Data Profiling – Effiziente Entdeckung Struktureller Abhängigkeiten[1]

Thorsten Papenbrock[2]

Abstract: Daten sind nicht nur in der Informatik, sondern auch in allen anderen wissenschaftlichen Disziplinen ein unverzichtbares Wirtschaftsgut. Sie dienen dem Austausch, der Verknüpfung und der Speicherung von Wissen und sind daher unverzichtbar in Forschung und Wirtschaft. Leider sind Daten häufig nicht ausreichend dokumentiert um sie direkt nutzen zu können – es fehlen Metadaten, welche die Struktur und damit Zugriffsmuster der digitalen Informationen beschreiben. Informatiker und Experten anderer Disziplinen verbringen daher viel Zeit damit, Daten strukturell zu analysieren und aufzubereiten. Da die Suche nach Metadaten jedoch eine hoch komplexe Aufgabe ist, scheitern viele algorithmische Ansätze schon an kleinen Datenmengen.

In dieser Dissertation stellen wir daher drei neuartige Entdeckungsalgorithmen für wichtige und zugleich schwierig zu findende Typen von Metadaten vor: Eindeutige Spaltenkombinationen, funktionale Abhängigkeiten und Inklusionsabhängigkeiten. Die vorgeschlagenen Algorithmen übertreffen deutlich den bisherigen Stand der Technik in Laufzeit und Ressourcenverbrauch und ermöglichen so die Nutzbarmachung von erheblich größeren Datensätzen. Da die Anwendung solcher Algorithmen für fachfremde Nutzer nicht einfach ist, schlagen wir außerdem das Programm METANOME vor, das ein praktisches Werkzeug zur Datenanalyse darstellt. METANOME macht dabei nicht nur die in dieser Arbeit vorgeschlagenen Algorithmen nutzbar, sondern auch Entdeckungsalgorithmen für andere Typen von Metadaten. Am Anwendungsfall der Schema-Normalisierung zeigen wir schließlich, wie die effektive Nutzung der gefundenen Metadaten erfolgen kann.

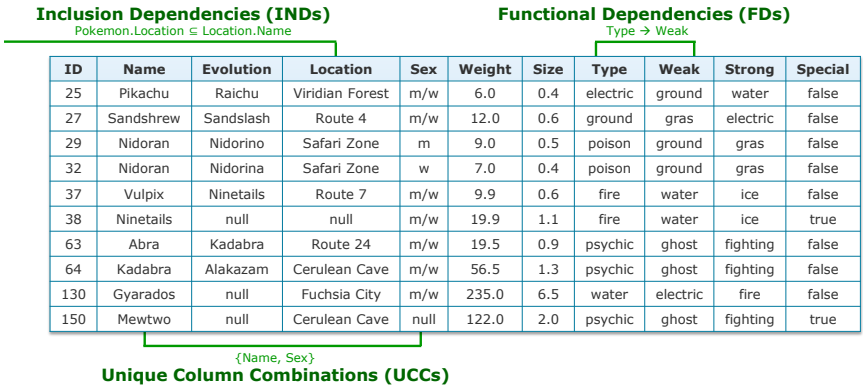
1 Extraktion struktureller Metadaten

Data Profiling ist eine Disziplin der Informatik, in der Datensätze mit dem Ziel analysiert werden, deren Metadaten zu bestimmen. Die verschiedenen Typen von Metadaten reichen von einfachen Statistiken wie Tupelzahlen, Spaltenaggregationen und Wertverteilungen bis hin zu weit komplexeren Strukturen, insbesondere Inklusionsabhängigkeiten (INDs), eindeutige Spaltenkombinationen (UCCs) und funktionale Abhängigkeiten (FDs). Sofern vorhanden dienen diese Statistiken und Strukturen dazu, die Daten zu verstehen, sie effizient zu speichern, zu lesen und zu ändern. Da die meisten Datensätze ihre Metadaten aber nicht explizit als beschreibendes Regelwerk zur Verfügung stellen, sind Informatiker häufig gezwungen diese strukturellen Regeln mittels Data Profiling zu bestimmen.

Während einfache Statistiken noch relativ schnell zu berechnen sind, stellen die komplexeren Strukturen schwere, zumeist NP-vollständige Entdeckungsaufgaben dar. In der Regel ist es daher auch mit gutem Domänenwissen nicht möglich, sie händisch zu bestimmen. Es

[1] Englischer Titel der Dissertation: “Data Profiling – Efficient Discovery of Dependencies”

[2] Universität Potsdam, Hasso-Plattner-Institut, Informationssysteme, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Deutschland thorsten.papenbrock@hpi.de



ID	Name	Evolution	Location	Sex	Weight	Size	Type	Weak	Strong	Special
25	Pikachu	Raichu	Viridian Forest	m/w	6.0	0.4	electric	ground	water	false
27	Sandshrew	Sandslash	Route 4	m/w	12.0	0.6	ground	gras	electric	false
29	Nidoran	Nidorino	Safari Zone	m	9.0	0.5	poison	ground	gras	false
32	Nidoran	Nidorina	Safari Zone	w	7.0	0.4	poison	ground	gras	false
37	Vulpix	Ninetails	Route 7	m/w	9.9	0.6	fire	water	ice	false
38	Ninetails	null	null	m/w	19.9	1.1	fire	water	ice	true
63	Abra	Kadabra	Route 24	m/w	19.5	0.9	psychic	ghost	fighting	false
64	Kadabra	Alakazam	Cerulean Cave	m/w	56.5	1.3	psychic	ghost	fighting	false
130	Gyarados	null	Fuchsia City	m/w	235.0	6.5	water	electric	fire	false
150	Mewtwo	null	Cerulean Cave	null	122.0	2.0	psychic	ghost	fighting	true

Abb. 1: Eine relationale Tabelle mit Daten über Pokémon, die drei ausgewählte Schlüsselbeziehungen zeigt: ein potentieller Primärschlüssel (UCC), ein Fremdschlüssel (IND) und eine innere Schlüsselabhängigkeit (FD).

wurden daher bereits verschiedenste Profiling Algorithmen entwickelt, um die Entdeckung zu automatisieren. Keiner dieser Algorithmen kann allerdings Datensätze von heutzutage typischer Größe verarbeiten, weil entweder der Ressourcenverbrauch oder die Rechenzeit effektive Grenzen überschreitet.

In dieser Arbeit stellen wir neuartige Profiling Algorithmen vor, die automatisch die drei populärsten Typen komplexer Metadaten entdecken, nämlich UCCs, FDs und INDs. Die Popularität dieser drei Strukturen begründet sich in der Tatsache, dass mit ihrer Hilfe die wichtigsten Formen von Schlüssel-Abhängigkeiten beschrieben werden: UCCs beschreiben Schlüssel *für* eine relationale Tabelle, FDs beschreiben Schlüssel *innerhalb* einer relationalen Tabelle und INDs beschreiben Fremdschlüsselbeziehungen *zwischen* relationalen Tabellen. Sie dienen damit nicht nur der Identifikation von Entitäten in einem Datensatz, sondern auch der Verknüpfung, Bereinigung, Anfrage, Integration und logischen Formatierung von Daten. Abbildung 1 zeigt eine Beispielrelation über Pokémon Daten – kleine “Taschenmonster” für Kinder. Von den ebenfalls dargestellten Schlüsselabhängigkeiten lernen wir, dass Pokémon über ihren Namen und ihr Geschlecht eindeutig identifiziert werden (siehe UCC), der Typ eines Pokémon auch dessen Schwäche bestimmt und zusätzliche Informationen über die Herkunft eines Pokémon in einer speziellen anderen Tabelle gefunden werden können.

Die Aufgabe eines Entdeckungsalgorithmus ist es alle gültigen Vorkommen einer Schlüssel-Abhängigkeiten aus einer gegebenen relationalen Instanz zu extrahieren â- ein induktives Such- und Prüfverfahren also, welches die Daten systematisch auf Schlüsseleigenschaften untersucht. Die von uns entwickelten Algorithmen nutzen dazu sowohl bewährte Entdeckungstechniken aus verwandten Arbeiten des Data Profiling, als auch für das Data Profiling neuartige Techniken, wie beispielsweise Teile-und-Herrsche Verfahren, hybrides Suchen, Progressivität, Speichersensibilität, Parallelisierung und innovative Streichungsregeln. Über eine Reihe systematischer Experimente zeigen wir, dass die vorgeschlagenen Algorithmen nicht nur um Größenordnungen schneller sind als alle verwandten Algorith-

men, sie heben auch einige der aktuellen Beschränkungen auf, da sie in der Lage sind Datensätze von häufig vorkommender Größe, d.h. mehrerer Gigabyte Größe, mit akzeptablem Speicher- und Zeitverbrauch zu verarbeiten. Um die entwickelten Algorithmen der Forschungs- und Entwicklungsgemeinschaft, sowie Informatik-Laien zugänglich zu machen, haben wir alle Verfahren in das praktische Profiling Werkzeug METANOME[2] integriert, welches frei und quelloffen verfügbar ist. Zusammengefasst leistet diese Arbeit daher die folgenden Beiträge:

1. HYFD: Ein effizienter Algorithmus zur Entdeckung funktionaler Abhängigkeiten, der ein hybrides Suchverfahren zur Identifikation valider FDs im Suchraum einsetzt [PN16].
2. HYUCC: Ein effizienter Algorithmus zur Entdeckung eindeutiger Spaltenkombinationen, der ebenfalls auf eine hybride Suche setzt, um Schlüsselkandidaten zu finden [PN17].
3. BINDER: Ein effizienter Algorithmus zur Entdeckung von Inklusionsabhängigkeiten, der mittels Datenpartitionierung die Prüfung von IND-Kandidaten beschleunigt [Pa15c].
4. METANOME: Ein leicht erweiterbares Data Profiling Werkzeug, das verschiedene Entdeckungsalgorithmen praktisch nutzbar macht [Pa15a].
5. NORMALIZE: Ein Algorithmus zur Schema-Normalisierung, der entdeckte Schlüssel-Abhängigkeiten automatisiert bewertet und zur Schema-Reorganisation einsetzt [Pa17].

Im Folgenden erläutern wir zunächst die theoretischen Grundlagen für diese Arbeit und fassen anschließend die einzelnen Beiträge der Dissertation kapitelweise zusammen. Die vollständige Fassung der Dissertation ist in englischer Sprache erschienen [Pa17].

2 Schlüsselabhängigkeiten

Das relationale Datenmodell stellt Daten in tabellarischer Form mittels eines festen Schemas dar und ist damit das zur Zeit am weitesten verbreitete Modell. Jede Spalte hat eine eindeutige Bezeichnung, und jede Zeile beschreibt eine in der Tabelle gespeicherte Entität. Spalten und Zeilen werden häufig auch als Attribute und Einträge bezeichnet. Wir definieren nun unsere drei Schlüsselabhängigkeiten als Beziehungen zwischen verschiedenen Spalten:

Funktionale Abhängigkeiten (FDs) werden geschrieben als $X \rightarrow A$ und drücken damit aus, dass alle Paare von Einträgen mit gleichem Wert in der Attribut-Menge X auch den gleichen Wert im Attribut A haben – die X-Werte bestimmen funktional die A-Werte. Wenn wir die relationale Instanz mit r und ihr Schema mit R bezeichnen, dann definiert sich dieser Zusammenhang formal als $\forall t_i, t_j \in r : t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A]$, wobei $X \subseteq R$ und $A \in R$ ist. Um alle funktionalen Abhängigkeiten eines Datensatzes zu bestimmen reicht es aus, nur minimale Abhängigkeiten aufzuzählen, bei denen aus der Attributmenge X kein Attribut entnommen werden kann, ohne die FD zu verletzen. Da das Hinzufügen beliebiger Attribute auf der linken Seite der FD die Abhängigkeit nicht verletzt, lassen sich alle nicht-minimalen FDs aus der vollständigen Menge aller minimalen FDs leicht ableiten.

[2] www.metanome.de

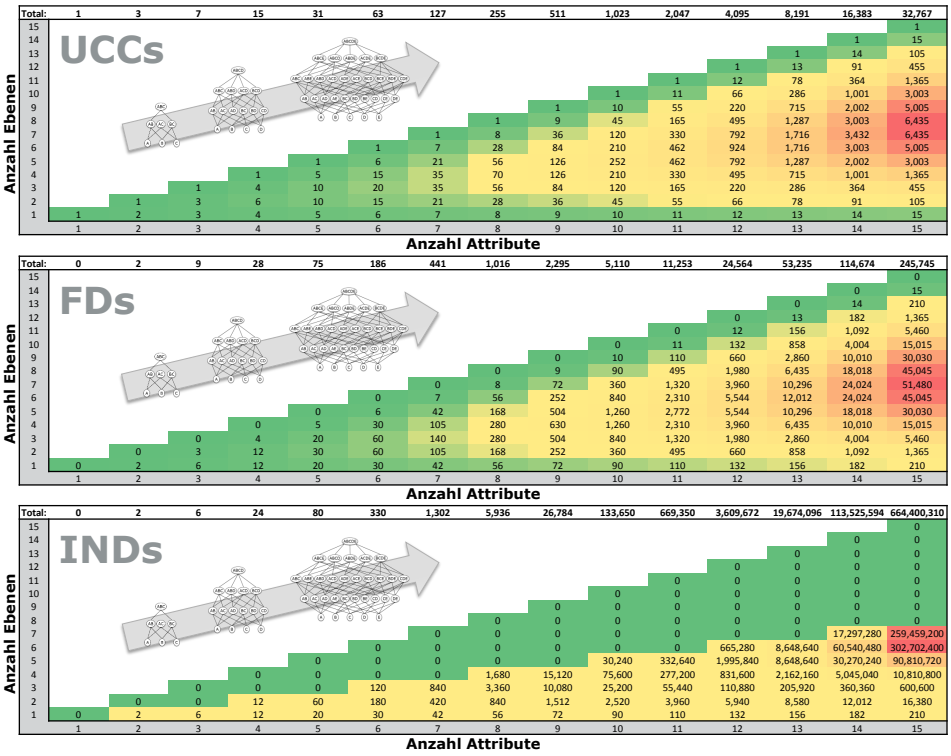


Abb. 2: Wachstum der Suchräume für UCCs, FDs und INDs mit der Anzahl Attribute.

Eindeutige Spaltenkombinationen (UCCs) sind Mengen von Attributen X, in denen kein Wert (bzw. keine Wertekombination) doppelt vorkommt. Jeder einzelne Wert identifiziert daher eindeutig einen bestimmten Eintrag in der Tabelle. Formal definieren wir eindeutige Spaltenkombinationen X mit $X \subseteq R$ als $\forall t_i, t_j \in r, i \neq j : t_i[X] \neq t_j[X]$. Analog zu FDs reicht es zur Entdeckung aller UCCs aus nur solche aufzuzählen, die minimal sind. Eine minimale eindeutige Spaltenkombination ist jene, die bei Entfernung eines beliebigen Attributes aus X ungültig wird. Aus den minimalen UCCs lassen sich dann ebenfalls alle anderen UCCs ableiten, indem diese durch weitere Attribute ergänzt werden.

Inklusionsabhängigkeiten (INDs) werden geschrieben als $R_i[X] \subseteq R_j[Y]$ und besagen, dass alle Werte (bzw. Wertekombinationen) der Attribute X auch in der Menge der Werte von Y vorkommen – sie sind also darin enthalten. Nützlich ist dieser Zusammenhang, weil über Join-Operationen die Einträge, die an diesen Werten hängen, miteinander verbunden werden können. Eine formale Definition dieses Zusammenhangs ist $\forall t_i[X] \in r_i, \exists t_j[Y] \in r_j : t_i[X] = t_j[Y]$, wobei r_i und r_j relationale Instanzen der Schemata R_i und R_j darstellen. Im Gegensatz zu den meisten anderen Arten von Datenabhängigkeiten suchen wir bei INDs nach maximalen Ausdrücken, da beim Entfernen von Attributen auf linker und rechter Seite der Beziehung die Gültigkeit immer erhalten bleibt, die Gültigkeit aber verloren gehen kann, wenn Attribute hinzugefügt werden.

Die Mengen aller FD, UCC und IND Kandidaten wird bestimmt durch die Potenzmengen ihrer linken Seiten. Der Suchraum wird daher häufig als Gitter (engl. lattice) von Kandidaten modelliert: Beginnend mit Beziehungen zwischen einzelnen Attributen werden sukzessiv mehr Attribute zu diesen Kandidaten hinzugefügt. Abbildung 2 visualisiert wie diese Gitter, also die Suchraumgrößen, für die einzelnen Abhängigkeiten mit der Anzahl an Attributen im Datensatz immer weiter wachsen. Bezüglich der Anzahl von Attributen m liegen die Komplexitäten der automatischen Entdeckung dieser drei Arten von Metadaten daher in $\mathcal{O}(2^m)$ für UCCs, $\mathcal{O}(2^m \cdot (\frac{m}{2})^2)$ für FDs und $\mathcal{O}(2^m \cdot m!)$ für INDs (wobei $m!$ eine Vereinfachung ist) [Li12].

3 Entdeckung von funktionalen Abhängigkeiten

Um effizient alle minimalen funktionalen Abhängigkeiten zu finden haben verwandte Arbeiten bereits zwei unterschiedliche Ansätze vorgeschlagen: Der erste Ansatz testet die FD Kandidaten im Suchraumgitter systematisch von unten nach oben und streicht dabei als ungültig erkennbare Kandidaten von der weiteren Suche [Hu99]; der zweite Ansatz vergleicht alle Paare von Einträgen in der Tabelle, berechnet aus diesen Vergleichen alle Verletzungen der FDs (entsprechend ihrer Definition) und leitet am Ende aus der Menge aller Verletzungen die Menge aller gültigen minimalen FDs ab [FS99]. In einer evaluierenden Arbeit [Pa15b] haben wir festgestellt, dass der erste Ansatz auf breiten Datensätzen (ca. 40 Attribute und aufwärts) ineffizient wird und der zweite Ansatz bei langen Datensätzen (ca. 100.000 Einträge und aufwärts) Schwächen aufweist, viele Datensätze aber sowohl breit als auch lang sind. Wir schlagen daher den hybriden Algorithmus HyFD vor, der beide Strategien so kombiniert, dass sie sich gegenseitig unterstützen, um sowohl auf breiten als auch langen Datensätzen effizienter zu sein als je eine Strategie für sich allein.

Abbildung 3 veranschaulicht die hybride Suchstrategie von HyFD: In der `Preprocessor` Komponente wird der Datensatz zunächst in kompakte Indexstrukturen übersetzt: Positionslisten-Index (PLI) und komprimierter Entitäten-Index (PLIRecord). Anschließend beginnt die `Sampler` Komponente bestimmte Zeilen in der Tabelle zu vergleichen und daraus Verletzungen der FDs zu berechnen. Dies ist eine der beiden Suchstrategien, die allerdings abgebrochen wird, sobald eine überwiegende Mehrheit an Zeilenvergleichen keine neuen Verletzungen mehr geliefert hat – die Strategie ist ineffizient geworden. Daraufhin leitet der `Inductor` des HyFD Algorithmus all jene FD Kandidaten ab, die unter Berücksichtigung der bisher gefundenen Verletzungen noch minimal und gültig sein können. Das Ergebnis formt nun ein Kandidatenset, welches die `Validator` Komponente systematisch mit dem Gitter-Verfahren zu überprüfen beginnt. Auch hier gilt nun, dass HyFD die Validierung dynamisch abbricht, sobald eine Überzahl an Überprüfungen ein negatives Ergebnis liefern und sich die Validierung als ineffizient herausstellt. Wir tauschen nun Vergleichsvorschläge mit der `Sampler` Komponente aus und wechseln zurück zur ersten Suchstrategie. Alle Ergebnisse, die vom `Validator` bisher als gültig identifiziert wurden, sind exakte, minimale funktionale Abhängigkeiten und werden ausgegeben. Der Algorithmus terminiert, sobald dem `Validator` keine weiteren Kandidaten mehr vorliegen.

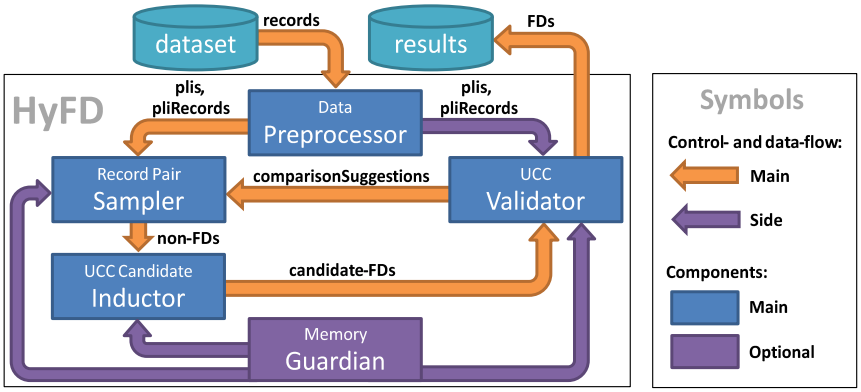


Abb. 3: Übersicht über den HyFD Algorithmus.

Tabelle 1 zeigt die Laufzeiten von HyFD für verschiedene Echtwelt-Datensätze. Die aufgeführten Datensätze bereiten den bisherigen Entdeckungsalgorithmen große Schwierigkeiten, da diese Ansätze entweder ein Speichervolumen von mehr als 100 GB überschreiten oder auch innerhalb mehrerer Tage kein vollständiges Ergebnis berechnen können. Der hybride Ansatz konnte jede Analyse auf einem Rechner mit 16 Prozessorkernen und 128 GB RAM (wovon tatsächlich aber nur ein Bruchteil genutzt wird) problemlos durchführen. Vor allem aber zeigt sich in diesen (und weiteren) Experimenten, dass die Laufzeit mehr von der zu findenden Ergebnisgröße abhängt als von der Größe des Datensatzes – eine ideale Laufzeiteigenschaft für Profiling Algorithmen.

Datensatz	Spalten [#]	Zeilen [#]	Größe [MB]	FDs [#]	HyFD [s/m/h/d]
TPC-H.lineitem	16	6 m	1,051	4 k	4 m
PDB.ATOM_SITE	31	27 m	5,042	10 k	64 m
SAP_R3.ILOA	48	45 m	8,731	16 k	8 h
SAP_R3.CE4HI01	65	2 m	649	2 k	10 m
NCVoter.statewide	71	1 m	561	5 m	31 h
UCI.flight	109	1 k	1	982 k	54 s

Tab. 1: Laufzeiten des HyFD Algorithmus auf verschiedenen Datensätzen.

4 Entdeckung von eindeutigen Spaltenkombinationen

Zur Entdeckung eindeutiger Spaltenkombinationen nutzt der HyUCC Algorithmus eine sehr ähnliche Strategie wie der HyFD Algorithmus: Nach der Komprimierung des Datensatzes in Indexstrukturen wechseln sich eine Suchstrategie, die auf dem Vergleich von Einträgen basiert, und eine Suchstrategie, die Kandidaten im Suchgitter vergleicht, gegenseitig ab. Die Strategien tauschen Zwischenergebnisse untereinander aus und schlagen zudem ein paar Optimierungen wie beispielsweise das frühzeitige Abbrechen und Parallelisieren von Kandidaten-Checks vor. So ist auch HyUCC allen bisherigen Ansätzen der UCC Entdeckung, die auch jeweils immer nur auf einer Suchstrategie beruhen, überlegen.

Mit HyUCC konnten wir zeigen, dass hybrides Suchen nicht nur für funktionale Abhängigkeiten, sondern auch für viele weitere Arten von komplexen Metadaten eine vielversprechende Strategie ist. So haben sich weitere Entdeckungsalgorithmen mit demselben Suchprinzip angeschlossen, nämlich Aid-FD [Bl16b] für approximative funktionale Abhängigkeiten (approximate FDs), MvdDetector [Dr16] für sogenannte Multivalued Dependencies (MVDs), Hydra [Bl16a] für Denial Constraints (DCs) und HyMD [Dr18] für Matching Dependencies.

5 Entdeckung von Inklusionsabhängigkeiten

Die größte Herausforderung bei der Suche nach Inklusionsabhängigkeiten in einem relationalen Datensatz ist die effiziente Ausführung einer extrem großen Anzahl von Kandidaten-überprüfungen. Jede einzelne Überprüfung eines IND Kandidaten ist dabei aufwendiger als die Überprüfung eines FD oder UCC Kandidaten, weil nicht nur die Positionen gleicher Werte in einer Spalte relevant sind, sondern das exakte Übereinstimmen von Werten in verschiedenen Spalten. Eine Komprimierung des Datensatzes in Positionslisten ist daher nicht möglich. Um die Kandidaten möglichst effizient zu prüfen setzen einige IND Entdeckungsalgorithmen – wie auch unser Binder Algorithmus – auf das simultane Testen mehrerer Kandidaten. Das von Binder vorgeschlagene Testverfahren ähnelt einem großen Hash-Join aller Attribute, wohingegen Konkurrenzalgorithmen wie Spider [BLN06] auf Sort-Merge-Joins setzen.

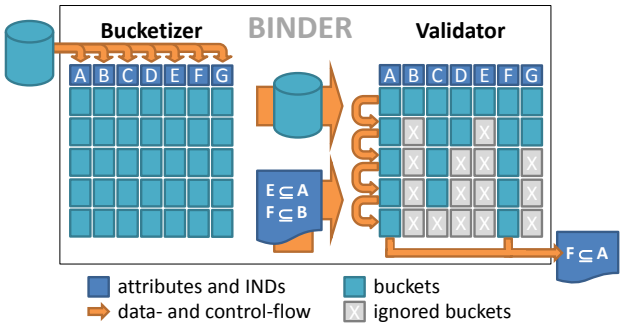


Abb. 4: Bucketierung und Validierung im Binder Algorithmus.

Solch simultane Validierungsverfahren für INDs sind recht effizient, allerdings ist auch der benötigte Speicherbedarf hoch. Damit der Algorithmus nicht – wie einige verwandte Verfahren – an Speichermangel scheitert, schlägt Binder einen Teile-und-Herrsche Ansatz für die Datenhandhabung vor: Wie in Abbildung 4 veranschaulicht wird der Datensatz zunächst Attribut-weise per Hash-Verfahren in Körbe aufgeteilt. Der Algorithmus entfernt dabei doppelte Werte in den Körben, um diese möglichst klein zu halten. Alle Körbe landen schließlich auf der Festplatte und werden dann zum Validieren der IND Kandidaten schrittweise wieder eingelesen. Wurde ein Attribut von allen IND Kandidaten entfernt, so müssen dessen Buckets in kommenden Validierungsschritten nicht mehr mitgelesen werden. Alle Kandidaten, die alle Validierungsschritte überstehen, sind gültige INDs und werden von Binder ausgegeben.

Unsere Experimente zum BINDER Algorithmus zeigen, dass dieser meist effizienter ist als seine Konkurrenz. Der wichtigste Beitrag besteht aber darin, dass der Algorithmus an keiner der bisher bekannten Grenzen scheitert: Er setzt nicht das Vorhandensein einer Datenbank voraus, er scheitert nicht an unzureichend großem Hauptspeicher und er erschöpft nicht das Maximum offener Dateizeiger (engl. file handles) eines Betriebssystems.

6 Das Metanome Data Profiling Werkzeug

Aufgrund der praktischen Relevanz des Data Profilings hat die Industrie verschiedene Profiling Werkzeuge entwickelt, die Informatiker in ihrer Suche nach Metadaten unterstützen sollen. Diese Werkzeuge bieten zwar eine gute Unterstützung für die Berechnung einfacher Statistiken, und sie sind auch in der Lage einzelne Abhängigkeiten zu validieren. Allerdings mangelt es ihnen an Funktionen zur echten *Entdeckung* von Metadaten. Um diese Lücke zu schließen schlagen wir das Werkzeug METANOME vor, eine erweiterbare Profiling Plattform, die nicht nur unsere eigenen Algorithmen, sondern auch viele weitere Algorithmen anderer Forscher integriert. Derzeit bieten wir 21 Entdeckungsalgorithmen für 9 verschiedene Arten von Metadaten an und arbeiten kontinuierlich mit Wissenschaftlern anderer Institute, darunter Institute in Frankreich, Italien, Kanada, Neuseeland, Indien, China und den USA, an weiteren Verfahren. Wir haben METANOME ebenfalls in Kooperation mit den Unternehmen FlixBus, IBM und SAP erfolgreich eingesetzt.

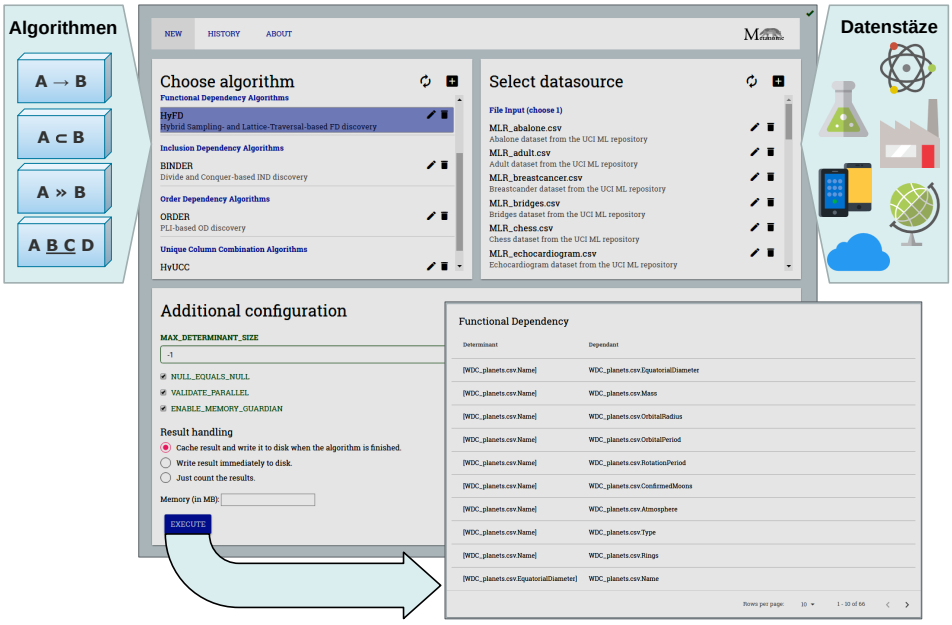


Abb. 5: Die Nutzersicht auf das METANOME Werkzeug mit dem Fenster zur Erstellung von Profiling-Läufen (Mitte) und einem Ergebnis-Fenster (Vorne).

Abbildung 5 zeigt die Nutzeroberfläche des Tools: Im linken Teil der Oberfläche werden die zu entdeckenden Metadaten über ihre Algorithmen ausgewählt, links daneben befindet sich der Import- und Auswahlbereich für Datensätze, und im unteren Teil können Profiling Prozesse konfiguriert und ausgeführt werden. Die Ergebnisse werden dann nach erfolgreicher Ausführung in einer weiteren Maske angezeigt. Auf diese Weise machen wir mit METANOME unsere Forschungsergebnisse für alle Informatiker und auch fachfremde Nutzer zugänglich. Neben der Metadaten-Entdeckung bietet die Plattform auch Unterstützung bei der Bewertung und Visualisierung gefundener Metadaten.

7 Metadaten getriebene Schema-Normalisierung

Unsere neuen Profiling Algorithmen ermöglichen die effiziente Entdeckung aller syntaktisch korrekten Metadaten auf realistisch großen Datenstzen. Dies führt nun zur Folgeaufgabe, aus den gefundenen Abhängigkeiten nur die für einen gegebenen Anwendungsfall semantisch bedeutsamen Teile zu extrahieren. Das Extrahieren bedeutsamer Metadaten aus allen findbaren Abhängigkeiten ist eine Herausforderung, da zum einen die Mengen der gefundenen Metadaten überraschenderweise groß sind (oft größer als der untersuchte Datensatz selbst) und zum anderen die Entscheidung über die Anwendungsfall-spezifische semantische Relevanz einzelner Abhängigkeiten schwierig ist.

Um zu zeigen, dass die Vollständigkeit der Metadaten sehr wertvoll für ihre Nutzung ist, veranschaulichen wir die effiziente Verarbeitung und effektive Bewertung von Schlüssel Abhängigkeiten am Anwendungsfall der Schema-Normalisierung: Das NORMALIZE Verfahren bewertet automatisch alle gefundenen Abhängigkeiten daraufhin, ob sie auch semantisch korrekte Schlüssel darstellen und strukturiert die zugehörige Tabelle anschließend entsprechend um. Wir haben NORMALIZE getestet, indem wir zunächst wohlgeformte Schemata denormalisiert haben, dann alle Schlüsselabhängigkeiten mit METANOME finden ließen und anhand dieser Abhängigkeiten schließlich ein normalisiertes Schema abgeleitet haben. Da das abgeleitete Schema dem ursprünglichen sehr ähnlich ist, schließen wir auf eine hohe Effektivität für den NORMALIZE Algorithmus.

Literaturverzeichnis

[Bl16a] Bleifuß, Tobias: Efficient Denial Constraint Discovery. Masterarbeit, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, 2016.

[Bl16b] Bleifuß, Tobias; Bülow, Susanne; Frohnhofen, Johannes; Risch, Julian; Wiese, Georg; Kruse, Sebastian; Papenbrock, Thorsten; Naumann, Felix: Approximate Discovery of Functional Dependencies for Large Datasets. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM). S. 1803–1812, 2016.

[BLN06] Bauckmann, Jana; Leser, Ulf; Naumann, Felix: Efficiently Computing Inclusion Dependencies for Schema Discovery. In: ICDE Workshops. S. 2, 2006.

[Dr16] Draeger, Tim: Multivalued Dependency Detection. Masterarbeit, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, 2016.

[Dr18] Draeger, Tim: Efficient Discovery of Matching Dependencies. Masterarbeit, Hasso-Plattner-Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, 2018.

[FS99] Flach, Peter A; Savnik, Iztok: Database dependency discovery: a machine learning approach. AI Communications, 12(3):139–160, 1999.

[Hu99] Huhtala, Ykä; Kärkkäinen, Juha; Porkka, Pasi; Toivonen, Hannu: TANE: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal, 42(2):100–111, 1999.

[Li12] Liu, Jixue; Li, Jiuyong; Liu, Chengfei; Chen, Yongfeng: Discover Dependencies from Data – A Review. IEEE Transactions on Knowledge and Data Engineering (TKDE), 24(2):251–264, 2012.

[Pa15a] Papenbrock, Thorsten; Bergmann, Tanja; Finke, Moritz; Zwiener, Jakob; Naumann, Felix: Data Profiling with Metanome. Proceedings of the VLDB Endowment, 8(12):1860–1863, 2015.

[Pa15b] Papenbrock, Thorsten; Ehrlich, Jens; Marten, Jannik; Neubert, Tommy; Rudolph, Jan-Peer; Schönberg, Martin; Zwiener, Jakob; Naumann, Felix: Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. Proceedings of the VLDB Endowment, 8(10):1082–1093, 2015.

[Pa15c] Papenbrock, Thorsten; Kruse, Sebastian; Quiané-Ruiz, Jorge-Arnulfo; Naumann, Felix: Divide & Conquer-based Inclusion Dependency Discovery. Proceedings of the VLDB Endowment, 8(7):774–785, 2015.

[Pa17] Papenbrock, Thorsten: Data Profiling - Efficient Discovery of Dependencies. doctoralthesis, Universität Potsdam, 2017.

[PN16] Papenbrock, Thorsten; Naumann, Felix: A Hybrid Approach to Functional Dependency Discovery. In: Proceedings of the International Conference on Management of Data (SIGMOD). S. 821–833, 2016.

[PN17] Papenbrock, Thorsten; Naumann, Felix: A Hybrid Approach for Efficient Unique Column Combination Discovery. In: Proceedings of the Conference Datenbanksysteme in Büro, Technik und Wissenschaft (BTW). S. 195–204, 2017.

Thorsten Papenbrock wurde 1987 in Mettingen geboren und erlangte dort im Juni 2007 das Abitur am Kardinal-von-Galen-Gymnasium. Anschließend begann er sein Informatik Studium am Hasso-Plattner-Institut an der Universität Potsdam. Er erhielt den Bachelor-Abschluss im Sommer 2010 und den Master Abschluss im Winter 2013. Während seiner Studienzeit absolvierte er Praktika bei der SAP AG in Waldorf (2009; zwei Monate), BBF GmbH in München (2010; zwei Monate) und SAP AG in Belfast (2011; 6 Monate). Ab April 2013 arbeitete er als wissenschaftlicher Mitarbeiter bei Prof. Naumann am Lehrstuhl für Informationssysteme. Neben seiner Forschungstätigkeit, die zu den hier beschriebenen Ergebnissen führte, hat er sich auch in der Lehre in nunmehr 29 Veranstaltungen engagiert. Seit der Einreichung seiner Doktorarbeit im Juni 2017 forscht und lehrt er als Postdoc am Hasso-Plattner-Institut. Neben seinem Forschungsinteresse in den Bereichen Data Profiling, Datenreinigung und verteiltes Rechnen ist er begeisterter Schwimmer und Jogger.