traditional scenarios of data cleaning and data integration, but also in web search, online product comparisons, etc. Various automatic algorithms have been developed in order to solve ER problems, which generally include two classes of techniques: similarity-based and learning-based approaches. Similarity-based techniques use similarity functions, where values of record pairs similarity is above a preset threshold are considered to be matched. Learning-based techniques use machine learning modeling ER as a classification problem and are training classifiers to identify matching and non-matching record pairs [14]. However, the accuracy



Are the following two records the same?

1. Otto-von-Guericke-University
2. Magdeburg University

Your answer:

Yes

No

Figure 1: A HIT for ER

of both classes of computer-based algorithms is still far from perfect, particularly for big data without fixed types and structures.

Crowdsourcing was first introduced in the year of 2006 [6] and is gaining growing interest in recent years. Current approaches investigate how Crowdsourcing is suitable to improve the accuracy of ER, because people are better at solving this problem than computers. Although the research on crowdsourcing ER was only started in recent years, there have been several significant research contributions. This paper gives an overview on the current research state of crowdsourcing ER, classifies and compares the related research, and tries to point out some open issues.

The rest of the paper is organized as follows. As crowdsourcing is still not fully established as a technology in the database community, in Section 2 background information on crowdsourcing ER are presented. Then in Section 3, the current state of the research in crowdsourcing ER is presented, different research approaches are classified and compared. After that, open issues are presented in Section 4. Finally, a conclusion is presented in Section 5.

2. CROWDSOURCING

In their early days, computers were mainly used for specific domains that required strong calculation powers. But up until today many tasks, especially those requiring complex knowledge and abstract reasoning capabilities, are not well supported by computers. Crowdsourcing derives its name from “crowd” and “sourcing”, as it outsources tasks to the crowd via the Internet. Many crowdsourced marketplaces, which are represented for instance by Amazon’s Me-

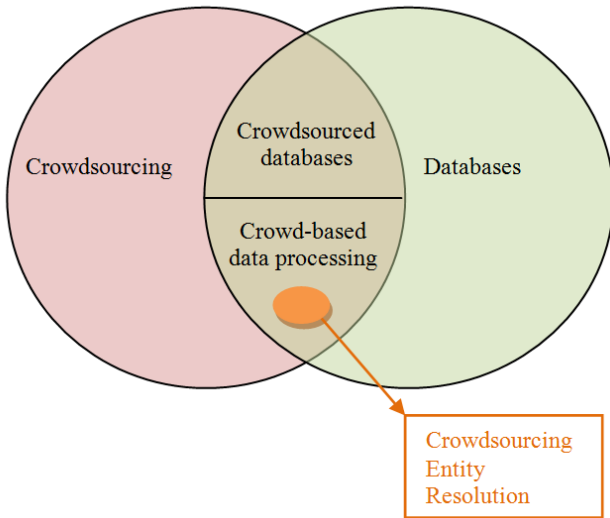


Figure 2: Research directions between crowdsourcing and databases

chanical Turk (MTurk), provide convenience for companies, institutions or individuals recruiting large numbers of people to complete tasks that are difficult for computers or cannot be performed well by computer with acceptable effort. Entity resolution is one of such tasks, which can apply crowdsourcing successfully. People are better at solving ER than computers due to their common sense and domain knowledge. For example, for “Otto-von-Guericke-University” and “Magdeburg University” it is easy for people around Magdeburg to know that both of them refer to the same university, while for computer algorithms they are very likely to be judged as different universities. The tasks on MTurk are called Human Intelligence Tasks (HITs). Figure 1 depicts a possible HIT example of ER. One HIT is usually assigned several workers to guarantee the result quality. Small amounts of money, e.g. 3-5 cents are typical at MTurk, are paid for each worker per HIT.

The current research on crowdsourcing and databases covers two aspects (see Figure 2): on the one hand, databases should be adjusted to support crowd-based data processing (see [3, 10, 11]); on the other hand, crowd-based technology can help to make more effective and broader data processing possible (see [2, 5, 8, 9]). More effective data processing means that the returned query results could be more accurate after leveraging crowdsourcing. Besides, traditional databases cannot handle certain queries such as incomplete data queries or subjective operation queries. By using crowdsourcing, the scope of queries to be answered is broadened. Crowdsourcing ER belongs to the second area, i.e., crowd-based data processing. Specifically, crowdsourcing ER is an important part of join queries. Join queries allow establishing connections among data contained in different tables and comparing the values contained in them [1]. This comparison is not necessarily simple, since there are different expressions for the same real-world fact, more cases with the comparisons among different media and more cases that need human’s subjective comparison. Then ER is a necessary step to better answer the join queries.

3. OVERVIEW AND CLASSIFICATION OF CURRENT RESEARCH STATE

Figure 3 presents a classification for the main research on crowdsourcing ER. From the perspective of crowdsourcing, most of the researches uses crowdsourcing only for identifying matching record pairs. Only one recent approach proposes leveraging crowdsourcing for the whole process of ER, which gives a novel and valuable view on crowdsourcing ER. The workflows of approaches, which leverage crowdsourcing solely for identifying matching record pairs, vary widely. In general, the workflow of crowdsourcing ER has been developed step by step. From the beginning, ER is proposed to be solved by crowdsourcing only, until now a much more complete workflow is formed by integrating different research (see Figure 4). In addition, these approaches focus on different problems. Many novel ideas and algorithms have been developed to optimize crowdsourcing ER. Gokhale et al. presented the Corleone approach and tried to leverage crowdsourcing for the whole process of ER, which is described as hands-off crowdsourcing [4].

In Subsection 3.1 the research leveraging crowdsourcing only for the identification of matching record pairs is described. The corleone approach, which leverages crowdsourcing for the whole process of ER, is then discussed in Subsection 3.2.

3.1 Crowdsourcing for Identifying Matching Record Pairs

Figure 4 depicts a complete hybrid workflow for ER, which contains all proposals to optimize the workflow in crowdsourcing ER. Instead of letting crowds answer HITs that contain matching questions for all records, the first step in the complete workflow is to choose a proper machine-based method to generate a candidate set that contains all record pairs with their corresponding similarities. Then pruning is performed to reduce the total number of required HITs. In order to further reduce the number of required HITs, the transitive relation is applied in the procedure of crowdsourcing, i.e., if a pair can be deduced by transitive relation, it does not need to be crowdsourced. For instance, given three records a, b, and c, the first type of transitive relation is, if a matches b and b matches c, then a matches c. The other type of transitive relation means, if a matches b and b does not match c, then a does not match c. After all record pairs are further identified by crowdsourcing or transitive relations, a global analysis can be performed on the initial result. The global analysis was suggested by Whang et al. [16]. They apply transitive relations only as an example of a global analysis after the process of crowdsourcing. In the case of the integrated ER workflow, which applies transitive relations during the process of crowdsourcing, its global analysis may be implemented by other techniques such as correlation clustering to further improve the result [16]. After a global analysis, the final result is obtained.

The three segments in the dashed box are optional, i.e., in some research, some of them are not included. In the following, specific workflows for different approaches are presented and their contributions are summarized. One important assertion needs to be pointed out: most of the research is done given the assumption that people do not make mistakes when answering HITs. Therefore, each HIT is only assigned to one worker. The problems caused by mistakes

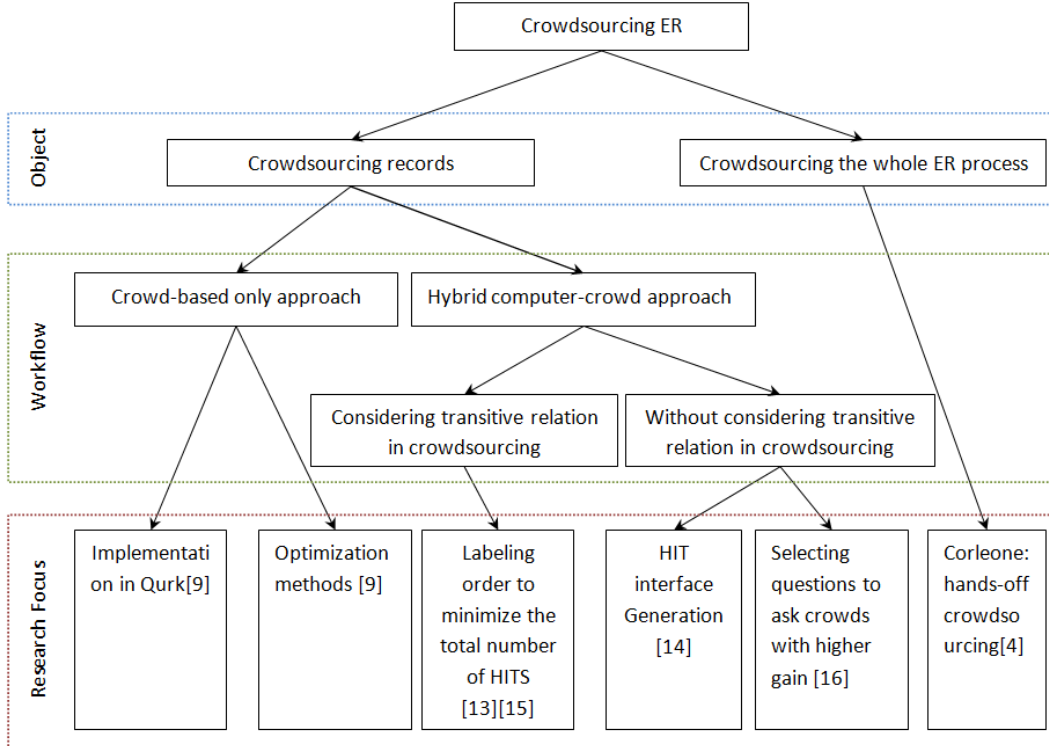


Figure 3: Classification for the research on crowdsourcing ER

that crowds may make are addressed for instance in [7, 12].

3.1.1 Crowd-based Only ER

Marcus et al. proposed to solve ER tasks only based on crowdsourcing [9]. They suggested to using Qurk [10], a declarative query processing system to implement crowd-based joins, and the workflow for ER is only crowd-based, i.e., crowdsourcing is used to complete the whole process of ER. Correspondingly, their workflow does not contain any segment of the three dashed boxes. As mentioned in Section 2, although crowdsourcing can improve the accuracy of ER, it causes monetary costs and is much slower than automatic algorithms. In order to save money and reach lower latency, Marcus et al. proposed two optimization methods, summarized here.

Batching:

the basic interface of crowdsourcing ER is similar to Figure 1, which asks workers to answer only one question in each HIT and is called simple join. The question in simple join is pair-based, i.e., ask workers whether two records belong to the same entity. For a task to identify the same entities from two sets of records respectively with m and n records, $m*n$ HITs are needed. Two optimizations are provided. One is called naive batching. It asks workers to answer b questions in each HIT. Each question in it is also pair-based. In this way, the total amount of HITs is reduced to $(m*n)/b$. The other one is called smart batching. Instead of asking workers whether two records belong to the same entity or not, it asks workers to find all matching pairs from two record lists. If the first list contains a records that are selected from

one data set and the second list contains b records that are selected from the other data set, the total number of HITs are $(m*n)/(a*b)$.

Feature filtering optimization:

for some kinds of records, certain features are useful for being join predicates. For instance, there are two groups of people photos, and if each photo is labeled with the gender of the person on it, only photos with a matching gender are necessary to be asked. The feature should be extracted properly, or it leads to large amounts of unnecessary label work and is unhelpful to reduce the total number of HITs.

Even though the above described optimization methods are applied for ER, the crowd-based only workflow cannot satisfy the development for larger and larger data sets. Therefore, the research in rest of this subsection tends to develop a hybrid workflow for crowdsourcing ER.

3.1.2 Hybrid Computer-Crowd Entity Resolution Without Considering Transitive Relation During the Process of Crowdsourcing

Wang et al. proposed an initial hybrid workflow [14], which contains only the segment in the red dashed box. The pruning technique in it is simply abandoning the pairs with similarities under a threshold. It describes two types of HIT generation approaches: pair-based HIT generation and cluster-based HIT generation. These two types are actually similar to naive batching and smart batching mentioned above in the optimization part of [9]. A pair-based HIT consists of k pairs of records in a HIT. k is the permitted number of record pairs in one HIT, because too many

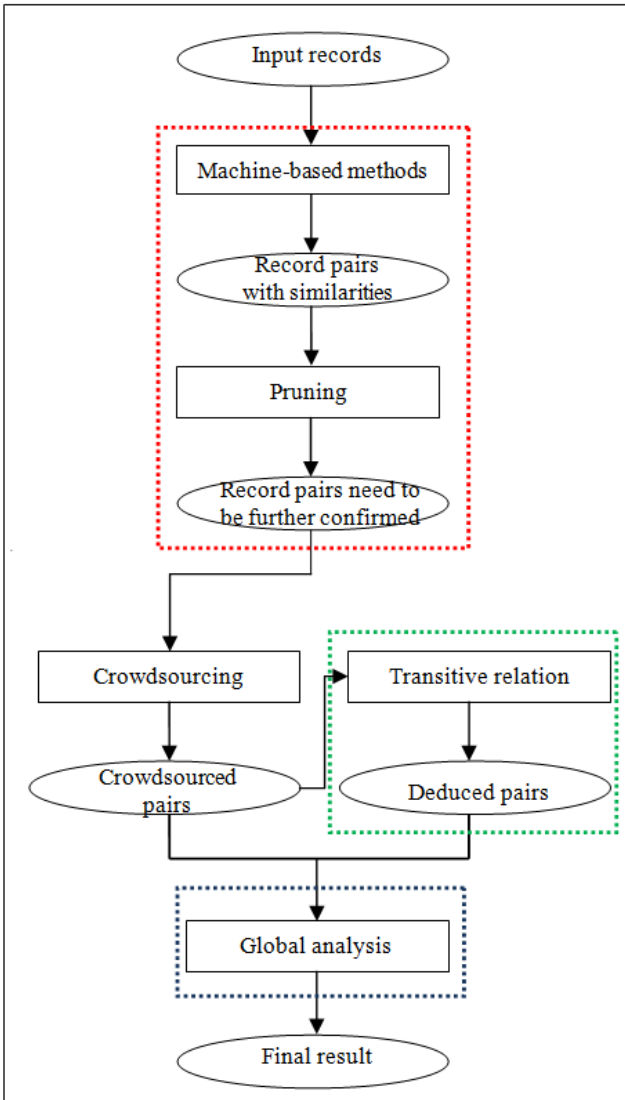


Figure 4: Complete workflow of crowdsourcing ER

pairs may lead to accuracy reductions of crowds’ answers.

A cluster-based HIT consists of a group of k individual records rather than pairs but based on the given pairs. Wang et al. defined the cluster-based HIT generation formally and proves that this problem is NP-hard. Therefore, they reduced the cluster-based HIT generation problem to the k -clique edge covering problem and then apply an approximation algorithm to it. However, this algorithm fails on generating a minimum number of HITs. Therefore, a heuristic two-tiered approach is proposed to generate as few as possible cluster-based HITs. In addition, the following conclusion is made according to their own experimental results.

1. The two-tiered approach generates fewer cluster-based HITs than existing algorithms.
2. The initial hybrid workflow achieves better accuracy than machined based method and generates less HITs than the crowd-based only workflow.
3. The cluster-based HITs provide less latency than pair-

based. HITs.

Wang et al. proved that cluster-based HITs have lower latency than pair-based HITs under the premise of reaching the same accuracy as pair-based HITs, which provides a baseline to prefer cluster-based methods for HIT design. However, the introduced approximation algorithm performs worse than a random algorithm, so it is not presented in this paper.

Whang et al. added a global analysis step to the initial workflow introduced above by Wang et al., but they did not consider using transitive relations to reduce the number of HITs either [16]. The global analysis after the process of crowdsourcing uses transitive relations as an example and permits other techniques, such as correlation clustering to improving the accuracy. Whang et al. focused their research on developing algorithms to ask crowds HITs with the biggest expected gain. Instead of simply abandoning the pairs with similarity scores under a threshold, which is adopted by Wang et al. [14], it develops an exhaustive algorithm, which computes the expected gain for asking crowds questions about each record pair and then chooses the question with the highest estimated gain for crowdsourcing. The exhaustive estimation algorithm is #P-hard. Therefore, the proposed GCER algorithm produces an approximate result within polynomial time and includes the following optimization on the exhaustive algorithm:

1. It only computes the expected gain of record pairs with very high or low similarities.
2. It uses the Monte-Carlo approximation to substitute the method of computing the expected gain for one record pair.
3. The results of the calculations that have already been made before can be shared to the later calculation, instead of recalculation.
4. Instead of resolving all records, only resolving records that may be influenced by the current question.

At last, because the GCER algorithm is still complex, it proposes a very simple half algorithm to choose questions for crowdsourcing. The half algorithm chooses the record pairs with a matching probability closest to 0.5 to ask crowds whether they match or not. In summary, it is non-trivial that Whang et al. uses transitive relations to remedy the accuracy loss caused by such HITs that people cannot answer correctly. However, Whang et al. have not further applied transitive relations during the process of crowdsourcing. Besides, although several optimizations are proposed to improve the performance of the exhaustive algorithm, the complexity of the algorithm is still very high and infeasible in practice.

3.1.3 Hybrid Computer-Crowd ER Considering Transitive Relation During the Process of Crowdsourcing

Vesdapunt et al. [13] and Wang et al. [15] considered the transitive relation in the process of crowdsourcing. Therefore, their workflows contain the segments in the red and green dashed boxes in Figure 4. Their research defines the problem of using transitive relations in crowdsourcing formally and proposes a hybrid transitive-relations and crowdsourcing labeling framework. The basic idea is that, if a

record pair can be deduced according to the matching results that are obtained by crowdsourcing, there is no need to crowdsource it. A record pair can be deduced if transitive relations exist. Two types of transitive relations are defined: positive transitive relations and negative transitive relations, which refer to the two types of transitive relations described at the first paragraph of Section 3.1. A record pair x_1, x_n can be deduced to be a matching pair, only if all pairs among the path from x_1 to x_n are matching pairs. A record pair x_1, x_n can be deduced to be a non-matching pair, only if there exist one non-matching record pairs among the path from x_1 to x_n .

The number of HITs is significantly effected by the labeling order because of applying transitive relations for crowdsourcing ER, i.e. in their work Vesdapunt et al. suggest to first ask crowds questions for real matching record pairs and then for real non-matching record pairs. Because whether two records match or not in reality cannot be known, HITs are first generated for record pairs with higher matching probabilities. However, the latency that a question is answered by crowdsourcing is long, and it is not feasible to publish a single record pair for crowdsourcing and wait for its result to decide whether the next record pair can be deduced or has to be crowdsourced. In order to solve this problem, a parallel labeling algorithm is devised to reduce the labeling time. This algorithm identifies a set of pairs that can be crowdsourced in parallel and asks crowds questions about these record pairs simultaneously, then iterates the identification and crowdsourcing process according to the obtained answers until all record pairs are resolved. The proposed algorithm is evaluated both in simulation and a real crowdsourcing marketplace. The evaluation results show that its approaches with transitive relations can save more monetary costs and time than existing methods with little loss in the result quality.

However, Vesdapunt et al. proved that the algorithm proposed by Wang et al. may be $\Omega(n)$ worse than optimal, where n is the number of records in the database. This proof means the algorithm is not better than any other algorithms, because Vesdapunt et al. also prove that any algorithm is at most $\Omega(n)$ worse than optimal. Therefore, Vesdapunt et al. presented their own strategies to minimize the number of HITs considering transitive relations in the process of crowdsourcing. One simple strategy is to ask crowds questions of all record pairs in a random order without considering the matching probabilities of record pairs. Although this strategy is random, it is proved that it is at most $o(k)$ worse than the optimal, where k is the expected number of clusters. For both approaches a graph-clustering-based method is adopted to efficiently show the relations among possibly matching records. Since the expected number of clusters cannot exceed the number of records, the random algorithm is better than the algorithm proposed by Wang et al. Another strategy called Node Priority Querying is also proved to be at most $o(k)$ worse than the optimal. These algorithms are evaluated using several real-world data sets. The results in different data sets are not stable. However, overall the node priority algorithm precedes the random algorithm and the algorithm proposed by Wang et al. The random algorithm is superior to the algorithm proposed by Wang et al. in some cases.

In summary, both Vesdapunt et al. and Wang et al. considered transitive relations during the process of crowdsourc-

ing, which opens a new perspective to further decrease the number of HITs to reduce the latency and lower the monetary cost. However, their proposed algorithms perform not stably on different data sets and can be optimized or re-designed.

3.2 Applying Crowdsourcing for the Whole ER Process

All approaches introduced so far apply crowdsourcing only for identifying whether record pairs are matching or not. The proposed algorithms have to be implemented by developers. Nowadays, the need for many enterprises to solve ER tasks is growing rapidly. If enterprises have to employ one developer for each ER task, for so many tasks the payment for developers are huge and not negligible. Even in some cases, some private users cannot employ developers to help them complete ER tasks by leveraging crowdsourcing, as they have only a small amount of money. In order to solve this problem, Gokhale et al. proposed hands-off crowdsourcing for ER, which means that the entire ER process is completed by crowdsourcing without any developer [4]. This hands-off crowdsourcing for ER is called Corleone, which can generate blocking rules, train a learning-based matcher, estimate the matching accuracy and even implement an iteration process by crowdsourcing. Each of the above mentioned aspects is settled into a module, and specific implementations are presented to each module.

4. OPEN ISSUES

In this section, open issues on crowdsourcing ER are presented that are from the authors perspective the most important ones to be addressed by future research.

Machine-based methods and pruning approaches: The machine-based methods used in most approaches are similarity-based and pruning methods are simply abandoning the record pairs, whose matching probabilities are above or below given thresholds. Such approaches cannot get satisfactory results in the case of more and more complex data environments, because record pairs with very high matching probabilities may refer to different entities, and vice versa. Therefore, machine-based methods should be extended to learning-based and pruning approaches should be rigorously designed to avoid abandoning the record pairs, which indeed need to be further confirmed.

Only transitive relations applied: currently, transitive relations have been widely adopted in crowdsourcing ER, which are not considered at the early stage of crowdsourcing ER. Other techniques such as correlation clustering could be applied to ER.

Limited comparability of results: both [13] and [15] develop strategies to minimize the number of HITs that are needed to be sent to crowdsourcing. Both of them evaluate their own algorithms using different data sets and compare the performance of their own algorithm with other existing algorithms. However, some of the evaluation results are inconsistent. The reason is that the same algorithm performs varies for different data sets perhaps leading to a different result. Therefore, research to study which algorithm is more suitable for

specific kinds of data sets is necessary for the development of crowdsourcing ER. In addition, new algorithms can be designed, which performs stably on different kinds of data sets.

Only initial optimization strategies: as leveraging crowdsourcing for the whole process of ER, i.e., hands-off crowdsourcing, is just in the beginning of its development. The following optimization techniques can be developed: first, the current hands-off crowdsourcing for ER is based on the setting of identifying record pairs from two relational tables, which may be extended to other ER scenarios. Second, the current strategy to extract samples for generating blocking rules is quite simple, and better sampling strategies should be explored.

Ontologies and indexes: once a decision is made, the knowledge injected by the crowd is widely lost. Another interesting possible research question could be, how this feedback could be gathered and described for instance by an ontology.

5. CONCLUSIONS AND FUTURE WORK

This paper gives an overview of the current research state in crowdsourcing ER. Most of the approaches focus on leveraging crowdsourcing to verify the matching of record pairs. From the early stage of the research to more recent approaches, the workflow was optimized step by step and more aspects were considered for the process of crowdsourcing, which developed from crowd-based only workflow to hybrid computer-crowdsourcing workflow, which considers transitive relations. However, this does not mean that the research on the initial workflow is less significant. In contrast, every approach is valuable and contributes to various research aspects of crowdsourcing ER.

Most recently, a novel perspective to crowdsourcing is proposed, which extends the crowdsourcing object and leverages crowdsourcing not only to verify the matching of record pairs, but also to implement the algorithms, train a learning-based matcher, estimate the matching accuracy and even implement an iteration process. This novel idea makes the crowdsourcing more applicable and further reduces the cost for employing dedicated people.

In Section 4, some important open issues are presented. My future work will focus on the first possible research direction, i.e., exploring techniques to be applied in crowdsourcing ER, such as correlation clustering.

6. ACKNOWLEDGMENTS

I would like to thank the China Scholarship Council to fund this research. I also express my gratitude to Eike Schallehn and Ziqiang Diao for their precious feedback.

7. REFERENCES

- [1] P. Atzeni. *Database systems: concepts, languages & architectures*. Bookmantraa. com, 1999.
- [2] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, pages 225–236. ACM, 2013.
- [3] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.
- [4] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 601–612. ACM, 2014.
- [5] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 385–396. ACM, 2012.
- [6] J. Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.
- [7] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.
- [8] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *Proceedings of the VLDB Endowment*, 6(2):109–120, 2012.
- [9] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.
- [10] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of quirk: a query processor for humanoperators. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1315–1318. ACM, 2011.
- [11] H. Park, H. Garcia-Molina, R. Pang, N. Polyzotis, A. Parameswaran, and J. Widom. Deco: A system for declarative crowdsourcing. *Proceedings of the VLDB Endowment*, 5(12):1990–1993, 2012.
- [12] P. Venetis and H. Garcia-Molina. Quality control for comparison microtasks. In *Proceedings of the first international workshop on crowdsourcing and data mining*, pages 15–21. ACM, 2012.
- [13] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment*, 7(12):1071–1082, 2014.
- [14] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.
- [15] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 229–240. ACM, 2013.
- [16] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *Proceedings of the VLDB Endowment*, 6(6):349–360, 2013.

Toward GPU Accelerated Data Stream Processing

Marcus Pinnecke
Institute for Technical and
Business Information Systems
University of Magdeburg,
Germany
pinnecke@ovgu.de

David Broneske
Institute for Technical and
Business Information Systems
University of Magdeburg,
Germany
dbronesk@ovgu.de

Gunter Saake
Institute for Technical and
Business Information Systems
University of Magdeburg,
Germany
saake@ovgu.de

ABSTRACT

In recent years, the need for continuous processing and analysis of data streams has increased rapidly. To achieve high throughput-rates, stream-applications make use of operator-parallelization, batching-strategies and distribution. Another possibility is to utilize co-processors capabilities per operator. Further, the database community noticed, that a column-oriented architecture is essential for efficient co-processing, since the data transfer overhead is smaller compared to transferring whole tables.

However, current systems still rely on a row-wise architecture for stream processing, because it requires data structures for high velocity. In contrast, stream portions are *in rest* while being bound to a window. With this, we are able to alter the per-window event representation from row to column orientation, which will enable us to exploit GPU acceleration.

To provide general-purpose GPU capabilities for stream processing, the varying window sizes lead to challenges. Since very large windows cannot be passed directly to the GPU, we propose to split the variable-length windows into fixed-sized window portions. Further, each such portion has a column-oriented event representation. In this paper, we present a time and space efficient, data corruption free concept for this task. Finally, we identify open research challenges related to co-processing in the context of stream processing.

Keywords

Stream Processing; GPU; Large Windows; Circular Buffer

1. INTRODUCTION

Traditional Database Management Systems (DBMS) are designed to process a huge collection of data *in rest*. Queries are assumed to run once, deliver a single result set and then terminate immediately. This approach has been shown to be not suitable anymore to meet the requirements of new applications where *high velocity* data has to be processed *continuously* or *complex* operations are preformed on *high*

volume data [1]. Real-time Stream Processing Systems (SPS) support *high velocity* and *high volume* [25] in combination with data flow graphs to achieve continuous computations over data streams in real-time. High throughput is achieved by distributed computing, parallelization, batching strategies and buffer management [1, 3, 21].

Research of the database community showed that the use of GPUs as co-processors are promising for data-intensive systems [14, 13]. Since a graphic card has a dedicated memory, each data to process has to be transferred from main memory to the graphic card memory and, after processing, vice versa. Since this transfer takes time, the transfer-cost might be a bottleneck for some applications [22]. A columnar DBMS is a suitable architecture for GPU acceleration, because it avoids transferring unneeded data (compared to a row store) and also favors data compression [5]. Since the content of a data stream changes rapidly, structured data-based SPSs process a stream of events as a stream of tuples [2, 3, 15, 21, 25]. Nevertheless, there is one fundamental buffering-technique called *windowing* that bounds a possible infinite data to finite portions. As stream-operators process a stream of windows either with a tuple-at-a-time or batch-at-a-time approach, we propose to focus on enabling efficient GPU accelerated operators for structured data in general and, hence, we address on-demand capabilities to convert regular length-variable windows into fixed-sized window portion-streams of columnar represented events.

In this paper, we address a strategy to enable this capability rather than a discussion of GPU-based operators itself. First, we show windowing in stream processing and graphic cards acceleration in DBMSs (Section 2). Afterwards, we continue with our main contributions:

- We examine a concept that splits any stream of variable-length windows into a stream of fixed-size window portions with columnar representation (Section 3)
- We identify open research challenges in context of co-processor-ready stream processing (Section 4)

We will finish with related work (Section 5) and our conclusion that sums up the paper's content (Section 6).

2. BACKGROUND

Motivated by the work of Karnagel et al., who showed a throughput increase for band join computations over streams using GPU acceleration [17], we believe that a general-purpose GPU-ready stream processing framework should be established. The reasons are (1) it enables a single system for regular row-oriented stream processing via CPU and

efficient column-oriented stream processing via GPU and, hence, the load can be shared between both devices, (2) it allows us to run streaming applications on CPU, on GPU, or in mixed-mode and (3) it provides a further abstraction to achieve *what should be done* rather than *how should it be done*. Since time-based windows vary in length, it is not capable to send their contents directly to the GPU. Moreover, the current event presentation leads to memory usage overhead. To highlight these issues in more detail, we examine in the following stream processing and windowing as well as GPU acceleration in the context of DBMSs.

2.1 Data-Intensive Systems and GPUs

With the advent of general-purpose computing on graphics processing units (GPGPU), the usage of a GPU to solve arbitrary problems have become popular. Because of its high parallelism, a GPU can outperform a CPU by orders of magnitude. However, such performance improvements are only possible, if considering the special architecture of the GPU and its programming model.

2.1.1 GPU Architecture

Considering the GPU as an arbitrary co-processor for data processing, we have to care about two things: the *GPU execution model* and the *data transfer cost*. The overall execution model using OpenCL or CUDA uses a host process (on the CPU) and a *kernel* process (mainly a function with implicit parallelism) on the (co-)processor (here the GPU). The host with its host code manages data transfer and schedules the execution on the co-processor. To execute a kernel process on the GPU, there are four things to be done: (1) allocate enough memory for the input and output data, (2) copy the input data to the allocated device memory, (3) execute one or more kernel programs and (4) copy output data back to the host. Hence, batch-processing using a columnar event representation is the right choice for stream processing on the GPU, because the overhead of steps (2) and (4) will be too high otherwise.

2.1.2 Application Scope

Since data copying is an essential step to use a GPU, it is also the biggest bottleneck. For this, the host process (executed on the CPU) schedules the data transfer and provide necessary data over the PCI-Express Bus. However, compared to the bandwidth between the GPU memory and its cores, the bandwidth of the PCI-Express Bus is very low [5]. Hence, the GPU is most efficient for compute-bound problems, where data transfer plays only a minor role [22].

2.1.3 GPU Acceleration in Databases

Unregarded the high transfer costs, executing database operations on the GPU has shown notable performance benefits. For instance, the specialized sort implementation of Govindaraju et al. [13], GPU-TeraSort, achieves a speedup of 10 compared to a CPU implementation. Also, specialized GPU join algorithms perform 3-8x better than reported CPU join implementations [16]. However, there are also operations, e.g., selections, that executed on the GPU may also harm performance, because they are not fully parallelizable [14].

Since not all operators benefit from GPU acceleration, further database research in the direction of load-balancing between co-processors is needed to get a benefit for the operators of a whole query plan, e.g., in CoGaDB [4]. This

attempt relates to our effort in creating an adaptive system that also distributes its work between the CPU and GPU.

2.2 Stream Processing

Stream processing is a paradigm to continuously process, analyze and monitor a (possibly infinite) sequence of data, that is called a stream. Since traditional DBMSs assume *data in-rest*, exact results, queries initiated by humans and no real-time services for applications, they are not adequate for this task [1]. Therefore, around 2003, the database community researched adequate systems for stream processing. The academic projects *Aurora* and its fork *Borealis* are notable here, because they provide an innovative model to deal with data streams and provide well-engineered strategies for scheduling, load shedding, high availability, high performance, and dynamic query management capabilities [1].

Since stream processing is data-driven, a user defined query consumes streams and produces streams containing the results. Those queries are *online* until they are terminated manually. This is in contrast to traditional DBMS queries, which terminate after their execution automatically. Those queries over streams are a loosely coupled data-flow network of operators.

2.2.1 Windowing

Since a stream is a possible infinite sequence of data, it is infeasible to store the complete input data in main memory. Therefore, one fundamental concept is a buffering technique called *windowing* that provides finite stream portions.

2.2.2 Window Types

There are many window variations due to the fact that there are many approaches about what to buffer (time-based or count-based for instance), how to handle new data, when to release old data, and when to trigger an operator [2, 3, 15].

If a window buffers a data stream on a *count*-based approach, the stream of outgoing windows has a fixed-length each. For instance, a count-based jumping window contains a fixed number of tuples and is updated after receiving a predefined number of new tuples. Hence, it contains tuples that occur in a variable time span such that the count of tuples in a window is stable.

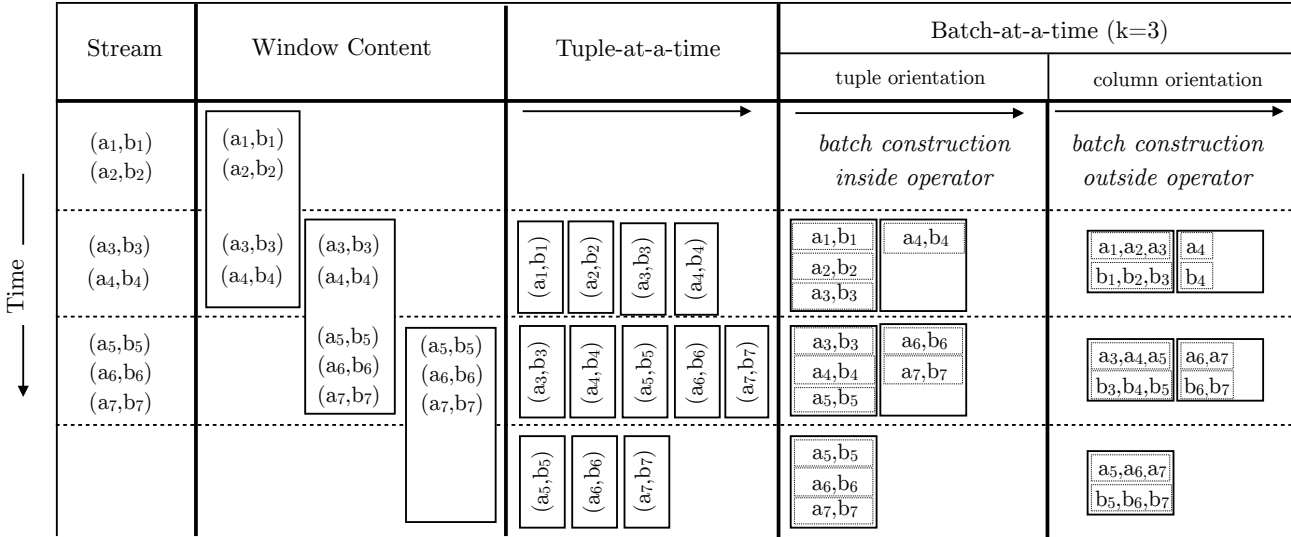
On the other hand, if a data stream is buffered by a *time*-based approach, the tuple count inside a window varies. For instance, a time-based jumping window contains input data related to a given time span (e.g., five minutes) and is updated automatically after a certain time has passed, e.g. two minutes. Hence, it contains tuples that occur in a fixed time span such that the count of tuples in a window is variable.

Since a time-based approach is probably more common than its count-based counterpart [24], the varying length is critical in view of GPU-acceleration since (1) GPU memory is fixed-size allocated and (2) time-based windows might contain thousands of tuples when the time span is large enough and many events occur per instant. As a result, the complete batch might be too large for graphic card memory.

2.2.3 Large-Scale Window Management

We examine in the following large-scale window maintenance and processing as well as load distribution in traditional stream processing.

Figure 1: A continuous stream of 7 events bound to a time-based window of size 2 instants. The Figure shows the processing using a tuple-at-a-time and batch-at-a-time approach. The latter shows row-orientation designed for CPU processing and column-orientation designed for co-processor usage.



Maintaining. To maintain large-scale windows in regular stream processing, the window might be partly swapped to disk. For instance, the storage manager in Aurora tries to hold the most relevant content in main memory while low prioritized ranges are paged to disk using a special replacement policy [1]. Besides *holding* large-scale windows, another important issue is to *process* these windows.

Processing model. One common approach is a tuple-at-a-time approach [15, 20, 21, 25] that does not care about the actual number of tuples in a window, since the required space depends on a single tuple. On the other hand, providing the ability to consume more than one tuple at a time using a batch-like approach [2, 11] could be achieved by iteration over the window’s content inside the operator, utilizing the built-in RAM/disk swapping facilities. Here, an operator consumes k elements as a single block in each iteration, by setting some pointers into the window, until the window has been fully consumed [1]. Obviously, the event representation is unchanged, since it is optimized for CPU processing.

Figure 1 shows different processing approaches. The left hand side shows the content of a stream that is a sequence of $i = 1, \dots, 7$ tuples (a_i, b_i) per time. Here, more than one tuple occur in one instant. The next column shows the actual window content, when applying a time-based window over two time instants. These windows are processed either in a tuple-at-a-time or batch-at-a-time processing manner. The left column for batching shows the actual construction that hold k tuples (row-orientated) per batch by the traditional stream processing approach.

Load Distribution. To increase the throughput per operator, load distribution strategies can be used such that the load to process a window can be shared. This could be achieved per-window (blocks are processed by parallel running operators [23, 25]) or per-block (a single operator

processes partitions of each block in parallel [17, 19]).

3. GPU-READY STREAM PROCESSING

We introduce an operation that we call *bucketing* which transforms the output of windows such that they can efficiently be consumed by GPU-based operators. To distinguish between what a stream-operator consumes, namely a window, we call what a GPU-operator consumes a bucket. We explore the differences in the following sub section. In contrast to windowing logic, these buckets are fixed-size in length independent from the window they listen to, such that we can pipe the window content bucket by bucket to the GPU-based operators. Moreover, we will examine how to flip the event representation from row to column and vice versa efficiently, to avoid unnecessary transfer costs to the graphic card.

We target fixed-sized bucketing of windows with a dedicated operator since this task should not be in the responsibility of any GPU-based operator for several reasons, such as redundant logic might occur otherwise. To show our strategy, we will introduce it step by step. We explore the differences to windowing first and show afterwards how buckets can be created efficiently. The latter approach is more general since we not address the representation-flipping here. How the transformation between row- and column-orientation could be achieved is explained afterwards.

3.1 Motivation

Since SPSs are tuple-oriented, their primary data structure to express an entity’s content is a tuple. Although it is a common structure, it might not be efficient in terms of GPU data transfer cost and memory consumption. Consider for example a stream R with schema $\mathcal{R} = \{A, B\}$ where $\text{dom}(A) = \text{char}(50)$ and $\text{dom}(B) = \text{int}$. R could contain a tuple (the, 268) for instance. Further assume a size of 54 bytes per tuple, where 4 bytes are used for the `int` data type. If an operation only requires B -values, sending entire tuples will waste $\approx 93\%$ of data transfer time and graphic card memory. Therefore, we focus on flipping the event

representation and bounding the size of windows to avoid out-of-memory situations when employing the graphic card.

3.2 Buckets vs. Windows

Windowing is one of the fundamental concepts of stream processing. To be clear about where our proposed operator is different, we examine similarities and differences of buckets and windows.

First of all, the purpose of both concepts is different. Window operators are used to bound a (infinite) stream of data to a collection of data, called a window, that can be processed by set-oriented operators. Hence, a window operator consumes a stream of events and produces a stream of windows. As mentioned earlier, the actual window length can vary depending on a given policy, such as in the time-based approach. In contrast, our operator consumes a stream of windows, each might vary in length, and partitions each window into portions of a user-defined size k . At the same time, the row-wise representation is changed to columnar representation. Each such portion is called a bucket.

Assume a stream R with schema $\mathcal{R} = \{A, B\}$ and consider the right hand side of Figure 1. As one can see, regular batching with a batch-size $k = 3$ contains at most three tuples $(a, b, c) \in R$ per batch entry, while a bucket of size three contains exactly $|\mathcal{R}|$ tuples each with most k components. These components per tuple are from the same domain, while the components of a window tuple are mixed-domains. Therefore, a bucket's tuple t with components from a domain T can forwarded directly to GPU since t is, sloppy saying, an array of length k and type T .

3.3 Portioning Variable-Length Windows

Whereas windowing a continuous data stream leads to several strategies to match different policies, bucketing a window is relatively straightforward. At any time, during runtime or initialization, the subscribers S_1, \dots, S_m that request to consume buckets are known. Also their individual bucket sizes $k_i = \text{size}(S_i) \in \mathbb{N}^+$ are known. We receive a window ω of some size. Therefore, our task is to forward ω in portions of size k_i to each S_i until ω is completely consumed. We assume here w.l.o.g. that the bounds of each window are marked and that the length of each row-oriented tuple is fixed.

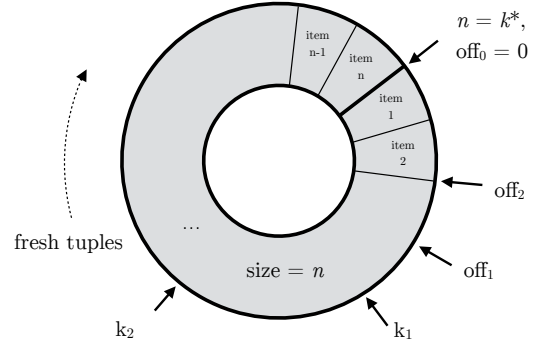
3.3.1 Algorithm

Let $k^* := \max\{k_1, \dots, k_m\}$. We propose to use a fixed-size circular buffer CB that is able to hold items (here, tuple) at k^* positions. For each subscriber S_i we construct a range that we call **slice** $_i := (\text{off}_i, k_i)$ that contains S_i 's current offset off_i and its desired portion size $k_i > 0$. Each **slice** $_i$ points inside CB – its range starts at off_i and ends at $\text{off}_i + k_i - 1$.

At initialization, we set $\text{off}_i = 0$ for all i and $h = 0$. Every time ω outputs a tuple, this tuple is inserted into CB . This insertion operation moves the internal head pointer h of CB further. Hence, h points to the position that will be written next. If $h = k^*$ holds, h is set to 0 since CB is a circular buffer. If after insertion the condition $(h \equiv \text{off}_i + k_i) \bmod k^*$ for a given subscriber S_i holds, the subscriber is notified. In case of a notification to S_i , S_i 's offset off_i is moved to h .¹ When ω is completely read into CB , all S_i are finally notified

¹Notably, each **slice** $_i$ moves as a count-based jumping window of sized length k_i and jump size k_i over the content of ω until ω is fully consumed.

Figure 2: A circular buffer of fixed size k^* and three subscribers. Each subscriber $i \in \{0, 1, 2\}$ has its own slice (off_i, k_i) . Fresh tuples are inserted and slices are moved to partition a possibly infinite data stream. Hence, the stream is split into k_i -length portions that are forwarded to subscribers S_i



about the eventually remaining portion in **slice** $_i$ and about the end of the operation. Afterwards each off_i is set to h such that it is reset to consume the next window. Figure 2 shows such a buffer for $m = 3$ subscribers.

If a new subscriber S_{i+1} occurs during runtime, we set its $\text{off}_{i+1} = h$. Since new data is more important than old data, S_{i+1} starts consequentially with an empty portion. However, if the bucket size of S_{i+1} is greater than the defined k^* , CB has to be resized.

3.3.2 Data Corruptions

Data corruption is overwriting data that has not been sent to subscribers. This is equivalent to a rearrangement of the time order of the data inside at least one **slice** $_i$. This order is inducted by the order of the insertion operations into CB .

We claim that there can be no data corruption and proceed by contraposition. Assume using our construction and a data corruption. Then at least one **slice** $_i$ contains two items a and b with b was inserted after a into CB but **slice** $_i$ states a before b . This happens only if there is a confusion in **slice** $_i$'s view, since data is written sequentially and ordered into a circular buffer CB and **slice** $_i$ points inside CB . A confusion occurs, if from perspective of **slice** $_i$, h moved more than one step. Consequentially, the range of **slice** $_i$ is smaller than expected, since there should be an one-to-one mapping between the actual move of h and the perceived moved of h by **slice** $_i$. This happens if the range of **slice** $_i$ could not be embedded into CB at once and was wrapped around. That happens if CB is too small. This is a contradiction, since CB is as large as the longest requested portion in our construction. Therefore, there is no data corruption possible.

3.4 Flipping the Event Representation

We showed how to achieve an efficient window splitting without changes in the event representation in the previous sub section. Now, we will utilize our approach to flip row-oriented windows into a stream of column-oriented buckets.

Since the complexity of portioning any variable-length window is mainly driven by the actual window length, we suppose to utilize this concept for representation flipping.

The schema $\mathcal{S} = \{A_1, \dots, A_n\}$ for the contents in a window is known. Hence, maintaining one circular buffer as in Section 3.3 for each attribute $A_i \in \mathcal{S}$ splits each incoming tuple into its components. We propose to use a *bucketing*-operator that is responsible for portioning the incoming windows such that it holds n circular buffers, each buffering one attribute of the incoming tuple. Since external subscribers S_1, \dots, S_ℓ are known to the *bucketing*-operator, the operator adds itself as "internal" subscriber to each $1 \leq j \leq n$ circular buffers, delegates the desired size_i for each external S_i to all buffers. This leads to notification of all circular buffers to the operator at the same time once size_i is achieved for some external subscriber S_i . Now, the operator packages the n portions delivered to it into a collection, that is send to the external subscriber.

This allows a chain of operators where each consumes and produces a column-oriented event representation. To convert back to row-oriented event representation, another operator performs the described operations backwards. Hence, to revert the representation flip, another operator reads buckets and outputs them as a stream of regular tuples.

We propose to run these constructions in dedicated operators, since this allows sharing the costs of window portioning between several external subscribers and reverting the operation.

4. OPEN RESEARCH CHALLENGES

Based on existing research and related work, we cannot completely answer all relevant aspects. Therefore, we present two open research challenges in the following section.

4.1 Stream Processing on Modern Hardware

We propose an additional *bucketing*-operator to support event representation changes and window portioning to target GPU-specialized counterparts of existing stream operators. On the other hand, current trends in hardware bring further co-processors (e.g., Intel Xeon Phi or FPGA) with special characteristics into view. Consequently, these co-processors could be used to accelerate stream processing in addition to the GPU. With the increasing amount of different devices, we have to take care of optimized algorithms and execution models for the respected processor to also reach optimized performance [7, 8]. An essential part is to tune a given operator to the used (co-)processor, because each processing device has its own set of optimizations that can be applied [9]. Furthermore, with the availability of different devices, we have to decide where to execute a given operator to reach optimal performance. Here, it is not only important to find the device with the best performance for the given operator, but also to distribute the load between the devices similar to the work of Breß et al. [6]. As far as we can see, further research should be investigated to find limitations and benefits for applied modern hardware in this context.

4.2 Scheduler for Heterogeneous Devices

As proposed by Breß et al. [6], in the context of GPU acceleration for columnar databases, heterogeneous devices with dedicated memory are an opportunity since data transfer from one memory to another is optional. Utilizing OpenCL's possibility to execute kernels on different devices, Karnagel et al. suggest to use a unified kernel and a load balancer that partitions the incoming data and distributes them either to the CPU or GPU depending on the load [18]. This load bal-

ancer contains a job queue and a dictionary that maps tasks to several states. OpenCL takes free jobs as soon as a device is available and deploys it to a specific device on its own. An interesting question is, how an advanced load balancer improves execution performance even further, if the targeted device runs a specialized kernel. This could be achieved with a more high-level load balancer that could decide on its own when to send jobs to device with an unified kernel, and when to to a dedicated device with highly optimized execution code that fits most to the device architecture.

5. RELATED WORK

The design space of DBMSs using GPU as co-processor is well explored [5, 26] and already applied in many applications [4, 10, 13, 22]. He et al. present a novel design and implementation of relational join algorithms for GPUs by introducing a set of data-parallel primitives [14]. Stream processing could benefit from this research results in context of DBMSs, but first progress is made. Karnagel et al. show that a stream band-join might be computed faster with a speedup of nearly 70x using a graphic card [18]. For Complex Event Processing, an approach similar to stream processing, Cugola et al. suggest in 2012 to run the pattern detection automaton on parallelized hardware. They conclude that GPU acceleration can bring speedups in this context but also highlight limitations due to memory restrictions [12]. Hence, current approaches for GPU accelerated stream processing focus on specific topics; instead, we suggest an approach to enable GPU-ready stream processing in general. While we focus on a strategy to handle variable-length windows to enable GPU-operators over fixed-sized batches with a column-orientation ("buckets"), Karnagel et al. use a load balancer that mainly deploys band-join computation tasks to both CPU and GPU. Although these tasks also contain tuple-batches from the input sources, our approach has another focus since we do not actually address load balancing and construct batches outside the responsibility of a specific operator or balancer. Hence, we provide a stream of buckets, built from a stream of windows, that can be consumed by any operator. Bucket streams can be shared by different operators and can form an operator chain before the buckets are converted back to a regular tuple stream.

To enable GPU processing capabilities it is reasonable to process batches, since a GPU might outperform a CPU only if a bulk of data is present at once. Some SPSs do only support a tuple-at-a-time approach [15, 25] such that an internal buffering per operator is required. However, our approach enables those architectures to convert tuple-at-a-time windows to bucket streams. Other SPSs such as Aurora offer batch-processing. Here, each operator stores its output in an output queue that is accessible by subscribers via pointers indicating the current ranges in-use. Aurora cleans up these queues, if a tailed range is not used by any subscriber anymore [1]. Since Aurora manages windowing with output queues, these queues vary as the window content and processing performance of subscriber vary. In contrast, our approach uses a fixed-sized circular buffer and performs window portioning and event representation changes rather than managing window states as Aurora.

6. CONCLUSION

In this paper we motivated and introduced a concept for a dedicated stream processing operator, the *bucketing*-operator, that consumes a stream of length-varying windows and produces a stream of fixed-sized window portions with a column-oriented event representation. We motivated the revertible event representation transposition to match the GPU architecture better, since a GPU uses the SIMD approach and otherwise we would waste memory and increase transfer costs. However, we suggest a strategy to perform *bucketing* using a fixed-sized circular buffer for each attribute of a given schema. This approach is efficient in time and space, since it mainly depends linearly on the actual window content length and could be stored in a predefined sized buffer per-attribute. We ensured here, that data corruption cannot occur using our construction.

Finally, we identified two research questions for processing data streams on modern hardware and scheduling for heterogeneous devices.

7. ACKNOWLEDGMENTS

We thank Bernhard Seeger for fruitful discussions that heavily influenced this work.

8. REFERENCES

- [1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J. h. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: A data stream management system. In *SIGMOD*, page 666, 2003.
- [2] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, K. Ito, R. Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Springer, 2004.
- [3] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. IBM infosphere streams for scalable, real-time, intelligent transportation services. SIGMOD'10, pages 1093–1104, NY, USA, 2010. ACM.
- [4] S. Breß. The design and implementation of CoGaDB: A column-oriented GPU-accelerated DBMS. *Datenbank-Spektrum*, 14(3):199–209, 2014.
- [5] S. Breß, M. Heimel, N. Siegmund, L. Bellatreche, and G. Saake. Exploring the design space of a GPU-aware database architecture. In *GID Workshop @ ADBIS*, pages 225–234. Springer, 2014.
- [6] S. Breß, N. Siegmund, M. Heimel, M. Saecker, T. Lauer, L. Bellatreche, and G. Saake. Load-aware inter-co-processor parallelism in database query processing. *DKE*, 2014.
- [7] D. Broneske. Adaptive reprogramming for databases on heterogeneous processors. In *SIGMOD/PODS Ph.D. Symposium*. ACM, 2015. to appear.
- [8] D. Broneske, S. Breß, M. Heimel, and G. Saake. Toward hardware-sensitive database operations. In *EDBT*, pages 229–234. OpenProceedings.org, 2014.
- [9] D. Broneske, S. Breß, and G. Saake. Database scan variants on modern CPUs: A performance study. In *IMDM@VLDB, LNCS*, pages 97–111. Springer, 2014.
- [10] I. Buck, T. Foley, D. Horn, J. Sugerma, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: Stream computing on graphics hardware. In *SIGGRAPH*, pages 777–786, 2004.
- [11] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing. In *SIGMOD*, pages 668–668, 2003.
- [12] G. Cugola and A. Margara. Low latency complex event processing on parallel hardware. *J. Parallel Distrib. Comput.*, 72(2):205–218, Feb. 2012.
- [13] N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha. GPUteraSort: High performance graphics co-processor sorting for large database management performance graphics co-processor sorting for large database management. In *SIGMOD*, pages 325–336, 2006.
- [14] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, and P. Sander. Relational joins on graphics processors. In *SIGMOD*, pages 511–524, 2008.
- [15] B. Hoßbach, N. Glombiewski, A. Morgen, and B. Ritter, Franz und Seeger. JEPC: The java event processing connectivity. *Datenbank-Spektrum*, 13(3):167–178, 2013.
- [16] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk. GPU join processing revisited. *DaMoN*, pages 55–62, 2012.
- [17] T. Karnagel, D. Habich, B. Schlegel, and W. Lehner. The HELLS-join: A heterogeneous stream join for extremely large windows. In *DaMoN*, pages 2:1–2:7, 2013.
- [18] T. Karnagel, B. Schlegel, D. Habich, and W. Lehner. Stream join processing on heterogeneous processors. In *BTW Workshops*, pages 17–26, 2013.
- [19] H. G. Kim, Y. H. Park, Y. H. Cho, and M. H. Kim. Time-slide window join over data streams. *Journal of Intelligent Information Systems*, 43(2):323–347, 2014.
- [20] J. Krämer. *Continuous Queries over Data Streams - Semantics and Implementation*. PhD thesis, Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, 2007.
- [21] J. Krämer and B. Seeger. Pipes: a public infrastructure for processing and exploring streams. In *Proceedings of the 2004 ACM SIGMOD*, pages 925–926. ACM, 2004.
- [22] A. Meister, S. Breß, and G. Saake. Toward GPU-accelerated database optimization. *Datenbank-Spektrum*, 2015. To appear.
- [23] S. Z. Sbz, S. Zdonik, M. Stonebraker, M. Cherniack, U. C. Etintemel, M. Balazinska, and H. Balakrishnan. The aurora and medusa projects. *IEEE Data Engineering Bulletin*, 26, 2003.
- [24] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4):42–47, 2005.
- [25] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm@twitter. In *SIGMOD*, pages 147–156, 2014.
- [26] Y.-C. Tu, A. Kumar, D. Yu, R. Rui, and R. Wheeler. Data management systems on GPUs: promises and challenges. In *SSDBM*, page 33, 2013.

Where- und Why-Provenance für syntaktisch reiches SQL durch Kombination von Programmanalysetechniken

Tobias Müller
Universität Tübingen
Tübingen, Deutschland
to.mueller@uni-tuebingen.de

ABSTRACT

Das hier vorgestellte Verfahren ermöglicht die Analyse der Data Provenance von beliebigen SQL-Queries. Von der ebenfalls hier skizzierten Implementierung des Verfahrens werden unter anderem unterstützt: Subqueries, Aggregierungen, rekursive Queries und Window Functions. Eingabequeries werden zunächst in eine imperative Programmiersprache übersetzt. Der Programmcode wird mit einem neuen Verfahren analysiert, das auf bekannte Techniken aus dem Bereich der Programmanalyse aufbaut: Program Slicing, Kontrollflussanalyse und abstrakte Interpretation. Dadurch erhält man eine Berechnung von Where- und Why-Provenance auf der Granularitätsebene einzelner Tabellenzellen.

Categories and Subject Descriptors

F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—*Program analysis*; H.2.3 [Database Management]: Languages—*Query Languages*; H.2.4 [Database Management]: Systems—*Relational Databases*

General Terms

Languages

Keywords

Data provenance, SQL, program analysis

1. EINFÜHRUNG

Wir stellen einen neuen Ansatz für die Analyse der *Data Provenance* [5] von SQL-Queries vor sowie eine prototypische Implementierung davon. Der hier präsentierte Ansatz erlaubt eine Analyse beliebiger (lesender) SQL-Queries. Die algebraische Ebene wird nicht berührt, wodurch auch keine algebraischen Restriktionen auftreten. Zum Beispiel ist [1] eingeschränkt auf eine *positive relationale Algebra*.

Die theoretische Anwendbarkeit auf beliebige Queries wird dadurch erreicht, dass wir Eingabequeries zuerst in eine imperative (Turing-vollständige) Programmiersprache überset-

zen, das resultierende Programm in eine *linearisierte Form* (erläutert in Abschnitt 4) umwandeln und anschließend die eigentliche Provenance-Analyse durchführen. Dieser von uns entwickelte Ansatz basiert auf dem Prinzip des *Program Slicing* [10, 3].

Von bisherigen Arbeiten wurde sukzessive der Umfang der analysierbaren SQL-Konstrukte erweitert. Dazu zählen beispielsweise geschachtelte Subqueries [6] sowie Aggregierungen [1]. Mit der in der vorliegenden Arbeit vorgestellten Implementierung ist eine Analyse dieser Konstrukte ebenfalls möglich. Nach dem Wissensstand der Autoren erlaubt unsere Implementierung erstmalig auch die Analyse von Window Functions und rekursiven Queries.

1.1 Data Provenance

Das allgemeine Ziel der Berechnung von Data Provenance ist es, den Ursprung und die zurückgelegten Verarbeitungsschritte des Resultats von Datenverarbeitung sichtbar zu machen. Auf dem Gebiet der relationalen Datenbanken haben wir uns konkret mit der Frage beschäftigt, auf welchen Ursprungsdaten (hier: Tabellenzellen) genau das Ergebnis einer SQL-Query beruht.

Seit [2] unterscheidet die wissenschaftliche Literatur zwei Arten von Provenance:

- *Where-Provenance* charakterisiert sowohl eine direkte Abhängigkeit von Werten durch Kopieren sowie zum Beispiel auch bei Aggregierungen, in denen viele Werte zu einem zusammengefasst werden.
- *Why-Provenance* charakterisiert Abhängigkeiten durch Kontrollflussentscheidungen. Diese liegen zum Beispiel vor, wenn die Existenz eines (Teil-)Ergebnisses von einem anderen Wert abhängig ist, der als Filterkriterium dient (= Semantik einer **WHERE**-Klausel in SQL).

Basierend auf dem hier vorgestellten Ansatz sowie mit unserer prototypischen Implementierung können beide Arten von Provenance berechnet werden.

In Abschnitt 2 werden Beispiel-Queries und deren Analyseergebnisse vorgestellt. Die Abschnitte 3 bis 4 erläutern die Grundzüge des Analyseverfahrens. In Abschnitt 5 beschreiben wir eine konkrete Implementierung. Aus Platzgründen ist nur eine verkürzte Darstellung möglich.

2. BEISPIEL-QUERIES

Im Folgenden werden zwei Beispiel-Queries und die Ergebnisse der mit unserer Implementierung durchgeführten Provenance-Analyse vorgestellt. Die erste Query greift ein Beispiel aus der Literatur auf und die zweite wurde gewählt, um die Mächtigkeit unseres Ansatzes zu demonstrieren.

agencies			
	name	based_in	phone
t_1	BayTours	San Francisco	415-1200
t_2	HarborCruz	Santa Cruz	831-3000

externaltours				
	name	destination	type	price
t_3	BayTours	San Francisco	cable car	\$50
t_4	BayTours	Santa Cruz	bus	\$100
t_5	BayTours	Santa Cruz	boat	\$250
t_6	BayTours	Monterey	boat	\$400
t_7	HarborCruz	Monterey	boat	\$200
t_8	HarborCruz	Carmel	train	\$90

Abbildung 1: Bootstouren-Beispiel: *Where-* und *Why-Provenance* sind markiert mit sowie . Falls beides zutrifft, wird verwendet. Tupel sind mit t_i bezeichnet.

```
SELECT e.name, a.phone
FROM agencies AS a,
externaltours AS e
WHERE a.name = e.name
AND e.type = 'boat'
```

(a) SFW-Query

output		
	name	phone
	HarborCruz	831-3000
	BayTours	415-1200
	BayTours	415-1200

(b) Ergebnis

Abbildung 2: Welche Agenturen bieten Bootstouren an?

2.1 Bootstouren

Diese Beispiel-Query stammt aus [4] und reproduziert die dort gefundene Data Provenance. In Abbildung 1 sind die Eingabetabellen dargestellt: `agencies` enthält Stammdaten von Reiseveranstaltern und `externaltours` enthält deren angebotene Touren. Die Query in Abbildung 2(a) findet diejenigen Veranstalter, die Bootstouren im Angebot haben. Die Ausgaberelation steht in Abbildung 2(b).

Zelle ① ist hier als Where-abhängig von t_5 : `BayTours` markiert. Ein Blick auf die SQL-Query (`SELECT e.name`) bestätigt dieses Ergebnis: denn hier werden Werte aus der Eingabetabelle ins Resultat kopiert. Dies entspricht den Kriterien von Where-Provenance, wie wir sie in Abschnitt 1 angeben haben.

Die Markierungen zeigen außerdem Why-Abhängigkeiten von t_1 : `BayTours`, t_5 : `BayTours` und t_5 : `boat`. Diese drei Werte werden im `WHERE`-Teil der SQL-Query für die Join- und Filterkriterien benutzt.

① und ② zeigen, dass die wertemäßig nicht unterscheidbaren `BayTours` und `BayTours` anhand ihrer jeweiligen Provenance (t_5 oder t_6) unterscheidbar werden.

③ veranschaulicht, dass Data Provenance entgegen der wörtlichen Bedeutung auch in Vorwärts-Richtung funktioniert. Die Farbmarkierungen besagen, dass t_1 : `415-1200` mit zwei Werten von der Ausgabertabelle auf Werteebene zusammenhängt. Konkret wird hier die Telefonnummer zwei Mal kopiert.

2.2 Endlicher Automat

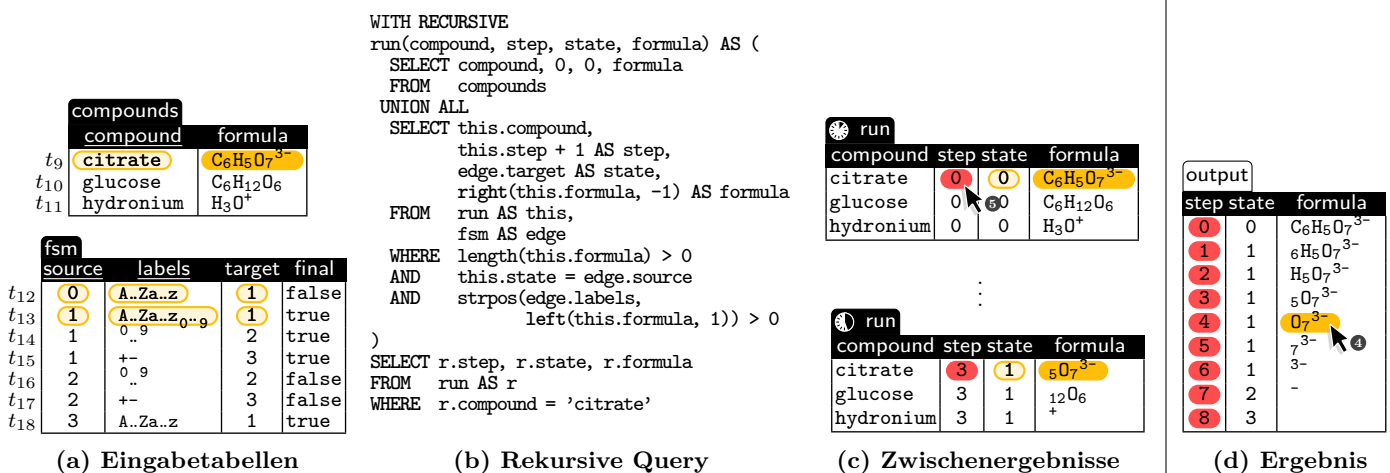
Die hier vorgestellte Provenance-Analyse einer rekursiven Query ist nach dem Wissen der Autoren mit keiner anderen existierenden Implementierung möglich. Die Query besteht auch aus einer Anzahl von Funktionsaufrufen. Interessante Ausschnitte des Ergebnisses unserer Provenance-Analyse werden erneut mit farbigen Markierungen dargestellt.

Abbildung 3(a) zeigt die Eingaberelationen. `compounds` enthält chemische Summenformeln und ihre Bezeichnungen. In Tabelle `fsm` ist ein endlicher Automat codiert, der die Syntax dieser Formeln überprüfen kann.

Die SQL-Query in Abbildung 3(b) führt diesen Automaten aus. Da alle Formeln in `compounds` parallel verarbeitet werden, existieren mehrere Automaten parallel. In jedem Schritt eines Automaten wird das erste Zeichen einer Formel abgeschnitten und der entsprechende Zustandsübergang durchgeführt. Aktueller Zustand und "Restformel" sind in der Tabelle `run` gespeichert, die bei jedem Schritt der Automaten neu berechnet wird. In Abbildung 3(c) sind zwei Versionen von `run` abgebildet: direkt nach der Initialisierung (`step`: 0) und nach drei Schritten (`step`: 3). Wenn die Formeln vollständig verzehrt sind, beendet sich der rekursive Teil der Query. Als Endresultat werden die Ableitungsschritte für `citrate` (siehe Abbildung 3(d)) zurückgegeben.

Die Where-Provenance von ④ zeigt an, von welchen Werten `O73-` abgeleitet wurde. Dazu zählt erst einmal die vollständige Summenformel t_9 : `C6H5O73-`. Aber auch alle Zwischenzustände werden von der Analyse erfasst, wie anhand der Markierungen in Abbildung 3(c) zu erkennen ist.

Die interessantesten Why-Abhängigkeiten von ④ sind innerhalb der Tupel t_{12} und t_{13} zu finden. Das sind nämlich gerade die Kanten des Automaten, die besucht wurden, um `O73-` abzuleiten.



(a) Eingabetabellen

(b) Rekursive Query

(c) Zwischenergebnisse

(d) Ergebnis

Abbildung 3: Endlicher Automat, der die Syntax von chemischen Summenformeln überprüft.

```

1 def query(agencies, externaltours):
2     #FROM clause: read source tables
3     rows = []
4     for tupVar2 in agencies:
5         for tupVar3 in externaltours:
6             rs = {"tupVar2": tupVar2,
7                  "tupVar3": tupVar3,
8                  "tmp": {}},
9             }
10            rows.append(rs)
11 #WHERE clause: compute where predicate
12 rowIdx = 0
13 while rowIdx < len(rows):
14     rs = rows[rowIdx]
15     col4 = rs["tupVar2"]["name"]
16     col5 = rs["tupVar3"]["name"]
17     res6 = col4 == col5
18     col7 = rs["tupVar3"]["type"]
19     val8 = "boat"
20     res9 = col7 == val8
21     res10 = res6 and res9
22     rs["tmp"]["where"] = res10
23     rowIdx = rowIdx + 1
24 #WHERE clause: apply where predicate
25 filtered = []
26 for rs in rows:
27     if rs["tmp"]["where"]:
28         filtered.append(rs)
29 rows = filtered
30 #SELECT clause: compute result columns
31 rowIdx = 0
32 while rowIdx < len(rows):
33     rs = rows[rowIdx]
34     col11 = rs["tupVar3"]["name"]
35     col12 = rs["tupVar2"]["phone"]
36     rs["tmp"]["eval0"] = col11
37     rs["tmp"]["eval1"] = col12
38     rowIdx = rowIdx + 1
39 #SELECT clause: assemble result table
40 ship = []
41 for rs in rows:
42     row = {}
43     row["name"] = rs["tmp"]["eval0"]
44     row["phone"] = rs["tmp"]["eval1"]
45     ship.append(row)
46 return ship

```

Listing 1: Übersetzung der Bootstouren-Query. Es findet (noch) keine Code-Optimierung statt.

Zuletzt wird mit ⑤ noch die Einsicht transportiert, dass die Schrittzahl des Automaten keinerlei Einfluss auf die Ableitung von *citrate* hat. Die Markierungen zeigen lediglich eine Where-Provenance zwischen den Schritten ① bis ⑧ an, die auf das Inkrementieren zurückzuführen ist. Es gibt keine Why-Provenance, das heißt keine (versehentliche) Beeinflussung des Endresultats durch die Schrittzählung.

3. SQL-ÜBERSETZUNG

Die zu analysierenden SQL-Queries werden in unserer Implementierung zunächst in Python-Programme übersetzt. Python hat als Zwischensprache den Vorteil, sehr leichtgewichtig und daher für die weitere Analyse gut zugänglich

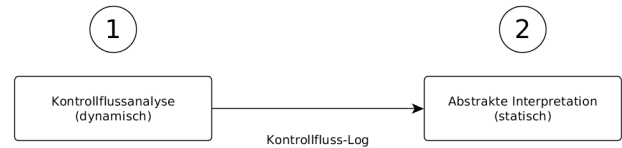


Abbildung 4: Hauptelemente der Provenance-Analyse

zu sein. Es wird außerdem von Kooperationspartnern eingesetzt. Ein Nachteil von Python ist die schlechtere Performance (gegenüber Sprachen wie C). Es gibt jedoch keinen Hinderungsgrund, die mit Hilfe von Python erarbeiteten Techniken der Provenance-Analyse nicht auf andere Sprachen wie LLVM zu übertragen.

Damit würden wir einem aktuellen Forschungstrend in der Datenbank-Community folgen, SQL nicht länger mit dem Volcano-Iterator-Model zu implementieren, sondern Queries just-in-time zu kompilieren (siehe [9]).

Aus Platzgründen wird der von uns verwendete Übersetzer nur anhand eines Beispiels vorgestellt. Listing 1 zeigt, wie die Übersetzung von Query 2(a) aussieht. Je Query oder Sub-Query wird eine eigene Python-Funktion erzeugt. Die Argumente und Rückgabewerte sind Tabellen, die in Python als Listen von Dictionaries implementiert sind. Die SQL-Klauseln einer Query werden in eine Reihenfolge gebracht, die eine Berechnung im imperativen Paradigma erlaubt: SFW wird beispielsweise zu FWS.

4. PROVENANCE-ANALYSE IN ZWEI STUFEN

Wie in Abbildung 4 dargestellt, teilt sich die Provenance-Analyse im Wesentlichen auf zwei Schritte auf: ① *Kontrollflussanalyse* (dynamisch, zur Laufzeit) und ② *abstrakte Interpretation* (statisch, zur Kompilierzeit).

Die Kontrollflussanalyse ist dafür zuständig, alle für den Kontrollfluss benötigten Prädikate zu bestimmen und deren Werte zu speichern. Beispielsweise würde bei der Ausführung einer *if*-Anweisung die zugehörige Kontrollflussinformation darin bestehen, ob entweder der *if*- oder der *else*-Rumpf ausgeführt wird. Die Information kann durch einen einzelnen booleschen Wert codiert werden.

In Phase ② findet die eigentliche Provenance-Analyse statt. Hier wird das Kontrollfluss-Log benutzt, um den tatsächlichen Ausführungspfad nachvollziehen zu können, den das Programm zur Laufzeit genommen hat.

In Abschnitt 4.1 wird die Motivation für die soeben skizzierte Struktur geschildert sowie ein Bezug zu Ergebnissen der theoretischen Informatik hergestellt, die einer Programmanalyse harte Grenzen setzt. Abschnitt 5 erläutert die beiden Analyseschritte genauer sowie deren Implementierung in Python.

4.1 Linearisierung

Eine rein statische Provenance-Analyse ist im Allgemeinen für Python-Programme nicht möglich, denn Python ist eine Turing-vollständige Programmiersprache. Der *Satz von Rice* besagt, dass nicht-triviale Laufzeiteigenschaften (wie Data Provenance) für allgemeine Turing-Maschinen nicht algorithmisch entscheidbar sind.

Um den Konsequenzen des Satzes von Rice zu entgehen, ändert dieses Verfahren die Voraussetzungen. Wie in Abbildung 4 dargestellt, wird der zur Laufzeit aufgezeichnete Kontrollfluss an die abstrakte Interpretation übergeben.

Zu Kontrollflussanweisungen zählen unter anderem **if**- und **while**-Anweisungen. Für diese Konstrukte besteht das zugehörige Kontrollfluss-Log lediglich aus einer Folge von booleschen Werten:

- Wird entweder **if** oder **else** ausgeführt?
- Wird der Rumpf von **while** (nochmal) ausgeführt oder wird die Schleife beendet?

Dieses Log steht also während der statischen Analyse zur Verfügung. Das heißt, die statische Analyse weiß für jedes **if x:**, ob es in Wirklichkeit entweder ein **if True:** oder **if False:** ist - je nachdem, ob **True** oder **False** im Log steht.

Mit dem Kontrollfluss-Log findet deshalb eine Linearisierung des Python-Programms statt. Die Abfolge der Anweisungen im Programm ist statisch festgelegt, weil der Kontrollfluss festgelegt ist. Kontrollfluss-Konstrukte verhalten sich nun transparent und im einfachsten Fall besteht die restliche Analyse des Programms nur noch darin, Zuweisungen an Variablen zu betrachten.

Dadurch liegt in ② keine Turing-Vollständigkeit mehr vor. Der Satz von Rice gilt nicht mehr und eine Provenance-Analyse ist (quasi-statisch) möglich.

Das Beispiel in Abschnitt 5 wird das verdeutlichen.

4.2 Granularität und Auflösung

Als *Granularität* (oder *level of detail*) wird in [7] bezeichnet, was die kleinstmöglichen Datenstrukturen sind, die in einer Provenance-Analyse berücksichtigt werden. Meist sind das entweder Tupel oder Tabellenzellen. In dem hier besprochenen Ansatz besteht die Granularität in Zellen.

Orthogonal dazu wollen wir den Begriff *Auflösung* verwenden, um damit die Größe der Programmfragmente zu bezeichnen, für die am Ende der Analyse eine Data Provenance ausgegeben wird.

Beispielsweise könnte es in einer niedrigen Auflösung so sein, dass das Programm nur als Ganzes analysiert wird. Das heißt, die Data Provenance bezieht sich gerade auf die Ein/Ausgabedaten des Programms selbst und zeigt an, wie die Ausgabedaten von den Eingabedaten abhängig sind. Eine andere Variante bestünde darin, für jeden einzelnen Ausdruck in diesem Programm eine Data Provenance auszugeben. Das heißt, für einen Ausdruck wie `b+c` wird als Ergebnis die Abhängigkeit von den Einzelwerten `b` sowie `c` ausgegeben. Hier ist die Auflösung sehr hoch.

Von uns wurde als Auflösung die Ebene von Funktionsaufrufen benutzerdefinierter Funktionen gewählt. Im Ergebnis der Provenance-Analyse sind deshalb die Parameter und Rückgabewerte von Funktionsaufrufen aufgeführt sowie die dazugehörige Data Provenance.

5. IMPLEMENTIERUNG

Als Grundlage für die Implementierung der Provenance-Analyse dient *CPython* in Version 3.4. Dabei handelt es sich um die stabile und zum Zeitpunkt des Schreibens dieses Artikels aktuelle Referenzimplementation von Python. Intern übersetzt sie Quelltext in Bytecode, der anschließend in der zugehörigen virtuellen Maschine (VM) ausgeführt wird. Als Zwischenschritt während der Übersetzung erzeugt *CPython* einen Abstract Syntax Tree (AST), auf dessen Basis wir das Analyseverfahren implementiert haben.

In Abbildung 5 ist dargestellt, wie die einzelnen Python-Komponenten zusammenarbeiten. Mit weißem Hintergrund dargestellt sind die ein/ausgehenden Datensätze (Relationen) sowie der Python Programmcode, der analysiert wer-

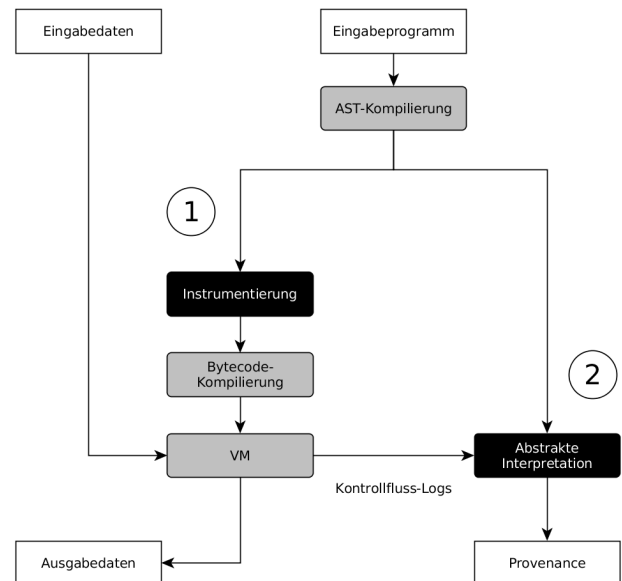


Abbildung 5: Komponenten der Python-Implementierung

den soll. Grau hinterlegt sind Komponenten der *CPython*-Implementierung selbst, die unmodifiziert verwendet werden. In schwarz ist, was wir zusätzlich implementiert haben.

Während der Analyse wird das zu einem AST kompilierte Eingabeprogramm zwei Mal verarbeitet. In Schritt ① wird es zunächst instrumentiert. Dabei werden zusätzliche Python-Anweisungen eingefügt, die einerseits die vorhandene Funktionalität nicht verändern und andererseits für das Schreiben des Kontrollfluss-Log zuständig sind. Das so modifizierte Programm wird zu Bytecode kompiliert und anschließend mit der VM ausgeführt. Während der Ausführung werden Eingabe/Ausgabedaten gelesen/geschrieben sowie das Kontrollfluss-Log produziert. Schritt ① wird in Abschnitt 5.1 genauer beschrieben.

In Schritt ② wird das unmodifizierte (nicht instrumentierte) Eingabeprogramm erneut hergenommen und mit Hilfe des Kontrollfluss-Logs die eigentliche Provenance-Analyse durchgeführt. Eine genaue Beschreibung dieses Teils folgt in Abschnitt 5.2. Bemerkenswert ist, dass hier keine Eingabedaten benötigt werden. Die abstrakte Interpretation findet auf symbolischer Ebene statt, das heißt es wird lediglich ermittelt, wie die im Programm benutzten Variablen voneinander abhängig sind. Dies genügt, um sowohl Why- als auch Where-Provenance zu berechnen.

5.1 Instrumentierung

Allgemein gesprochen müssen diejenigen Sprachkonstrukte instrumentiert werden, deren Verhalten bezüglich Data Provenance sich nicht durch rein statische Analyse bestimmen lässt. Bisher instrumentieren wir zwei Kategorien von Sprachkonstrukten: (i) Kontrollfluss und (ii) Indexzugriff.

Listing 2 zeigt ein bereits instrumentiertes Programmfragment, das je ein Beispiel für beide Fälle enthält. Die Funktion `dow()` erhält als Argumente den Wochentag als Zahlenwert (`num = 0...6`) und eine Liste mit Namen von Wochentagen. Sie liefert eine String-Repräsentation zurück. Sie unterstützt außerdem zwei verschiedene Datumsformate: Wochenbeginn am Montag (`fmt = True`) oder am Sonntag (`fmt = False`).

```

1 days = ["Sun", "Mon", "Tue",
2         "Wed", "Thu", "Fri", "Sat"]
3 def dow(num, fmt, days):
4     if fmt:
5         #true: week starts on Monday
6         logCtrl(True)
7         pos = (num+1) % 7
8     else:
9         #false: week starts on Sunday
10        logCtrl(False)
11        pos = num
12        logIdx(pos)
13        res = days[pos]
14        return res
15 dow(4, True, days) #returns: "Fri"

```

Listing 2: Beispiel für instrumentierten Programmcode. Die Instrumentierungsanweisungen sind umrandet.

Die beiden instrumentierten Anweisungen sind: (i) das `if/else`-Statement und (ii) der Ausdruck `days[pos]` (Zeile 13). An beiden Stellen wäre eine rein statische Analyse nicht ausreichend. Bei (i) ist nicht bekannt, welcher Rumpf überhaupt ausgeführt wird und bei (ii) ist unbekannt, auf welches Listenelement von `days` zugegriffen wird.

Im Gegenzug ist die Data Provenance einer Anweisung wie zum Beispiel `pos = (num+1) % 7` schon bei statischer Analyse klar: der Wert von `pos` ist Where-abhängig von `num`. In diesem Fall ist keine Instrumentierung erforderlich.

Wird `dow()` mit den Parametern von Zeile 15 aufgerufen, werden folgende Logs produziert:

- *Kontrollfluss*: [True]
- *Indices*: [5]

In diesem bewusst kurz gehaltenem Beispiel bestehen beide Logs aus jeweils nur einem Element. Für längere Programme würden weitere Einträge einfach in der Reihenfolge hinzugefügt, in der die Instrumentierungsanweisungen aufgerufen werden. In Schritt ② werden die Einträge in genau derselben Reihenfolge wieder gelesen. Deshalb braucht das Log nur sequenziell zu sein. Die Zuordnung eines Log-Eintrags zur zugehörigen Anweisung im analysierten Programm ist implizit gegeben.

5.2 Abstrakte Interpretation

Nachdem die Kontrollfluss-Logs erzeugt worden sind, kann jetzt die eigentliche Ableitung der Data Provenance stattfinden. Dazu werden wie in Abbildung 5 dargestellt, lediglich das Eingabeprogramm zusammen mit den Logs verwendet. Daten sind an dieser Phase nicht beteiligt. Dementsprechend werden auch keine Berechnungen von Werten ausgeführt, was Systemressourcen spart.

Die *abstrakte Interpretation* wird durchgeführt, indem alle Anweisungen des Programms in derselben Reihenfolge nachvollzogen werden, in der sie auch zur Laufzeit ausgeführt wurden. Die richtige Reihenfolge einzuhalten ist dank des aufgezeichneten Kontrollflusses sehr einfach. Immer dann, wenn eine Kontrollflussentscheidung benötigt wird (zum Beispiel: `if`- oder `else`-Block ausführen), kann diese Information im Kontrollfluss-Log nachgeschlagen werden.

Where-Provenance.

Während die Anweisungen nacheinander interpretiert wer-

```

1 days = [(), (), (),
2         (), (), (), ()]
3 def dow(num, fmt, days):
4     if fmt: #fmt: True
5         pos = (num+()) % ()
6     else:
7         pos = num
8         res = days[pos] #pos: 5
9         return res
10 dow((), (), days) #returns: ()

```

Listing 3: Pseudocode aus Sicht der abstrakten Interpretation.

den, wird eine Variablenumgebung gepflegt und mit jeder interpretierten Anweisung gegebenenfalls aktualisiert. Die Umgebung beinhaltet alle derzeit sichtbaren Variablen zusammen mit ihren jeweiligen Abhängigkeiten von den Eingabedaten.

Der Aufbau dieser Umgebung ist ein inkrementeller Prozess. Jede Zuweisung einer Variablen, wie zum Beispiel in `a = b`, wird eine entsprechende Aktualisierung der Umgebung nach sich ziehen. In diesem Beispiel müssten alle Abhängigkeiten von `b` nach `a` kopiert werden.

Auf diese Weise wird die Umgebung ständig aktuell gehalten und referenziert in ihren Abhängigkeiten stets die Eingabedaten. Am Ende der Analyse braucht nur noch die gewünschte Variable, zum Beispiel `res`, in der Umgebung nachgeschlagen zu werden.

Why-Provenance.

Die Ausführung von Anweisungen in einem `if`- oder `else`-Rumpf sind abhängig vom zugehörigen Prädikat des `if/else`-Konstrukts. Diese Abhängigkeit modellieren wir als Why-Provenance.

In der Analyse wird dazu eine Menge an Abhängigkeiten gepflegt, die dem Kontrollfluss selbst zugeordnet ist. Bei Eintritt in den Rumpf einer `if`-Anweisung wird dem Kontrollfluss eine Abhängigkeit vom Prädikat dieser `if`-Anweisung zugeordnet. Allen Zuweisungen, die in diesem Rumpf ausgeführt werden, werden dann wiederum die Abhängigkeiten des Kontrollflusses in Form von Why-Provenance hinzugefügt.

Nachdem der `if`-Rumpf abgearbeitet ist, werden die Abhängigkeiten des Kontrollflusses wieder zurückgesetzt. Ob `if`- oder `else`-Rumpf ausgeführt werden, macht hier keinen Unterschied: in beiden Fällen gilt dasselbe Prädikat. Die Interpretation von Schleifen funktioniert analog.

Beispiel-Analyse.

In Listing 3 ist abgedruckt, wie sich das aus dem Instrumentierungsschritt bereits bekannte Programmfragment in Schritt ② der Provenance-Analyse darstellen würde. Alle vorkommenden atomaren Werte sind durch `()` ersetzt worden. Code in dieser Form wird nicht erzeugt, doch das Listing veranschaulicht, auf welcher Basis die abstrakte Interpretation arbeitet.

Anhand dieses Beispiels wird nun dargestellt, wie die Ableitung der Data Provenance funktioniert. Das Ergebnis dieser Analyse besteht gemäß der gewählten Granularität und Auflösung (siehe Abschnitt 4.2) darin, wie die Variable `res` von den Funktionsparametern abhängig ist.

Um die Analyse zu initialisieren, werden die in den Funk-


```

days: [0e, 1e, 2e, 3e, 4e, 5e, 6e]
num: 7e
fmt: 8e

```

Abbildung 6: Initialisierung

tionsparametern vorkommenden Werte durch künstliche *ids* repräsentiert. Abbildung 6 zeigt die im Beispiel gewählte Repräsentation, die aus ansteigenden natürlichen Zahlen besteht. Die Variable `days` ist eine Liste und enthält dementsprechend für jedes Listenelement einen anderen Repräsentanten. Das tiefgestellte *e* soll anzeigen, dass eine Where-Abhängigkeit besteht (= Abhängigkeit vom Wert an dieser Stelle). Why-Abhängigkeiten kommen später hinzu und werden entsprechend durch *y* gekennzeichnet.

Diese Repräsentanten ersetzen während der abstrakten Interpretation die tatsächlichen Werte. Am Ende der Analyse von `dow()` wird die Variable `res` nicht mit dem Wert "Fri" belegt sein, sondern mit einer Menge von Repräsentanten.

In Tabelle 1 sind die einzelnen Analyseschritte der abstrakten Interpretation aufgeführt. In der ersten Spalte steht die Zeilennummer des Python-Statements, das soeben interpretiert wurde. Die mittlere Spalte enthält die Abhängigkeiten des Kontrollflusses. Die letzte Spalte zeigt den Inhalt der Variablenumgebung. Um die Übersichtlichkeit zu verbessern, wird hier die Variable `days` nicht dargestellt.

In Zeile 3 wird die Funktion betreten. Hier findet die Initialisierung der Datenstrukturen statt, das heißt die `ids` 0..8 werden vergeben. `num`, `fmt` und `days` sind die Parameter der Funktion und werden der Umgebung hinzugefügt.

In Zeile 4 beginnt das `if/else`-Konstrukt. Hier werden zunächst die Abhängigkeiten des `if/else`-Prädikats `fmt` den Kontrollfluss-Abhängigkeiten hinzugefügt. Dabei findet eine Transformation von `8e` zu `8y` statt. Die Umgebung bleibt unverändert. Als dritter Punkt wird das Kontrollfluss-Log gelesen, um festzustellen, ob der `if`- oder `else`-Rumpf besucht werden soll. Das Log liefert ein `True` zurück, also wird eine Analyse des `if`-Rumpfs durchgeführt.

Zeile 5 enthält eine Zuweisung an die Variable `pos`. Dazu wird zunächst der Ausdruck `(num+()) % ()` analysiert. Die beiden `()` haben hier keine Abhängigkeiten, werden also einfach ignoriert. Die Abhängigkeiten von `num` werden kopiert: `7e`. Außerdem werden die Kontrollflussabhängigkeiten übernommen: `8y`.

Zuletzt findet in Zeile 8 ein Listenzugriff statt. Hier wird das `Indices`-Log bemüht, das den Index 5 enthält. An 5. Stelle von `days` befindet sich `5e` und wird nach `res` kopiert. Darüber hinaus hat der Index wie auch der Kontrollfluss eine steuernde Funktion. Deshalb wird der Inhalt von `pos` als Why-Provenance kopiert.

Das Ergebnis der Analyse für `res` ist: `[8y, 7y, 5e]`, also Where-Abhängigkeit von "Fri" und Why-Abhängigkeit von `fmt: True` und `num: 4`.

6. ZUSAMMENFASSUNG UND AUSBLICK

Unser hier vorgestellter Ansatz und seine konkrete Implementierung erweitert die bisherigen Grenzen der Provenance-Analyse von SQL. Window Functions und rekursive Queries sind Bestandteile aktueller DBMS-Implementierungen und können von unserem Prototypen analysiert werden.

Derzeit wird im Rahmen einer studentischen Arbeit eine zusätzliche Python-Implementierung der hier vorgestellten Provenance-Analyse entwickelt. Ihr Merkmal besteht darin, dass sie Bytecode direkt analysiert (statt den Python-AST).

Zeile	Abhängigkeiten	
	Kontrollfluss	Variablen
3 (<code>dow()</code>)		num: 7 _e fmt: 8 _e
4 (<code>if fmt:</code>)	8 _y	num: 7 _e fmt: 8 _e
5 (<code>pos=</code>)	8 _y	num: 7 _e fmt: 8 _e pos: 8 _y , 7 _e
8 (<code>res=</code>)		num: 7 _e fmt: 8 _e pos: 8 _y , 7 _e res: 8 _y , 7 _y , 5 _e

Tabelle 1: Ableitung der Provenance

Wir versprechen uns eine Performanceverbesserung von dieser neuen Implementierung.

Als weitere Implementierung ist LLVM geplant, um mittels [9] kompilierte Queries analysieren zu können.

Habitat [8] ist ein SQL-Debugger, der eingesetzt wird, um potentiell fehlerhafte Queries direkt auf SQL-Sprachebene zu untersuchen. Dazu wird die verdächtige Query von Habitat instrumentiert und vom RDBMS ausgeführt. Die instrumentierte Query *beobachtet* (= zeichnet auf), wie die potentiell fehlerhafte Ergebnisrelation berechnet wird und präsentiert dem Benutzer diese Beobachtungen. Bei großen Eingabetabellen besteht das Problem, dass man vielleicht nur an der Beobachtung der Berechnung eines einzelnen Ergebnistupels interessiert ist, aber auch tausende andere Tupel zusätzlich beobachtet. Um genau die für ein bestimmtes Ergebnistupel relevanten Eingabetupel herauszufinden, ist Data Provenance genau das richtige Werkzeug. Eine Kombination von diesen beiden Techniken wird von uns angestrebt.

7. REFERENCES

- [1] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for Aggregate Queries. In *Proc. PODS*, pages 153–164. ACM, 2011.
- [2] P. Buneman, S. Khanna, and W.-C. Tan. Why and Where: A Characterization of Data Provenance. In *Proc. ICDT*, pages 316–330. Springer, 2001.
- [3] J. Cheney. Program Slicing and Data Provenance. *IEEE Data Engineering Bulletin*, 30(4):22–28, 2007.
- [4] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4), 2007.
- [5] Y. Cui, J. Widom, and J. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. *ACM TODS*, 25(2), 2000.
- [6] B. Glavic and G. Alonso. Provenance for Nested Subqueries. In *Proc. EDBT*, pages 982–993. ACM, 2009.
- [7] B. Glavic and K. Dittrich. Data Provenance: A Categorization of Existing Approaches. In *BTW*, volume 7, pages 227–241. Citeseer, 2007.
- [8] T. Grust and J. Rittinger. Observing SQL Queries in their Natural Habitat. *ACM TODS*, 38(1), 2013.
- [9] T. Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. In *Proc. VLDB*, 2011.
- [10] M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, SE-10(4), 1984.

Large-scale Analysis of Event Data

Stefan Hagedorn
TU Ilmenau, Germany
stefan.hagedorn@tu-
ilmenau.de

Kai-Uwe Sattler
TU Ilmenau, Germany
kus@tu-ilmenau.de

Michael Gertz
Heidelberg University,
Germany
gertz@informatik.uni-
heidelberg.de

ABSTRACT

With the availability of numerous sources and the development of sophisticated text analysis and information retrieval techniques, more and more spatio-temporal data are extracted from texts such as news documents or social network data. Temporal and geographic information obtained this way often form some kind of event, describing when and where something happened. An important task in the context of business intelligence and document exploration applications is the correlation of events in terms of their temporal, geographic or even semantic properties. In this paper we discuss the tasks related to event data analysis, ranging from the extraction of events to determining events that are similar in terms of space and time by using skyline processing and clustering. We present a framework implemented in Apache Spark that provides operators supporting these tasks and thus allows to build analysis pipelines.

1. INTRODUCTION

Traditionally, research on querying and analyzing spatio-temporal data focuses on data obtained from sensors that record the evolution and movement of objects in 2- or 3-dimensional space over time. Typical examples of such data include trajectories of moving objects (e.g., [7]) and remotely-sensed data (e.g., [10]). Spatio-temporal data, however, do not only arise in the context of sensor data or, more generally, from observing objects with their spatial extent over time. In this paper, we consider *events* as an important type of spatio-temporal data that thus far have been neglected in the context of data analysis. Events play an important role in information retrieval and text analysis in general, most prominently in the task of topic detection and tracking (TDT) [2, 13]. An event is often described as “something that happens at some place at some time”, e.g., [24]. Thus, events inherently have a spatial and a temporal component.

A prominent source for event data are textual information from newsfeeds, websites, and social media such as blogs, tweets or social networks. Underlying the extraction

of such information about events are temporal taggers for the temporal component and geo-taggers for the spatial or geographic components [21]. Such taggers detect, extract, and normalize respective expressions in textual data and provide subsequent tools as the basis for further tasks such as document clustering or classification. For example, from the sentence “Obama visited Germany in April 2009”, a temporal tagger would detect the expression “April 2009” and normalize it to “2009-04”; similarly, a geo-tagger would extract the expression “Germany” and often associate further information with it. This might include the spatial extent in the form of a polygonal description or that Germany is part of Europe (using some concept hierarchy). Such a pair of temporal component and geographic component then forms an event. Given that today’s taggers become more and more sophisticated, large repositories of event information can be built from news articles, blog entries, social network postings, and medical records, to name but a few. The task we focus on in this paper is to find events that are correlated to a given event in terms of its time and place of occurrence. The result of such a query is a list of pointers to documents in which similar (correlated) events have been detected.

For such correlation tasks, one is facing several challenges ranging from extracting event data from documents, dealing with imprecise event specifications resulting from the extraction process, to exploiting different correlation techniques for finding similar events, to scalable processing for large datasets.

In this paper, we describe a framework for analyzing event data and detecting correlations between events. Based on a model for spatio-temporal events at different granularities we discuss corresponding distance measures. We present the analysis tasks and discuss how these tasks can be supported by a set of basic operators for extracting event data from text documents, preparing the data, and analyzing event correlations using top-k and skyline processing as well as clustering. We further describe how these operators are supported as transformation operators in Apache Spark and outline the support for explorative analyses.

2. EVENT MODEL

An important data analysis task is to find similar events, i.e. events that share the same context or have other values in common. In this paper, we consider the spatio-temporal aspect of events for determining similarities, i.e., we focus on the similarity of their respective location and/or time of occurrence.

For our analysis framework, we assume an event model in which information about events has been extracted from some document (see Sect. 4) and is represented by useful and necessary information like ID, description, origin, etc. as well as a temporal and a geographic component, describing the when and where. The expressions underlying these components are based on concept hierarchies for time and space, as shown in Figure 1.

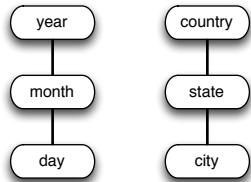


Figure 1: Concept hierarchies for temporal and geographic information

The temporal as well as the spatial expressions can be of different granularities. For temporal expressions we consider days to be of the finest and years of the coarsest granularity. Of course one could also extend this model with further granularity levels, such as weeks, hours, or minutes. In this paper, however, and for the sake of simplicity, we will only consider days, months, and years. We denote the corresponding domains as $T = \{T_{day}, T_{month}, T_{year}\}$. For example, “2001-05-20”, “2013-05”, and “1995” are all valid temporal expressions from these three different domains.

Analogously, geographic expressions are taken from the domains in $G = \{G_{city}, G_{state}, G_{country}\}$. We assume that with each expression a spatial object in the form of a single polygon (without holes) is associated. For example, the geographic expressions “Heidelberg” and “Germany” are both valid expressions of type G_{city} and $G_{country}$, respectively.

Definition. (Event) Given concept hierarchies T and G for temporal and geographic expressions, respectively. An event $e = \langle t, g \rangle$ consists of a temporal expression t with $t.type \in T$ and a geographic expression g with $g.type \in G$.

Examples of (imprecise) event specifications are (2013-09-02, Munich), (1955, Germany), or (2000-04, Bavaria). To account for these types of uncertainties, in our framework we make the following assumptions:

1. Temporal and geographic expressions of the finest granularity are certain.
2. Every temporal (resp. geographic) expression of type P' that refines a given temporal (resp. geographic) expression of type P , with P' being of finer granularity than P , is equally likely.

Distance Measures. To determine the similarity between events, a distance measure is needed that takes both the temporal and the geographic component of an event into account, both of which can be imprecise.

Precise events are those events for which both the location and temporal information is given as a fine-grained concept, i.e., they are defined on the city and day level. For such events, calculating the distance is trivial resulting in a scalar

value for time (e.g., distance in days) and location (e.g. distance in meters). Both values can be combined into a single (weighted) result. Although we cannot use the traditional distance functions for events that are imprecise in at least one component, we can specify new distance functions accordingly. However, the definition of such functions is beyond the scope of this paper and we will give only a brief overview below.

First, we convert the imprecise event data into intervals and regions. As imprecise temporal data is defined as a month or year (to be imprecise the day part is missing), we can create an interval of days that starts with the minimal possible day, i.e., the first day of the corresponding month or a year and ends with the maximal possible day, which is the last day of the month or year, respectively. Each subinterval is called a valid instance of this interval. As an example consider the imprecise temporal expression “2014-05” that is mapped to the (right-open) interval $i = [2014-05-01, 2014-05-30]$. Note that any subinterval in i is a valid instance of the temporal expression. Analogously, for imprecise geographic expression, i.e., expressions not at the city level, a polygon (or its minimum bounding box) is used. Then, a function that calculates the distance between such intervals and between polygons/boxes is needed. Such distance function can yield a scalar value, which may be the average distance between points in the respective intervals or polygons, or it can yield an interval, representing the minimum and maximum distance between the intervals/polygons.

The Hausdorff distance is a typical approach to calculate the distance between two intervals or polygons as a scalar value. Its main idea is to compute the maximum distance between any point in one interval to any other point of the second interval. For two temporal intervals t_1 and t_2 , the distance can be computed as

$$d_H(t_1, t_2) := \max\{\max_{i_1 \in t_1} d_h(i_1, t_2), \max_{i_2 \in t_2} d_h(i_2, t_1)\}$$

where the distance $d_h(i, t)$ between a day i and a temporal interval t defined as $d_h(i, t) := \min_{i' \in t} (|i - i'|)$.

For polygons/boxes, the Hausdorff distance can be defined as follows:

$$d_H(g_1, g_2) := \max_{x \in g_1} \min_{y \in g_2} \|x - y\|$$

This is the greatest distance from a point in g_1 to the closest point in g_2 . However, the calculation of this distance measure can become very expensive as the distance from every point in the first interval to every other point in the second interval has to be computed. For temporal intervals this is not a big problem, because when taking years as the most imprecise and days as most precise level, such an interval can have only 365 (or 366) points at most. However, for polygons the set of inner points is in principle infinite. On the one hand, one could use the cities that are located in the respective region. This approach may not lead to good results as many cities may be clustered around a large metropolis, whereas in rural regions only very few cities can be found. On the other hand, one could impose a raster on the respective regions using a fixed grid. Then, only the points of this grid will be used for calculations. Then, however, the definition of the grid size may be difficult since choosing a small grid step size may lead to many data points that will take part in the calculation and thus, will not improve the computation time. Choosing a large grid size will

result in only very few data points, which might again result in incorrect distance values. In order to use common distance functions, one could reduce a polygon to a single point that represents this polygon, e.g., the center point. However, these simplifications result in distances that may not necessarily reflect the real distances. Therefore, more sophisticated distance functions are needed.

3. ANALYSIS TASKS

Analysis of event data comprises several tasks. Depending on the sources of events (structured data, text documents) and the specific questions different tasks have to be combined in an analysis pipeline. In addition, such analyses are often interactive and explorative processes requiring a flexible combination of a set of powerful extraction and correlation operators. The whole process is depicted in Fig. 2. In the following, we describe a basic set of tasks that our framework supports by dedicated operators.

3.1 Event Information Extraction

Prior to any analysis task event information needs to be extracted from the text documents of a given corpus, such as Wikipedia or news articles. As an event is assumed to be composed of a temporal expression describing the “when” of an event and a geographic expression describing the “where” of an event, respective event components need to be determined in a given text and mapped to some standard format. The function of a temporal tagger is to determine such temporal information and to map each piece of information found to a temporal expression. One typically distinguishes between explicit, implicit and relative information temporal information. Explicit temporal information refers to a sequence of tokens that describe an exact date, month in a year or year, e.g., “April 1, 2015”. Implicit temporal information often refers to holidays or some named events, such as “Independence Day 2015” or “Summer Olympics 2012”. Finally, relative temporal information can only be determined using the context or document in which the token sequence appears. For example, in this paper, the string “last year” would refer to the year 2014. Popular temporal taggers such as HeidelTime [21] are able to detect such information and map them to a standard format, which is mostly based on the TIMEX3 annotation standard. For example, the above explicit temporal information “April 1, 2015” would be mapped to the temporal expression “2015-04-01”.

Similarly, for geographic information in documents, a geo-tagger is employed. Popular tools include GeoNames¹ or Yahoo! Placemaker². Mapping token sequences found in a text to some standardized format, however, is less trivial. For example, most taggers would map the string “Magdeburg” to a latitude/longitude pair rather than to some complex polygon description. Also, several geo-taggers also include some information about the granularity of the geographic expression based on concept hierarchies.

Once a temporal tagger and a geo-tagger have extracted and mapped temporal expressions from a given text, events are formed. In the most simple case, an event is a pair consisting of a temporal expression and a geographic expression found in close text proximity, typically at the sentence level.

¹<http://www.geonames.org/>

²<https://developer.yahoo.com/boss/geo/>

All events extracted from a document or document collection are then stored based on our event model described in Sect. 2. For each event detected, it describes the normalized temporal and geographic expression, the granularity of the expressions, and also some offset information (in what document and at what position each of the two components have been determined).

3.2 Correlation Analysis

Correlations on event data can be computed in different ways, depending on specific applications. In the following, we discuss the three operations for correlation computations.

Nearest Neighbor Queries.

A straightforward solution to find correlated, i.e., similar, events is to find the neighbors of a given reference event. Given a set of events \mathcal{E} , a reference event e_r and a distance function $dist$, the task is to find the set $kNN(e_r)$ of the k nearest events. In the case of our spatio-temporal event data this requires a single distance measure, which is usually defined using weights for the spatial and temporal distances. These weights are necessary, because we cannot directly combine the spatial distance with the temporal distance into one distance measure. However, with the weights we can adjust the impact of the spatial and temporal distance to the overall distance:

$$dist(e_1, e_2) = w_g \cdot dist_g(e_1, e_2) + w_t \cdot dist_t(e_1, e_2)$$

where $w_t, w_g \in [0, 1]$ and $w_t + w_g = 1$.

Skyline.

In contrast to the Nearest Neighbor search, Skyline queries do not need weights for the spatial and temporal distance, which are often difficult to determine. Adapted to our scenario, the notion of the Skyline algorithm is to find those events in \mathcal{E} that “match” a query event $q = \langle t_q, g_q \rangle$ best. Since we consider two dimension for events, time and space, it is thus intuitive to employ a skyline-based approach as there might be events that match t_q well but not g_q , and vice versa. A core concept of skylines is the dominance relationship. The skyline S_q consists of all events that are not dominated by any other event in \mathcal{E} with respect to q :

$$S_q = \{e_i | e_i \in \mathcal{E} \wedge \neg \exists e_j \in \mathcal{E} : e_j \neq e_i \wedge e_j \succ_q e_i\}$$

where \succ_q denotes a dominance relationship with $e_j \succ_q e_i$ meaning that e_j is “in closer proximity to q ” than e_i . Because the dominance of an event with respect to another event is determined based on their respective distances to q , the distance function outlined before comes into play.

Clustering.

Other than in the two approaches mentioned before, for clustering, a definition of a reference event is not needed. It is typically understood as the task of grouping objects based on the principle of maximizing the intra-class similarity and minimizing the inter-class similarity. However, there is a rich diversity in specific definitions of clustering and many different techniques exist [1].

The clusters can be formed using distance values between other events. Thus, events that belong to the same cluster can be considered to be correlated as they occur around the same time and place.

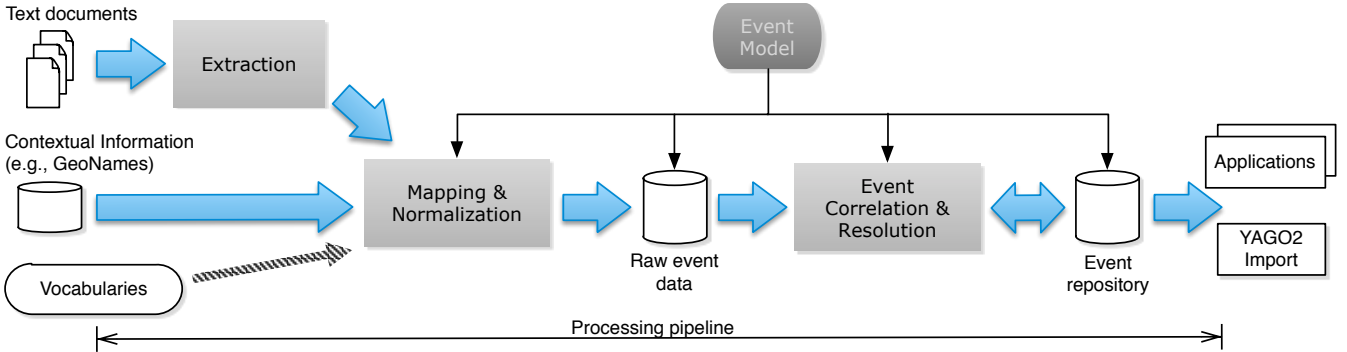


Figure 2: Overview of event analysis pipeline.

4. PROCESSING PIPELINE

After the events have been extracted from the text document using the approach as outline in Section 2, the events are fed into the correlation pipeline. This pipeline is implemented using the Apache Spark³ platform. However, it can be easily ported to other platforms like Apache Flink⁴ or even Google’s new DataFlow SDK⁵ if they provide a Scala-based API. We chose Spark over other MapReduce based approaches, e.g., Pig, because it provides a very efficient data structure called resilient distributed datasets (RDD). RDDs are immutable, in-memory partitioned collections that can be distributed among the cluster nodes. With the help of such a data structure, the performance of the computations is significantly faster than in MapReduce programs that materialize their intermediate results to disk. Furthermore, Spark allows to implement iterative models that are needed for our computations as well as for programs following the MapReduce paradigm.

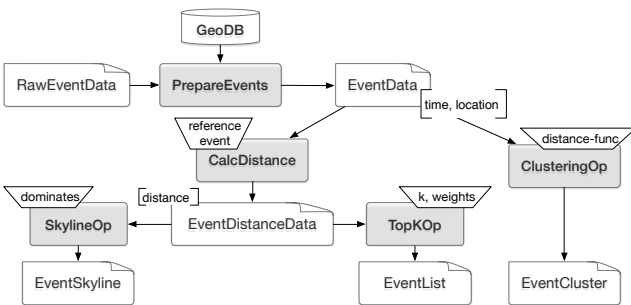


Figure 3: Framework overview

The Spark operators represent transformations on these RDD. Fig. 3 gives an overview of the correlation pipeline, where the tasks of the operators is described below:

PrepareEvents: This operator transforms a set of raw (textual) event data into a set of event records $\langle t, q \rangle$ conforming to our framework. This means that textual temporal and geographic properties are normalized into numerical values, i.e., date/time values and points or polygons for the spatial descriptions such as names of cities or locations.

CalcDistance: This implements a transformation operator for calculating the geographic and temporal distance $dist$ of each event of an RDD to a given reference event.

TopKOp: This operator computes the k nearest neighbors as a top- k list of events from an input RDD produced by **CalcDistance**. Parameters to this operator are k as well as the weights for the geographic (w_g) and temporal (w_t) distance.

SkylineOp: This operator computes the skyline of event records from an RDD produced by **CalcDistance**. Our implementation adopts the grid-based approach using bitsets described in [14] for the Spark platform. The dominance relation can be passed as parameter to the operator.

ClusteringOp: Finding groups of correlated events is realized by the **ClusteringOp** operator implementing a parallel variant of DBSCAN [9] for spatio-temporal data. Parameters are the standard clustering parameters ϵ and $MinPts$ as well as a global distance function taking both geographic and temporal distances into account.

5. TOWARDS AN INTERACTIVE EXPLORATION OF EVENT CORRELATIONS

We will use the above processing pipeline to build an event repository that makes the extracted event data publicly available as Open Data. As an underlying data structure we are going to use the Linked Data principle, which allows for a flexible schema for different types of events and also makes the information contained in the event itself machine readable. Furthermore, one can easily add links between events that have been identified to be correlated and thus make these correlations also available for querying and processing. Adding links to other datasets like YAGO2 or DBpedia will enrich our event repository with even more useful and broader information that can be used for data analysis and exploration.

The event repository will be free to download and will also be made queryable via web services. Via an API users can run Spark programs (using operators introduced in Sect. 3) or SPARQL queries.

Next to the event repository we plan to develop a data exploration platform that will allow users to interactively analyze the data. Different from traditional business intelligence frameworks that provide only a predefined set of tools

³<http://spark.apache.org>

⁴<http://flink.apache.org>

⁵<https://cloud.google.com/dataflow/>

```
%pig
studs = LOAD '/data/students.csv' using PigStorage(',') AS (id, name, major, year, dept);
tests = LOAD '/data/tests.csv' using PigStorage(',') AS (id, subject, student_id, grade);
joined = JOIN studs BY id, tests BY student_id;
grouped = GROUP joined BY dept;
avg = FOREACH grouped GENERATE group, AVG(grade) AS Average;
sorted = ORDER avg BY Average ASC;
DUMP sorted;|
```

FINISHED



SETTINGS

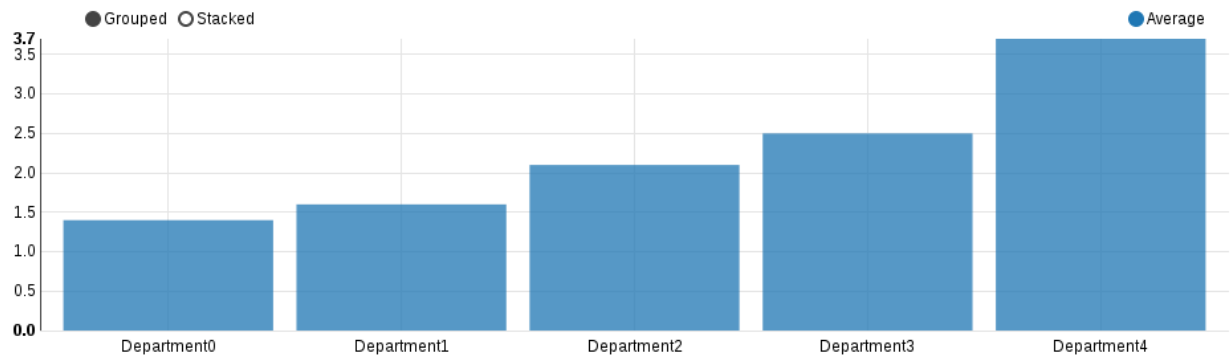


Figure 4: Screenshot of Zeppelin after executing a Pig script. The results are visualized as bar chart.

for reporting and visualization, our planned platform allows very different interaction paradigms. We integrate the idea of notebooks that was inspired by Mathematica’s interactive documents and IPython’s (and Jupyter’s) notebooks. With the help of such notebooks users can create their own programs that fit their needs and run them in a managed cluster environment without the need to set up any hardware or other software before. The idea of web-based notebooks allows users to organize text, scripts, and plots on a single webpage so that all information for data analytics can be kept in one place.

For our data exploration tool we chose the Apache Zeppelin⁶, because it already has support for Spark programs and also provides the possibility to visualize script results. Content in the format of CSV, HTML, or images can directly be visualized as tables, bar charts, line graphs, and scatter plots. A notebook in Zeppelin is organized in paragraphs where each of them can contain and execute a script of a different (supported) language. On execution, the script content of a paragraph is sent to the server, which will forward it to the appropriate interpreter. The interpreter is chosen by a magic keyword given by the user at the beginning of the script, e.g. `%spark`. When the execution has finished, the interpreter will return the results to the server, which in turn will publish them on the website. Figure 4 shows a screenshot of a notebook in Zeppelin that executed a Pig [16] script and shows the results of that script in a bar chart.

Zeppelin currently supports shell scripts and Spark programs written in Scala or Python out of the box and can

also make use of Spark modules like SparkSQL and Spark Streaming. Although the Spark support lets users create efficient data analytics programs, it may be hard for a non-programmer to create such programs.

We argue that a declarative approach will be easier to use for users that are not familiar with the Spark API and the concepts of their RDDs. For this reason, we integrate a new interpreter that allows to run Pig scripts. Since our event repository uses Linked Data, we do not use the conventional Pig interpreter, but our extended Pig version that allows to use SPARQL-like features, such as BGP, in Pig scripts [11]. To do so, we enhanced Pig’s `FILTER` operator. This allows the user to easily define queries on the Linked Data sets with a more powerful language than SPARQL. Though, the triple structure of the Linked Data concept is not very well suited for a tuple-based execution environment as it requires a lot of (self-)joins to reconstruct all details of an entity from the triples. To solve this issue, in [11] we introduced a new triple bag format (based on the concept introduced in [12]) that keeps the flexibility of triples with the convenience of tuples. We also showed that this format can lead to significant performance gains.

However, since Pig programs are compiled into MapReduce programs, the execution of these programs will take longer than for Spark programs. Thus, to combine the advantages of both approaches, in our ongoing work we use the Pig Latin language and compile it into Spark programs. This will allow users to write intuitive data flow scripts without the need to know a specific API and characteristics of the underlying platform and to get very efficient programs that will execute faster than MapReduce programs.

⁶<https://zeppelin.incubator.apache.org>

6. RELATED WORK

For our event correlation framework, we employ concepts developed in different areas of information extraction, event similarity especially for spatio-temporal similarity, as well as skylines and clustering algorithms for distributed environments.

To extract the spatial and temporal information from textual data, we use concepts that were described, e.g., in [22, 23]. In addition to these works, the notions of event similarity and relationships between events has also been studied in the context of social media, e.g., [3, 17, 18].

Our correlation operators mainly focus on Skylines and clustering. Among the numerous techniques that build on the concept of skyline queries [5], there also has been some work that study skylines for geographic and uncertain or probabilistic data. These include spatial skyline queries [19, 20] where no temporal aspects are considered for data points.

To compute skylines for large datasets, new algorithms have been proposed that allow the computation in MapReduce environments. Here, an important challenge is to partition the data in a way, that the partitioning itself is efficient and data is distributed to all nodes equally to ensure that all nodes get approx. the same workload to avoid one node having to do all the work while the others are idle. In [14] an approach using a grid based partitioning is introduced, and [6] shows an angular based partitioning approach.

For clustering, density based algorithms like DBSCAN [9] are chosen if the number of clusters is not known apriori. For dealing with spatio-temporal data an extension to DBSCAN has been proposed in [4], that uses two ϵ parameters (one for spatial and one for temporal distance). To run the clustering algorithms in a distributed environment for MapReduce, we rely on concepts developed in [15, 8].

7. SUMMARY

In this paper we presented our approach for an event correlation framework based on Apache Spark. The framework provides operators for importing data as well as for clustering, skylines and k nearest neighbor queries. We chose Apache Zeppelin for our exploration tool to allow an incremental creation of scripts and an immediate visualization of results.

Acknowledgements

This work was funded by the German Research Foundation (DFG) under grant no. SA782/22.

8. REFERENCES

- [1] C. C. Aggarwal and C. K. Reddy. *Data clustering: algorithms and applications*. CRC Press, 2013.
- [2] J. Allan, editor. *Topic detection and tracking: event-based information organization*. Kluwer Academic Publishers, 2002.
- [3] H. Becker, M. Naaman, and L. Gravano. Learning similarity metrics for event identification in social media. In *WSDM*, pages 291–300, 2010.
- [4] D. Birant and A. Kut. ST-DBSCAN: An algorithm for clustering spatial-temporal data. *Data & Knowledge Engineering*, 2007.
- [5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [6] L. Chen, K. Hwang, and J. Wu. MapReduce Skyline Query Processing with a New Angular Partitioning Approach. In *IPDPSW*, May 2012.
- [7] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, 2005.
- [8] B.-R. Dai and I.-C. Lin. Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition. In *CLOUD*, June 2012.
- [9] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *KDD*, 1996.
- [10] A. Ganguly, O. Omitaomu, Y. Fang, S. Khan, and B. Bhaduri. Knowledge discovery from sensor data for scientific applications. In *Learning from Data Streams*. 2007.
- [11] S. Hagedorn, K. Hose, and K.-U. Sattler. Sparqling pig - processing linked data with pig latin. In *BTW*, March 2015.
- [12] H. Kim, P. Ravindra, and K. Anyanwu. From SPARQL to MapReduce: The Journey Using a Nested TripleGroup Algebra. *PVLDB*, 4(12):1426–1429, 2011.
- [13] J. Makkonen, H. Ahonen-Myka, and M. Salmenkivi. Topic detection and tracking with spatio-temporal evidence. In *Advances in Information Retrieval*, volume 2633, pages 251–265. 2003.
- [14] K. Mullesgaard, J. L. Pedersen, H. Lu, and Y. Zhou. Efficient Skyline Computation in MapReduce. In *EDBT*, 2014.
- [15] M. Noticewala and D. Vaghela. MR-IDBSCAN: Efficient Parallel Incremental DBSCAN Algorithm using MapReduce. *Intl J Computer Applications*, 93(4):13–17, 2014.
- [16] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, 2008.
- [17] W. Ribarsky, D. Skau, X. Wang, W. Dou, and M. X. Zhou. Leadline: Interactive visual analysis of text data through event identification and exploration. *VAST*, 0:93–102, 2012.
- [18] A. D. Sarma, A. Jain, and C. Yu. Dynamic relationship and event discovery. In *WSDM*, pages 207–216, 2011.
- [19] M. Sharifzadeh, C. Shahabi, and L. Kazemi. Processing spatial skyline queries in both vector spaces and spatial network databases. *TODS*, 2009.
- [20] W. Son, M.-W. Lee, H.-K. Ahn, and S. won Hwang. Spatial skyline queries: An efficient geometric algorithm. In *SSTD*, pages 247–264, 2009.
- [21] J. Strötgen and M. Gertz. Multilingual and cross-domain temporal tagging. *Language Resources and Evaluation*, 47(2):269–298, 2013.
- [22] J. Strötgen and M. Gertz. Proximity2-aware ranking for textual, temporal, and geographic queries. In *CIKM*, pages 739–744, 2013.
- [23] J. Strötgen, M. Gertz, and C. Junghans. An event-centric model for multilingual document similarity. In *SIGIR*, pages 953–962, 2011.
- [24] Y. Yang, T. Pierce, and J. Carbonell. A study of retrospective and on-line event detection. In *SIGIR*, pages 28–36, 1998.

Flexible Online-Rec recommender-Systeme durch die Integration in ein Datenstrommanagementsystem

Cornelius A. Ludmann
Universität Oldenburg
Escherweg 2
26121 Oldenburg, Germany
cornelius.ludmann@uni-oldenburg.de

Marco Grawunder
Universität Oldenburg
Escherweg 2
26121 Oldenburg, Germany
marco.grawunder@uni-oldenburg.de

H.-Jürgen Appelrath
Universität Oldenburg
Escherweg 2
26121 Oldenburg, Germany
appelrath@uni-oldenburg.de

ZUSAMMENFASSUNG

In dieser Arbeit wird eine flexible, erweiterbare und domänenunabhängige Integration von Online-RecSys-Funktionen in ein Datenstrommanagementsystem (DSMS) vorgestellt. Dazu werden neue logische Operatoren eingeführt, mit der Nutzer eines DSMS auf einer abstrakten Ebene RecSys-Funktionen nutzen können. Des Weiteren wird eine beispielhafte Realisierung durch physische Operatoren vorgestellt, wie sie aus den logischen Operatoren durch Transformationsregeln erzeugt werden kann. Durch dieses Konzept können Benutzer ein RecSys mit Hilfe einer Anfragesprache auf die domänenspezifischen Bedürfnisse anpassen und mit anderen Funktionen eines DSMS kombinieren. Des Weiteren bringt ein DSMS einige Eigenschaften (z. B. Anfrageplanoptimierung, Fragmentierung, Scheduling etc.) mit, von denen ein Online-RecSys profitieren kann. Die Flexibilität eines DSMS ermöglicht den Vergleich und die Evaluation verschiedener RecSys-Algorithmen durch den Benutzer.

Keywords

recommender system, collaborative filtering, data stream management system, stream processing

1. EINLEITUNG

Recommender-Systeme (RecSys) haben das Ziel, das Interesse eines Benutzers an einem bestimmten Objekt zu schätzen, um dem Benutzer aus einer großen Menge an Objekten die subjektiv interessantesten Objekte zu empfehlen. Eine gängige Methode ist das modellbasierte, kollaborative Filtern (CF), bei dem aufgrund von bekannten Objektbewertungen verschiedener Benutzer (z. B. Produktbewertungen) ein Modell angelernt wird, mit dem geschätzt werden kann, wie ein Benutzer unbekannte Objekte bewerten würde. Durch die Entwicklung von Online-Algorithmen für das CF können neue Bewertungen von Benutzern in die Modelle integriert werden, ohne dass das Modell von Grund

auf neu gelernt werden muss. Das ermöglicht eine neue Betrachtung des RecSys-Problems: Während bisherige Methoden das RecSys-Problem als periodisches Anlernen eines Modells basierend auf einen statischen und endlichen Datensatz betrachten, kann durch Online-Algorithmen das Problem als Verarbeitung von Bewertungs-Ereignissen eines kontinuierlichen und potenziell unendlichen Datenstroms betrachtet werden. Das hat den Vorteil, dass durch die sofortige Integration neuer Bewertungsinformationen nicht nur das grundsätzliche Interesse, sondern auch das aktuelle Interesse des Benutzers berücksichtigt werden kann.

Zur Konzeption eines datenstrombasierten RecSys schlagen wir vor, auf ein anwendungsunabhängiges und generisches Datenstrommanagementsystem (DSMS) aufzubauen. Die Eingabedaten werden als kontinuierlich auftretende, zeitannotierte Ereignisse eines potenziell unendlichen Datenstroms betrachtet. Das DSMS berechnet mit Hilfe von Datenstrom-Operatoren und Anfrageplänen kontinuierlich ein RecSys-Modell. Dieser Ansatz hat folgende Vorteile:

(1) Die Modellierung der Daten als Datenströme kommt dem realen Einsatz eines RecSys näher als die Verwendung von statischen Datensätzen.

(2) Ein DSMS bringt als Grundlage für ein RecSys bereits Konzepte und Operatoren zur effizienten Verarbeitung von Datenströmen mit temporal kohärenten Ereignissen mit.

(3) In einem DSMS kann ein RecSys in logische Teile zerlegt werden, welche jeweils durch einen Operator repräsentiert werden. Das ermöglicht eine flexible und auf die konkrete Anwendung angepasste Komposition von RecSys-Operatoren mit Standardoperatoren, z. B. für die Datenvor- bzw. -nachbearbeitung, Kontextmodellierung etc.

(4) Eine Aufgabe, zum Beispiel das Modelllernen, kann durch austauschbare Operatoren mit gleicher Semantik aber unterschiedlicher Implementierung realisiert werden. Das ermöglicht den Vergleich sowie den bedarfsgerechten Einsatz von verschiedenen Lernalgorithmen.

Ziel der Arbeit ist die Einführung eines generischen Konzepts zur Integration von RecSys-Funktionen in ein DSMS. Dazu führen wir im folgenden Abschnitt zunächst die Grundlagen ein und stellen in Abschnitt 3 verwandte Arbeiten vor. Danach beschreiben wir die logischen Operatoren für die Umsetzung eines RecSys (Abschnitt 4) und stellen eine Transformation in physische Operatoren vor (Abschnitt 5). In Abschnitt 6 diskutieren wir anschließend einige Entscheidungen unseres Konzepts und stellen Alternativen vor. Zum Schluss präsentieren wir in Abschnitt 7 unsere prototypische Implementierung in ein Open-Source-DSMS und zeigen die

Machbarkeit durch eine Evaluation, bevor wir mit einer Zusammenfassung und einem Ausblick die Arbeit beenden.

2. GRUNDLAGEN

Ein **RecSys** hat die Aufgabe einem aktiven Benutzer $u' \in U$ eine Menge an Objekten aus I (Menge aller Objekte) zu empfehlen, für die dieser sich interessiert (Empfehlungsmenge). Dazu schätzt das RecSys für jedes zur Empfehlung infrage kommende Objekt eine Bewertung $\hat{r} \in R$, welche das Interesse des Benutzers an dem Objekt quantifiziert. Die Menge der infrage kommenden Objekte wird als Menge der Empfehlungskandidaten bezeichnet. Die Empfehlungsmenge wird durch die K Objekte der Empfehlungskandidaten gebildet, die die höchsten Bewertungen haben (Top- K -Menge). Um die Bewertung eines Objekts für einen Benutzer bestimmen zu können, ermittelt das RecSys (z. B. mit der Matrix-Faktorisierung) eine Annäherung f_R an eine wahre aber unbekanntere Bewertungsfunktion $f_R : U \times I \rightarrow R$.

Seit den 1970er/1980er Jahren sind relationale *Datenbankmanagementsysteme* (DBMS) die bedeutendste Technik, Daten dauerhaft zu speichern. Ein DBMS abstrahiert von der Komplexität der Datenspeicherung und sorgt für eine effiziente Verarbeitung von komplexen Anfragen, um Daten zu speichern, zu lesen, zu aktualisieren und zu löschen. DBMS sind allerdings nicht mit dem Ziel entwickelt worden, kontinuierlich erzeugte Daten zu verarbeiten. Diese Lücke schließen **DSMS**, die auf die fortlaufende Verarbeitung von Datenströmen ausgelegt sind.

Während ein DBMS wahlfreien Zugriff auf gespeicherte Daten hat, erhält ein DSMS die Daten von einem kontinuierlichen Datenstrom (potenziell unendliche Sequenz von Datenstromelementen). Die Datenstromelemente werden von einer aktiven Quelle erzeugt und an das DSMS übertragen. Ein DSMS hat keine Kontrolle über die Reihenfolge, in der die Datenstromelemente von der Quelle gesendet werden – weder innerhalb eines Datenstroms, noch zwischen verschiedenen Datenströmen. Sobald ein Element verarbeitet ist, wird es verworfen oder archiviert. Das DSMS kann nicht erneut auf bereits verarbeitete Elemente zugreifen, es sei denn, sie werden explizit im Arbeitsspeicher gehalten (one-pass paradigm). Dies kann allerdings nur für einen begrenzten Teil der Datenstromelemente geschehen, da der Speicher im Gegensatz zum Datenstrom in der Größe begrenzt ist [4].

Die Verarbeitung der Daten in einem DBMS erfolgt üblicherweise mit Hilfe von Einmal-Anfragen. Diese werden einmalig auf aktuellem Datenbestand ausgeführt und der anfragende DBMS-Benutzer erhält einmalig eine Antwort. Bei der Verarbeitung von kontinuierlichen Datenströmen stellt der Benutzer eine kontinuierliche Anfrage. Diese wird einmalig dem DSMS übergeben und produziert kontinuierlich Ergebnisse [4]. Zur Verarbeitung von relationalen Datenströmen kann ein DSMS eine datenstrombasierte Variante der relationalen Algebra einsetzen. Eine kontinuierliche Anfrage wird, wie bei Anfragen in DBMSs, in einer Anfragesprache formuliert. Beispiele für DSMS-Anfragesprachen sind die SQL-ähnliche Sprache CQL [3] und die PQL [2].

Eine Anfrage wird von einem DSMS in einen Anfrageplan übersetzt. Dieser besteht aus Operatoren, die durch Warteschlangen miteinander verbunden sind. Ein Anfrageplan kann als gerichteter Graph interpretiert werden, dessen Knoten die Operatoren und Kanten die Warteschlangen darstellen. Die Ausführung der Operatoren koordiniert ein Scheduler [4]. Zu einer Anfrage wird ein logischer Anfra-

geplan, bestehend aus logischen Operatoren, erzeugt. Auf einem logischen Anfrageplan können Optimierungen ausgeführt werden (z. B. Veränderung der Operatorreihenfolge), ehe er in einen physischen Anfrageplan mit physischen Operatoren überführt wird. Physische Operatoren enthalten konkrete Implementierungen für die Verarbeitung der Daten. Das DSMS wählt zu jedem logischen Operator ein oder mehrere physische Operatoren, die für die Operation am geeignetsten sind.

3. VERWANDTE ARBEITEN

Verschiedene Veröffentlichungen über *inkrementelle* oder *online* Algorithmen für CF (z. B. [10, 11]) zeigen wie neue Lerndaten in CF-Modelle integriert werden können, ohne dass das Modell komplett neu gelernt werden muss. Diese Algorithmen bieten die Basis für eine datenstrombasierte Verarbeitung von Bewertungsdaten für RecSys. Die Arbeit von Diaz-Aviles et al. [7] stellt eine Methode zur datenstrombasierten Verarbeitung von Bewertungsdaten zum Lernen eines Matrix-Faktorisierungs-Modells vor. Dazu wird das Modell mit Daten aus einem Reservoir aktualisiert, welches eine Stichprobe des Datenstroms vorhält. Im Gegensatz zu diesen Arbeiten stellen wir keinen neuen Algorithmus vor, sondern setzen den Fokus auf die Komposition eines flexiblen RecSys mithilfe von Datenstromoperatoren, in denen diese Algorithmen als Operatoren integriert werden können.

Einige Veröffentlichungen nutzen Frameworks zur Implementierung eines RecSys, die bei der Entwicklung von datenstrombasierten Systemen unterstützen. Zum Beispiel setzt Ali et al. [1] *Apache Storm* ein, um einen CF-Algorithmus zu realisieren. Das datenstrombasierte Data-Mining-Framework *Massive Online Analysis* (MOA) [5] ermöglicht die Evaluation unter anderem von datenstrombasierten RecSys-Algorithmen. Im Gegensatz zu unserem Konzept bieten diese Arbeiten keinen anwendungsunabhängigen, erweiterbaren und flexiblen Ansatz, der auf die Komposition von Datenstromoperatoren setzt. Unser Ansatz kann außerdem von diversen DSMS-Funktionen profitieren.

Mit *StreamRec* stellen Chandramouli et al. [6] einen Ansatz zur Nutzung eines DSMS für ein RecSys vor. Dort wird das DSMS *Microsoft StreamInsight* genutzt. Im Gegensatz zu unserem Konzept werden in *StreamRec* ausschließlich Basisoperatoren eines DSMS eingesetzt und es ist auf die Berechnung von Objektähnlichkeiten zur Bestimmung der Empfehlungsmenge beschränkt. Unser Konzept verfolgt einen generischeren Ansatz, der auch die Integration von modellbasierten Lernalgorithmen erlaubt und durch logische RecSys-Operatoren eine logische Abstraktionsebene für den Nutzer des DSMS bietet.

4. LOGISCHE OPERATORSICHT

Betrachtet man die Ein- und Ausgabedaten eines RecSys, so kann man diese als folgende Datenströme auffassen: Erstens überträgt die Benutzeranwendung Benutzer-Objekt-Bewertungs-Tripel (u, i, r) für jede Bewertung, die ein Benutzer vornimmt. Zweitens werden von einem Benutzer u Empfehlungen angefordert (z. B. dann, wenn ein Benutzer die Empfehlungsseite der Benutzeranwendung öffnet; im folgenden Empfehlungsanforderungen – engl. Request for Recommendations, RfR – genannt). Drittens sendet das RecSys eine Empfehlungsmenge zurück an die Benutzeranwendung. Des Weiteren können die Eingabedaten

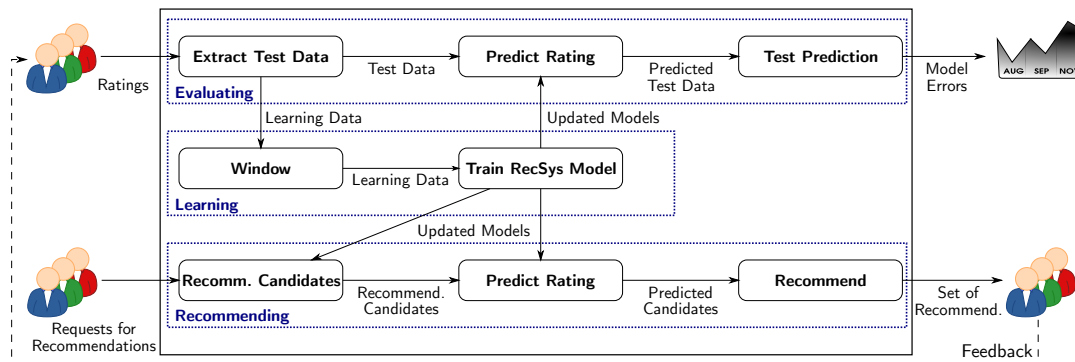


Abbildung 1: Logische Sicht auf die Operatoren eines DSMS-basierten RecSys

um ein Benutzerfeedback ergänzt werden. Im folgenden erwarten wir das Benutzerfeedback in der selben Form wie neue Bewertungen (u-i-r-Tripel). Möchte man das RecSys kontinuierlich evaluieren, so erhält man einen zusätzlichen Ausgabedatenstrom mit Fehlerwerten, der den Modellfehler angibt.

Zwischen den Ein- und Ausgabedatenströmen verarbeiten Datenstromoperatoren einer oder mehrerer Anfragen die Daten. Um ein RecSys mit Hilfe eines DSMS zu realisieren, muss mit Hilfe einer Anfrage aus den Eingabedatenströmen (u-i-r-Tripel und Empfehlungsanforderungen) als Antwort ein Datenstrom an Empfehlungsmengen erzeugt werden, der für jede Empfehlungsanforderung eine aktuelle und personalisierte Liste an Empfehlungen enthält.

Die Anfrage für ein RecSys haben wir in logische Operatoren zerlegt, die jeweils einen Teil der RecSys-Funktionalität übernehmen. Diese Operatoren bilden eine Abstraktionsebene, sodass der DSMS-Benutzer diese in der Anfrage nutzen kann, ohne die genaue Umsetzung durch physische Operatoren kennen zu müssen. Wo dies möglich ist, werden diese Operatoren bei der Transformation zu physischen Operatoren durch vorhandene Operatoren ersetzt.

Abbildung 1 zeigt eine Übersicht über die logischen Operatoren und wie diese in einer Beispielanfrage zusammenhängen. Auf der linken Seite sehen wir die Eingabedatenströme und auf der rechten Seite die Ausgabedatenströme. Die dazwischenliegenden Operatoren können in drei Kategorien eingeteilt werden: Operatoren zum Lernen des RecSys-Modells (mittig), zum Empfehlen von Objekten (unten) und zum Evaluieren (oben).

Die Anfrage besteht aus folgenden logischen Operatoren: **Window:** Ein zeitbasiertes Fenster (umgesetzt durch einen WINDOW-Operator) ermöglicht, die Gültigkeit der Bewertungsdaten zu beschränken. Das kann sinnvoll sein, um ein Modell anzulernen, welches sich auf die neuesten Daten beschränkt und ältere Daten verwirft (z. B. um Concept Drifts zu berücksichtigen). Zusätzlich beschränkt es die Menge der Bewertungsdaten, die im Speicher gehalten werden müssen. Sollen alle Daten für immer behalten werden, so kann dieser Operator entfernt werden.

Train RecSys Model: Dieser Operator lernt aus den Bewertungsdaten ein Modell an. Wenn neue Daten hinzugefügt werden gibt es ein aktualisiertes Modell aus. Das Modell ist für eine bestimmte Zeitspanne gültig, sodass zu jedem Zeitpunkt genau ein Modell für die Vorhersage der Bewertung zuständig ist.

Recomm. Candidates: Für jede Empfehlungsanforderung bestimmt dieser Operator die Empfehlungskandidaten. Diese sind in der Regel alle Objekte, die von dem anfordernden Benutzer noch nicht bewertet wurden.

Predict Rating: Dieser Operator wird sowohl für die Bestimmung der Empfehlungsmenge als auch für die Evaluation genutzt. Er schätzt die Bewertung des Benutzers für jeden Empfehlungskandidaten bzw. für jedes Testtupel ab. Mit diesem Operator wird sichergestellt, dass genau das Modell genutzt wird, welches zum gleichen Zeitpunkt wie die Empfehlungsanforderung bzw. das Testtupel gültig ist. Das stellt eine deterministische Verarbeitung der Daten sicher, was insbesondere für die Evaluation von Vorteil ist.

Recommend: Aus den bewerteten Empfehlungskandidaten werden die Objekte ausgewählt, die dem Benutzer empfohlen werden sollen. Das sind in der Regel die K Objekte mit den höchsten, geschätzten Bewertungen (Top- K -Menge). Eine weitere Möglichkeit besteht darin, nur Elemente mit einer minimalen Bewertung zu berücksichtigen.

Extract Test Data: Um das RecSys zu evaluieren, gibt dieser Operator neben den Lerndaten auch Testdaten aus. Eine mögliche Implementierung filtert 10% der Bewertungsdaten als Testdaten aus und leitet die restlichen Daten als Lerndaten an den Modellerner weiter.

Test Prediction: Dieser Operator implementiert eine Evaluationsmetrik, z. B. Root Mean Square Error (RMSE). Dabei vergleicht er die wahren Bewertungen aus den Testdaten mit den geschätzten Bewertungen zu den Testdaten oder die wahre Position in einem Ranking mit der Position aufgrund der geschätzten Bewertungen. Der daraus resultierende Modellfehler wird zu einem gleitenden Durchschnittswert oder einem gesamten Durchschnittswert aggregiert.

5. PHYSISCHE OPERATORSICHT

Die logischen Operatoren werden durch Transformationsregeln in physische Operatoren überführt. Abbildung 2 zeigt ein Beispiel für einen physischen Anfrageplan, der aus einer Anfrage generiert wird, der die logischen Operatoren nutzt. Die linke Spalte enthält die Operatoren für die Evaluation (entspricht dem oberen Teil von Abbildung 1), die mittlere Spalte die Operatoren für das Lernen des Modells (mittlerer Teil von Abbildung 1) und die rechte Spalte die Operatoren für die Berechnung der Empfehlungsmenge (unterer Teil von Abbildung 1). Dabei sind die Operatoren, die zu *einem* logischen Operator aus Abbildung 1 gehören, durch ein gestricheltes Kästchen zusammengefasst. Wo es möglich ist,

(ohne das Modell \hat{f} , welches nach der Berechnung von \hat{r} im folgenden Verlauf nicht mehr benötigt wird). Für die Evaluation zählt zu den weiteren Attributen des Eingabetupels insbesondere die wahre Bewertung r , die auch im Ausgabebetupel enthalten ist. Somit gibt der PREDICT-RATING-Operator für die Evaluation ein Tupel (u, i, r, \hat{r}) aus.

Im nachfolgenden Verlauf wird nun ein Fehlerwert berechnet. In unserer Beispielanfrage wird dazu ein gleitender RMSE berechnet. Dazu berechnet zuerst ein MAP-Operator den quadratischen Fehler $se = (r - \hat{r})^2$ der Schätzung. Anschließend wird mit einem TIME-WINDOW-Operator die gleitende Zeitspanne festgelegt, über die der Fehlerwert aggregiert werden soll (z. B. die letzten 24 Stunden). Der nachfolgende AGGREGATE-Operator berechnet das arithmetische Mittel der quadratischen Fehler, die sich in einem Aggregationsfenster befinden (z. B. alle Werte der letzten 24 Stunden). Abschließend berechnet der letzte MAP-Operator die Quadratwurzel aus dem aggregierten Fehlerwert, welches dem RMSE über das Aggregationsfenster entspricht. Diese Werte werden an den Ausgabedatenstrom übergeben.

Für die Evaluation ist es wichtig, dass zum Testen des Modells das zu testende Datum nicht im Modell enthalten ist. Eine einfache Möglichkeit dies sicherzustellen ist die Trennung von Test- und Lerndaten. Dies kann durch einen ROUTE-Operator realisiert werden, der zufällig 10 % der Daten als Testdaten und die restlichen Daten als Lerndaten weiterleitet. Der hier vorgestellte Operator implementiert stattdessen die Evaluationsmethode *Interleaved Test-Than-Train* [8]. Mit dieser werden alle Daten sowohl zum Lernen als auch zum Testen genutzt. Dabei wird ein Testdatum zuerst zum Testen des Modells genutzt und anschließend in das Modell integriert. Das hat die Vorteile, dass mehr Testdaten genutzt werden und keine Daten für das Lernen des Modells verloren gehen. Letzteres ist insbesondere wichtig, wenn ein RecSys im laufenden Betrieb evaluiert werden soll (in dem Fall würde das aussortieren von Daten als Testdaten die Schätzungen für die realen Benutzer beeinflussen) und wenn temporale Zusammenhänge in den Daten beim Modelllernen berücksichtigt werden sollen (in dem Fall würden fehlende Daten ggf. erschweren, die temporalen Zusammenhänge zu erkennen).

Um sicherzustellen, dass zur Schätzung der Bewertung eines jeden Testdatums ein Modell genutzt wird, dass nicht das Testdatum, aber ansonsten so viele Lerndaten wie möglich enthält, wird das Gültigkeitsintervall des Testdatums vom ITTT-Operator angepasst. Ist ein Lerndatum im Gültigkeitsintervall $[t_s, t_e]$ mit dem Startzeitpunkt t_s und dem Endzeitpunkt t_e gültig, so wird die Gültigkeit des Testdatums so gesetzt, dass dieses ungültig wird, gerade bevor das äquivalente Lerndatum gültig wird. Bei einem Lerndatum von einem Gültigkeitsintervall $[45, \infty)$ erhält das Testdatum das Gültigkeitsintervall $[44, 45)$. Es ist genau wie die Empfehlungsanforderungen nur ein Zeitpunkt gültig: zum Zeitpunkt $t_1 = 44$ ist es gültig und zum Zeitpunkt $t_2 = 45$ bereits ungültig (wegen des halboffenen Intervalls). Wie bereits weiter oben gefordert, soll ein Modell genau alle Lerndaten enthalten, welche im Gültigkeitsintervall des Modells ebenfalls gültig sind. Der BRISMF-Operator erzeugt also ein Modell mit dem Lerndatum, welches ab $t_2 = 45$ gültig ist und das vorherige Modell wird somit zum Zeitpunkt $t_2 = 45$ ungültig. Der CROSS-PRODUCT-Operator vor dem PREDICT-RATING-Operator der Evaluation führt nun unter Berücksichtigung der Gültigkeitsintervalle Testdatum und Mo-

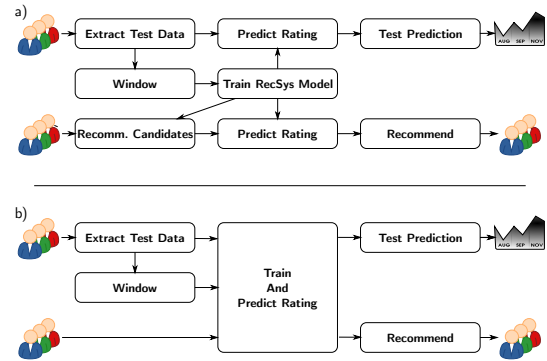


Abbildung 3: Gegenüberstellung: Drei Operatoren vs. ein Operator für die Bewertungsschätzung

dell zusammen. Da sich die Gültigkeitsintervalle des Testdatums und des Modells, welches das zum Testdatum äquivalente Lerndatum enthält, nicht überschneiden, werden diese nicht zusammengeführt. Dafür überschneidet sich das vorherige Modell mit dem Testdatum und wird somit, wie bei der ITTT-Methode gefordert, zur Bewertungsschätzung genutzt.

6. DISKUSSION

Die Aufteilung der Bewertungsschätzung in drei Operatoren

Bei dem hier vorgestellten Konzept haben wir das RecSys-Problem der Schätzung von Bewertungen im Wesentlichen auf drei Operatoren aufgeteilt: TRAIN RECSYS MODEL, RECOMM. CANDIDATES und PREDICT RATING. Das hat den Nachteil, dass eine Kopie des gelernten Modells vom TRAIN-RECSYS-MODEL-Operator an die anderen beiden Operatoren als Datenstromelement übertragen werden muss. Das führt, insbesondere bei großen Modellen, zu einem zusätzlichen Aufwand. Eine Alternative wäre, alle drei Operatoren zu einem Operator zu vereinen (vgl. Abbildung 3). Dies hätte folgende Nachteile:

Durch die Aufteilung in drei Operatoren kann jeder Operator für sich verschiedene Implementierungen (physische Operatoren) haben. Zum Beispiel kann man sich ein RecSys für Filme vorstellen, bei dem der Benutzer angeben kann, dass er z. B. eine Komödie und keinen Actionfilm sehen möchte. Dies würde die Empfehlungsanforderung um eine Genre-Angabe erweitern. Ein für diese Anwendung angepasster RECOMM-CAND-Operator kann von vornherein als Empfehlungskandidaten nur die Objekte berücksichtigen, die unter die Kategorie *Komödie* fallen. Alternativ könnte man zwischen RECOMM CAND und PREDICT RATING einen Selektionsoperator einfügen, der alle Kandidaten, die nicht unter *Komödie* fallen, herausfiltert. Die anderen Operatoren bleiben davon unberührt. Die Aufteilung sorgt somit für klare Zuständigkeiten der Operatoren und ermöglicht eine einfachere Erweiterung der Anfrage.

Der TRAIN-RECSYS-MODEL-Operator bestimmt die Gültigkeit eines Modells, so dass zu jedem Zeitpunkt genau ein Modell gültig ist. Durch die Zusammenführung von Modell und Empfehlungsanforderung durch einen Kreuzprodukt-Operator findet eine deterministische, temporale Zuordnung statt. Diese Zuordnung basiert alleine auf den Gül-

tigkeitsintervallen und ist nicht davon abhängig, ob aufgrund der Steuerung des Schedulers oder anderer Latenzen zuerst das Lerndatum oder die Empfehlungsanforderung den Operator erreicht. Das Zeitmodell des DSMS sorgt dafür, dass genau das zuständige Modell reproduzierbar für die Vorhersage genutzt wird. Des Weiteren kann der TRAIN-RECSYS-MODEL-Operator das Modell für den Gültigkeitszeitraum anpassen, z. B. um temporale Verzerrungen aufgrund von Concept Drift zu berücksichtigen.

Die Ausgabe der Empfehlungskandidaten

Der RECOMM-CAND-Operator gibt für jeden Empfehlungskandidaten einer Empfehlungsanforderung ein Datenstromelement aus. Eine Alternative wäre die Ausgabe eines Datenstromelements je Empfehlungsanforderung, welches eine Liste von Empfehlungskandidaten enthält. Das hätte den Nachteil, dass der PREDICT-RATING-Operator für die Bestimmung der Empfehlungen und für die Evaluation andere Eingabedaten verarbeiten können müsste: im ersten Fall eine Liste von Benutzer-Objekt-Paaren, im zweiten Fall einzelne Benutzer-Objekt-Paare.

Der RECOMM-CAND-Operator gibt nur die Empfehlungskandidaten ohne Modell aus. Dieser Operator könnte auch das Modell den Empfehlungskandidaten beifügen, so könnte man auf den CROSS-PRODUCT-Operator vor PREDICT RATING verzichten. Das hier vorgestellte Konzept hat den Vorteil, dass BRISMF an RECOMM CAND und PREDICT RATING unterschiedliche Modelle übergeben kann. Wenn zum Beispiel aus dem Modell für die Schätzung der Bewertung gar nicht hervorgeht, welche Objekte ein Benutzer nicht bewertet hat (und somit zu den Empfehlungskandidaten gehört), so kann an RECOMM CAND ein Modell übergeben werden, dass die Zuordnung von Benutzer zu unbewerteten Objekten ermöglicht, dafür aber keine Schätzung der Bewertung vornehmen kann (z. B. ein einfaches Mapping $u \mapsto$ unbewertete Objekte).

7. IMPLEMENTIERUNG UND EVALUATION

Das vorgestellte Konzept haben wir mit dem erweiterbaren Open-Source-DSMS *Odysseus*¹ [2] umgesetzt. Dazu haben wir *Odysseus* um neue Operatoren und Transformationsregeln erweitert.

Die Machbarkeit des Konzepts und die korrekte Implementierung konnten wir durch den Vergleich der Evaluationsergebnisse mit MOA und dem MovieLens-Datensatz nachweisen. Dazu haben wir als BRISMF-Operator die MOA-Implementierung des BRISMF-Algorithmus¹ [11] integriert, der eine inkrementelle Matrix-Faktorisierung implementiert. Da MOA die Gültigkeit von Lerndaten nicht begrenzt, haben wir den WINDOW-Operator entfernt und die RMSE über den gesamten Datensatz berechnet (kein *gleitender* RMSE). Den MovieLens-Datensatz haben wir, wie MOA, zeilenweise eingelesen um einen Datenstrom zu simulieren und haben nach jedem getesteten Tupel den RMSE ausgegeben. Die RMSE-Werte stimmen dabei mit denen von MOA exakt überein. Das zeigt, dass die temporale Zuordnung von Lern- und Testdaten mit den Modellen sowie die Umsetzung der Evaluationsmethode korrekt funktionieren.

8. ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit haben wir ein Konzept für die Umsetzung eines modellbasierten und kollaborativen RecSys mit einem auf Operatoren basierten DSMS vorgestellt. Dazu haben wir logische Operatoren definiert, die für Teilaufgaben eines RecSys zuständig sind. Zu einem Beispiel eines logischen Anfrageplans für ein RecSys haben wir eine beispielhafte Umsetzung durch einen physischen Anfrageplan gezeigt und die Umsetzung und deren Alternativen diskutiert. Unser Konzept haben wir durch eine Implementierung in dem DSMS *Odysseus* und dem Vergleich der Evaluationsergebnisse mit MOA validiert.

Um die Praxistauglichkeit nachzuweisen, soll das Konzept in Zukunft unter realen Bedingungen, z. B. in einem Living Lab wie *Newsreel*² [9], evaluiert werden. Dazu stehen insbesondere die für den Datenstromkontext wichtige Parameter Latenz, Durchsatz und Speicheranforderungen im Vordergrund. Des Weiteren sollen weitere Algorithmen für die Empfehlungen und die Evaluation (z. B. Ranking-basierte Methoden) implementiert werden.

9. REFERENCES

- [1] M. Ali, C. C. Johnson, and A. K. Tang. Parallel collaborative filtering for streaming data. *University of Texas Austin, Tech. Rep*, 2011.
- [2] H.-J. Appelrath, D. Geesen, M. Grawunder, T. Michelsen, and D. Nicklas. *Odysseus: A highly customizable framework for creating efficient event stream management systems*. In *DEBS'12*, pages 367–368. ACM, 2012.
- [3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.
- [4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS 2002*, pages 1–16. ACM, 2002.
- [5] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.
- [6] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. StreamRec: A Real-time Recommender System. In *SIGMOD'11*, pages 1243–1246. ACM, 2011.
- [7] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-Time Top-N Recommendation in Social Streams. In *ACM RecSys*, pages 59–66, 2012.
- [8] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A Survey on Concept Drift Adaptation. *ACM Comp. Surveys*, 1(1), 2013.
- [9] F. Hopfgartner, B. Kille, A. Lommatzsch, T. Plumbaum, T. Brodt, and T. Heintz. Benchmarking news recommendations in a living lab. In *CLEF'14*, LNCS, pages 250–267. Springer Verlag, 09 2014.
- [10] X. Luo, Y. Xia, and Q. Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. *Knowledge-Based Systems*, 27:271–280, 2012.
- [11] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.

¹<http://odysseus.informatik.uni-oldenburg.de/>

²<http://www.clef-newsreel.org/>

Das PARADISE-Projekt

Big-Data-Analysen für die Entwicklung von Assistenzsystemen (Extended Abstract)*

Andreas Heuer
Lehrstuhl DBIS, Institut für Informatik
Universität Rostock
18051 Rostock, Deutschland
heuer@informatik.uni-rostock.de

Holger Meyer
Lehrstuhl DBIS, Institut für Informatik
Universität Rostock
18051 Rostock, Deutschland
hme@informatik.uni-rostock.de

ZUSAMMENFASSUNG

Bei der Erforschung und systematischen Entwicklung von Assistenzsystemen fallen eine große Menge von Sensordaten an, aus denen Situationen, Handlungen und Intentionen der vom Assistenzsystem unterstützten Personen abgeschätzt (modelliert) werden müssen. Neben Privatheitsaspekten, die bereits während der Phase der Modellbildung berücksichtigt werden müssen, sind die *Performance* des Analysesystems sowie die *Provenance* (Rückverfolgbarkeit von Modellierungsentscheidungen) und die *Preservation* (die langfristige Aufbewahrung der Forschungsdaten) Ziele unserer Projekte in diesem Bereich. Speziell sollen im Projekt PARADISE die Privatheitsaspekte und die Performance des Systems berücksichtigt werden. In einem studentischen Projekt wurde innerhalb einer neuen *experimentellen* Lehrveranstaltung im reformierten Bachelor- und Master-Studiengang Informatik an der Universität Rostock eine Systemplattform für eigene Entwicklungen geschaffen, die auf Basis von klassischen zeilenorientierten Datenbanksystemen, aber auch spaltenorientierten und hauptspeicheroptimierten Systemen die Analyse der Sensordaten vornimmt und für eine effiziente, parallelisierte Verarbeitung vorbereitet. Ziel dieses Beitrages ist es, die Ergebnisse dieser studentischen Projektgruppe vorzustellen, insbesondere die Erfahrungen mit den gewählten Plattformen PostgreSQL, DB2 BLU, MonetDB sowie R (als Analysesystem) zu präsentieren.

1. EINLEITUNG

Ein Forschungsschwerpunkt am Institut für Informatik der Universität Rostock ist die Erforschung und systematische Entwicklung von Assistenzsystemen, etwa im DFG-Graduiertenkolleg MuSAMA. Da in Assistenzsystemen unterstützte Personen durch eine Vielzahl von Sensoren beobachtet werden, müssen bei der datengetriebenen Modellie-

*Eine Langfassung dieses Artikels ist erhältlich als [HM15] unter <http://www.ls-dbis.de/digbib/dbis-tr-cs-04-15.pdf>

rung von Situationen, Handlungen und Intentionen der Personen aus großen Datenmengen mittels Machine-Learning-Methoden entsprechende Modelle abgeleitet werden: ein Performance-Problem bei einer Big-Data-Analytics-Fragestellung.

Da Personen *beobachtet* werden, müssen auch Privatheitsaspekte bereits während der Phase der Modellbildung berücksichtigt werden, um diese bei der konkreten Konstruktion des Assistenzsystems automatisch in den Systementwurf zu integrieren. Somit gibt es für die Datenbankforscher unter anderem die Teilprobleme der performanten Berechnung der Modelle als auch der Wahrung der Privatheitsansprüche des Nutzers, die zu lösen sind und die in einer langfristigen Projektgruppe des Datenbanklehrstuhls angegangen werden: im Projekt PARADISE (Privacy AwaRe Assistive Distributed Information System Environment) werden effiziente Techniken zur Auswertung von großen Mengen von Sensordaten entwickelt, die definierte Privatheitsansprüche der späteren Nutzer per Systemkonstruktion erfüllen.

Während wir in [Heu15] ausführlicher auf die Verknüpfung der Aspekte *Privatheit* (Projekt PARADISE) und *Provenance* (Projekt METIS) eingegangen sind, werden wir uns in diesem Beitrag auf die beiden Schwerpunkte des PARADISE-Projektes konzentrieren, das ist neben der Privatheit die *Performance* durch Parallelität und Verteilung.

2. ASSISTENZSYSTEM-ENTWICKLUNG ALS BIG-DATA-PROBLEM

Um seine Assistenzaufgaben zu erfüllen, besteht ein Assistenzsystem üblicherweise aus fünf Schichten [Heu15]. In der untersten Schicht werden ständig viele Daten (etwa von Sensoren) erzeugt, in der obersten Schicht wird aber nur im Bedarfsfall (also eher selten) ein akustischer oder optischer Hinweis, also eine geringe Datenmenge, ausgegeben.

In der mittleren der fünf Schichten müssen Sensordaten gefiltert, erfasst, ausgewertet, verdichtet und teilweise langfristig verwaltet werden. Aufgrund der extrem großen Datenmenge (Big Data) muss die **Verarbeitung verteilt** erfolgen: teilweise eine Filterung und Verdichtung schon im Sensor, im nächsterreichbaren Prozessor (etwa im Fernseher oder im Smart Meter in der Wohnung) und im Notfall über das Internet in der Cloud. Neben Daten des Assistenzsystems müssen auch fremde Daten etwa über das Internet berücksichtigt werden, beispielsweise Wartungspläne beim Auto oder die elektronische Patientenakte beim Patienten. Allgemein können hier natürlich auch die Daten sozialer Netz-

werke, Kalenderdaten der Nutzer oder Wettervorhersagedaten ausgewertet werden, falls sie für das Assistenzziel eine Rolle spielen.

Eine Kernaufgabe bei der Erforschung und Entwicklung ist die datengetriebene Modellierung von Situationen, Handlungen und Intentionen, die eine Fragestellung im Forschungsgebiet Big Data Analytics sind. Big Data [Mar15] ist ein derzeitiges Hype-Thema nicht nur in der Informatik, das in seiner technischen Ausprägung auf vielfältige Forschungsprobleme führt. Technisch gesehen sind Big-Data-Probleme mit den vier *V* (Volume, Velocity, Variety, Veracity) charakterisiert. *Big Data Analytics* ist nun das Problem komplexer Analysen auf diesen Daten. In Datenbankbegriffen sind diese komplexen Analysen iterative Anfrageprozesse.

3. DIE VIER P ZU DEN VIER V

Die Forschungsschwerpunkte der Rostocker Datenbankgruppe lassen sich in diesem Zusammenhang mit vier *P* charakterisieren, die im Folgenden näher erläutert werden sollen.

Forschung und Entwicklung: In der Forschungs- und Entwicklungsphase eines Assistenzsystems ist das vorrangige Ziel, eine effiziente Modellbildung auf großen Datenmengen zu unterstützen. Dabei sollte möglichst automatisch eine Selektion der Daten (Filterung wichtiger Sensordaten nach einfachen Merkmalen) und eine Projektion der Daten (die Beschränkung der großen Sensormenge auf wenige, besonders aussagekräftige Sensoren) vorgenommen werden. Die nötige Effizienz in dieser Phase führt auf unser Forschungsthema **P3: Performance**. Da während der Entwicklung bei fehlerhafter Erkennung von Handlungen und Intentionen die dafür zuständigen Versuchsdaten ermittelt werden müssen, führt die Rückverfolgbarkeit der Analyseprozesse in der Entwicklung auf unsere Forschungsthemen **P2: Provenance Management** und **P4: Preservation** (Langfristarchivierung von Forschungsdaten).

Einsatz: In der Einsatzphase eines Assistenzsystems sind dagegen Privatheitsansprüche vorherrschend, die im Gesamtsystem durch stufenweise Datensparsamkeit erreicht werden können (unser Forschungsthema **P1: Privatheit**). Eine weitere Verdichtung (auch Reduktion und Aggregation) der live ausgewerteten Daten unterstützen aber nicht nur die Privatheit, sondern auch die Performance.

Die vier *P* behandeln wir in drei langfristigen Forschungsprojekten (METIS, PArADISE, HyDRA), in diesem Beitrag konzentrieren wir uns auf den Aspekt **P3 (Performance)** des PArADISE-Projektes.

4. DAS PARADISE-PROJEKT

Im Projekt PArADISE (Privacy AwaRe Assistive Distributed Information System Environment) arbeiten wir derzeit an Techniken zur Auswertung von großen Mengen von Sensordaten, die definierte Privatheitsansprüche der späteren Nutzer per Systemkonstruktion erfüllen.

Ein erster Prototyp ist von einer studentischen Arbeitsgruppe erstellt worden. Derzeit können Analysen zur Modellbildung auf Sensordaten in SQL-92, SQL:2003 oder iterativen Ansätzen über SQL-Anweisungen realisiert und auf die Basissysteme DB2 (zeilenorientiert oder spaltenorientiert: DB2 BLU), PostgreSQL (zeilenorientiert) sowie MonetDB (spaltenorientiert und Hauptspeicheroptimiert) abgebildet werden.

Während die grundlegenden Forschungsarbeiten zu PArADISE durch zwei Stipendiaten des Graduiertenkollegs MuSAMA (Hannes Grunert und Dennis Marten) in 2013 und 2014 starteten, wurden die ersten softwaretechnischen Umsetzungen des Projektes durch eine studentische Projektgruppe im Wintersemester 2014/2015 vorgenommen. Hier wurden dann verschiedene SQL-Anfragen und R-Programme zur Lösung der grundlegenden Regressions- und Korrelationsprobleme entwickelt, wobei als Vorgabe (zum Vergleich) folgende fünf Stufen realisiert werden sollten:

1. Umsetzung von Regression und Korrelation in Standard-SQL-92 (also per Hand, da keine Analysefunktionen außer den klassischen Aggregatfunktionen wie COUNT, SUM und AVG vorhanden).
2. Umsetzung in SQL:2003 mit den entsprechenden OLAP-Funktionen.
3. Umsetzung mit rekursivem oder iterativem SQL, sofern in den Systemen möglich.
4. Eine Integration der SQL-Anfrage mit R-Auswertungen.
5. Eine R-Auswertung pur ohne Kopplung an SQL.

Die in MuSAMA bisher verwendete Lösung mit *Plain R* wies dabei die schlechteste Effizienz auf, auch wenn man den Prozess des initialen Ladens der Daten in den Hauptspeicher herausrechnet. Unter den Varianten mit einer Analyse in reinem SQL-92 (Regression per Hand mit Aggregatfunktionen umgesetzt) war die MonetDB-Lösung etwas besser als die DB2-Variante, PostgreSQL fiel stärker ab. Die SQL:2003-Lösung konnte in MonetDB mangels vorhandener OLAP- und Rekursions-Fähigkeiten nicht umgesetzt werden, DB2 war hier wiederum deutlich besser als PostgreSQL. Weiterhin bemerkt man im Vergleich von SQL-92 und SQL:2003, dass der Optimierer von DB2 als auch PostgreSQL die direkte Verwendung der OLAP-Funktionen belohnt. Die beste Performance aller Varianten erreichte jedoch MonetDB mit integrierten R-Funktionen.

5. DANKSAGUNGEN

Wir danken der studentischen Projektgruppe PArADISE im Wintersemester 2014/2015, die im Rahmen einer experimentellen Projekt-Lehrveranstaltung die Basis für die softwaretechnische Umsetzung des PArADISE-Projektes gelegt hat: Pia Wilsdorf, Felix Köppl, Stefan Lüdtkke, Steffen Sachse, Jan Svacina, Dennis Weu.

6. LITERATUR

- [Heu15] Heuer, A.: METIS in PArADISE: Provenance Management bei der Auswertung von Sensordatenmengen für die Entwicklung von Assistenzsystemen. In: *Lecture Notes in Informatics, Band 242, BTW 2015 Workshop-Band, 131 – 135*, 2015.
- [HM15] Heuer, A.; Meyer, H.: Das PArADISE-Projekt: Big-Data-Analysen für die Entwicklung von Assistenzsystemen. Technischer Bericht CS-04-15, Institut für Informatik, Universität Rostock, 2015.
- [Mar15] Markl, V.: Gesprengte Ketten - Smart Data, deklarative Datenanalyse, Apache Flink. *Informatik Spektrum*, Band 38, Nr. 1, S. 10–15, 2015.