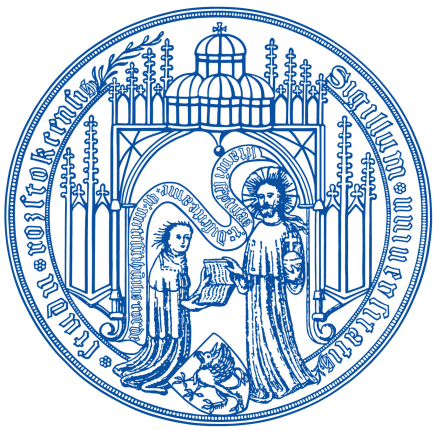
Die Graham-Reduktion als Alternative zur Berechnung von Verbundpfaden bei der Anfragerekonstruktion

Masterarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von:	Hauke Schur
Matrikelnummer:	213204169
geboren am:	10.02.1995 in Bützow
Erstgutachter:	Prof. Dr. Andreas Heuer
Zweitgutachter:	apl. Prof. Dr.-Ing. Meike Klettke
Betreuer:	Dr.-Ing. Holger Meyer
Abgabedatum:	27. Mai 2019

Abstract

In aktuellen Forschungen zum Reverse Engineering von komplexen Anfragen wurden neue Verfahren vorgestellt, welche zur Verbundpfaderstellung genutzt werden. Ein Beispiel dafür ist Paleo, welches speziell für die Rekonstruktion von Top-K-Anfragen entwickelt wurde. In der Veröffentlichung wurde auf einen Vergleich mit klassischen Verfahren, welche beispielsweise für die Anfrageninterpretation von Universalanfragen genutzt wurden, verzichtet. In dieser Masterarbeit wurde ein eigenes Tool entwickelt, welches auf der Graham-Reduktion von Hypergraphen basiert. Diese wurde gezielt für Universalrelationen entwickelt, weshalb gewisse Modifikationen nötig sind, damit ein allgemeiner Einsatz ermöglicht wird. Auch haben Zyklen negative Effekte, welche die Verbundpfade verfälschen. Es wurden mehrere Modifikationen untersucht, um den eigenen Ansatz auch auf beliebigen zyklischen Datenbankschemata durchführen zu können. Abschließend wurden die Ergebnisse des eigenen Tools mit denen von Paleo verglichen. Dafür wurden selbst entwickelte Anfragen und der TPC-H-Benchmark genutzt.

Recent research on reverse engineering of complex queries has introduced new techniques that are used to create join paths. An example of this is Paleo, which was developed specifically for the reconstruction of top-k queries. However, no comparison was made with classical methods used to interpret universal relation queries. In this thesis, a tool was developed which is based on the Graham reduction of hypergraphs. It was developed specifically for universal relations, which is why certain modifications are necessary in order to enable general use. Cycles also have negative effects on join paths. Several modifications have been investigated in order to be able to apply this approach to any cyclic database schema. In order to compare the created join paths with those of Paleo, self-developed queries and the TPC-H benchmark were used.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Top-K-Anfragen sind ein elementares Werkzeug von Informationssystemen, um Daten aufzubereiten und leichter zu präsentieren. Die Ergebnisse sind Ranglisten, welche nur eine begrenzte Anzahl an Einträgen haben. Es gibt sie in vielen Alltagssituationen und sie kommen, neben Suchmaschinen, auch in vielen anderen Bereichen zum Einsatz. Diese Ranglisten sind allerdings nur im Kontext ihrer Anfrage aussagekräftig. Es gibt jedoch mehrere Szenarien, in denen eine Liste vorliegt, aber die Anfrage fehlt oder nur teilweise bekannt ist. Das macht eine exakte Überprüfung oder Wiederholung der Anfrage unmöglich. Das ist aber zum Beispiel bei wissenschaftlichen Arbeiten gewünscht, um die Ergebnisse zu verifizieren. In manchen Fällen geht die Anfrage auch verloren. Das kann verhindern, dass neue Auswertungen erstellt oder mit den alten verglichen werden können. Im Zweifel sind die alten Listen damit unbrauchbar. Ein anderes Beispiel ist, dass nach Anfragen gesucht wird, welche die selben Ergebnisse liefern, sich aber von der originalen unterscheiden. Diese können schneller sein oder unbekannte Zusammenhänge in den Daten offenbaren.
Aktuelle Forschungen versuchen, nur mit der Ergebnisliste und der zugrundeliegenden Datenbank die Anfrage zu rekonstruieren. Ein Beispiel dafür ist Paleo, welches in [PWM17] vorgestellt wurde. Ein wichtiges Teilproblem dieser Arbeit ist die Erstellung von Verbundpfaden. Dafür wurde das Paleo-J Framework entwickelt, welches in [PM16] vorgestellt wurde. Der Ansatz leitet sich aus aktueller Forschung zum allgemeinen Reverse Engineering von Anfragen ab ([ZEPS13]). Leider wurde die Arbeit nicht mit klassischen Ansätzen verglichen.
In dieser Masterarbeit wird das Paleo-J Framework untersucht und mit der Graham-Reduktion von Hypergraphen verglichen. Letzteres ist ein Verfahren, welches zur Verbundpfaderstellung in Anfragesprachen von Universalrelationen verwendet wurde. Wie auch andere klassische Ansätze, ist die Graham-Reduktion nur für bestimmte Datenbankschemata gedacht. Deshalb werden Methoden entwickelt, welche diese Beschränkungen aufheben sollen. Es wird eigenständig ein Tool implementiert, welches mit Paleo-J verglichen wird. Dazu wird das Datenbankschema des TPC-H-Benchmarks verwendet. Der TPC-H-Benchmark wurde gezielt dafür entwickelt, um Datenbanken mit praxisnahen Anfragen

und Daten zu testen. Es handelt sich dabei um ein gutes Beispiel, da dieses Schema normalerweise nicht von der Graham-Reduktion verarbeitet werden kann. Zusätzlich wurden mehrere Anfragen entwickelt, welche gezielt die Stärken und Schwächen der beiden Ansätze aufzeigen sollen.

Die nachfolgende Arbeit ist in sechs Kapitel gegliedert. In Kapitel 2 werden verschiedene Grundlagen erläutert. Es gibt eine kleine Einführung in die URSA und Top-K-Anfragen. Für das Verständnis der Graham-Reduktion werden Hypergraphen definiert und Zyklen vorgestellt. Außerdem werden mit dem join tree und Tableaus vergleichbare klassische Ansätze vorgestellt. Diese können ebenso wie die Graham-Reduktion Verbundpfade erstellen und gleichen sich in ihren Merkmalen.
Im darauffolgenden Kapitel werden das Paleo-J Framework und die Graham-Reduktion erläutert. Letztere kann mittels einer Modifikation auch dazu genutzt werden, Verbundpfade zu erstellen. Beide Techniken werden mit Beispielen erklärt. Außerdem wird der für die Beispiele und den Vergleich verwendete TPC-H-Benchmark kurz vorgestellt.
Kapitel 4 enthält die Planung und den Entwurf eines eigenen Ansatzes basierend auf der Graham-Reduktion. Dafür werden die Unterschiede zum Paleo-J-Ansatz ermittelt und die Nachteile, welche daraus resultieren. Es werden Lösungsansätze präsentiert, mit denen die Graham-Reduktion auch für allgemeine Datenbankschemata eingesetzt werden kann.
In Kapitel 5 wird die eigene Implementierung dieser Lösungen vorgestellt. Dazu werden Ausschnitte des Java-Programmcodes vom entwickelten Prototypen verwendet.
Dieser wird in Kapitel 6 mit einer Version des Paleo Frameworks verglichen. Dazu wurde das TPC-H-Datenbankschema verwendet. Außerdem wurden eigene Anfragen erstellt, welche die Vor- und Nachteile beider Systeme aufzeigen sollen.
Im letzten Kapitel werden die Erkenntnisse zusammengefasst und ein abschließender Ausblick gegeben. Dieser enthält ungelöste Probleme und mögliche Verbesserungen.

Kapitel 2

Grundlegende Konzepte

Ziel dieser Arbeit ist es, einen alternativen Ansatz für das in [PWM17] vorgestellte System zu entwickeln. Dafür wird, statt des dort eingesetzten limitieren Suchbaums, eine bekannte, auf Hypergraphen basierende Technik eingesetzt. Die Grundlagen und weitere Ansätze werden in diesem Kapitel vorgestellt.

2.1 Relationale Datenbanken

Es wird vorausgesetzt, dass eine grundlegende Kenntnis im Bereich der relationalen Datenbanken vorhanden ist. Von Bedeutung sind vor allem Datenbankschema, Relation, Attribut, sowie die Grundlagen der relationalen Algebra. Diese werden ausführlich in [Mai83] erläutert.

Universalrelation

Bei einer Universalrelation oder auch *universal relation scheme assumption* (URSA), wie in [Mai83] beschrieben, handelt es sich um ein Datenbankschema, bei der alle Relationen die Projektionen einer einzigen Relation sind. Die direkte Folge ist, dass Attribute global eindeutig sein müssen. Haben sie den selben Namen, dann haben sie auch die gleiche Bedeutung.
Auf dieser Annahme wurden Anfragesprachen entwickelt, welche auch von unerfahrenen Nutzern eingesetzt werden sollen. Beispiele dafür wurden in [MHH93] vorgestellt. Ziel ist es, dass der Nutzer nur angeben soll, welche Attribute ihn interessieren und welche Filter genutzt werden. Damit die Anfrage bearbeitet werden kann, wird ermittelt, welche Relationen benötigt und wie diese miteinander verbunden werden müssen. Das soll vollautomatisch ausgeführt werden. Der Nutzer benötigt also kein Wissen über die Struktur der Datenbank oder wie die Anfrage mit dieser erstellt wird. Ein Ansatz für so eine Automatisierung ist eine Modifikation der Graham-Reduktion, welche in dieser Arbeit vorgestellt und als alternativer Ansatz für das Paleo-J-System evaluiert werden soll.

2.2 Top-K-Anfragen

Bei einer Top-K-Anfrage handelt es sich um ein Datenbankanfrage, bei der nur eine bestimmte Anzahl an Ergebnistupeln von Interesse ist. Das Ergebnis sind k Tupel, welche den höchsten bzw. niedrigsten Ranking-Score erreichen. Sie werden sehr häufig bei Suchanfragen verwendet, sind aber auch von großer Bedeutung bei statistischen Auswertungen von Daten. Die Tupel sind meist zweiteilig, bestehend aus einem identifizierenden und dem Ranking-Wert. Dementsprechend ist der Informationsgehalt niedrig und meist nur im Kontext der Anfrage informativ. Häufige Ranking-Kriterien sind MIN, MAX, AVG, SUM oder COUNT. Es gibt aber auch komplexere Auswertungen, welche erweiterte Methoden verwenden.

Beispiel 1.

```
SELECT
  n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM    ...
WHERE   ...
group by  n_name
order by  revenue desc limit 5;
```

Die gezeigte Anfrage wurde dem TPC-H-Benchmark [TPC] entnommen. Sie soll eine typische Top-K-Anfrage darstellen. Die Ausgabe ist ein Ranking der Nationen mit den höchsten Einnahmen. □

An dem Beispiel lässt sich die Form einer Top-K-Anfrage gut erkennen. So werden bei der Projektion meist ein identifizierendes Attribut und ein Aggregat als Ranking verwendet. Dabei zeigt dieses Beispiel schon ein Problem des Reverse Engineering der Anfragen. Das Aggregat berechnet nicht eine einfache Summe, sondern das Produkt von zwei Attributen. Handelt es sich nicht um eine Funktion eines einzelnen Wertes, wird es sehr schwer, die vielen verschiedenen mathematischen Ausdrücke zu überprüfen. Da sich diese Arbeit vorrangig mit den Verbundpfaden beschäftigt, sind vor allem die Probleme relevant, die sich mit der Auswahl der Relationen ergeben. So muss eine Menge von Relationen gefunden werden, welche das identifizierende und die Aggregat-Attribute miteinander verbinden. Es können aber auch andere Relationen am Verbund teilnehmen. Diese werden für den eigentlichen Verbund nicht benötigt, sondern sind Relationen die einen sogenannten Filter enthalten oder diese mit dem direkten Verbundpfad verbinden. Diese Filterrelationen sind nur für die Selektion notwendig.

2.3 Hypergraphen und Zyklen

Um die Graham-Reduktion erklären zu können, ist das Wissen über Hypergraphen und ihre Zyklen wichtig. Nach [Mai83] ist ein Hypergraph ein ungerichteter Graph, dessen Kanten (auch Hyperkanten

genannt) aus einer beliebigen, nicht leeren Menge von Knoten besteht. Statt nur zwei, kann eine solche Hyperkante also einen oder mehr Knoten enthalten. Diese Darstellung kann verwendet werden, um ein Datenbankschema zu beschreiben. Dazu werden alle Attribute zu Knoten des Hypergraphen und die einzelnen Relationenschemata werden als Kanten dargestellt. Mit verschiedenen Methoden können ein Hypergraph modifiziert und gewisse Eigenschaften nachgewiesen werden. Diese lassen Rückschlüsse auf die Eigenschaften das Datenbankschema zu. Hypergraphen werden in dieser Arbeit nur für die Untersuchung der Datenbankschemata eingesetzt und deshalb werden Attribut und Knoten, sowie Kante und Relation, häufig als Synonyme verwendet. Da in allen anderen Kapiteln hauptsächlich Hypergraphen behandelt werden, wird Graph häufig als Synonym für Hypergraph verwendet.
Zunächst werden grundlegende Definitionen aus [Mai83] für Hypergraphen benötigt:

Definition 2.3.1. Ein Hypergraph H ist ein Paar $\{\mathfrak{N}, \mathfrak{E}\}$. $\mathfrak{N}$ ist die Menge der Knoten und $\mathfrak{E}$ die Menge der Kanten oder auch Hyperkanten, welche je eine nichtleere Teilmenge von $\mathfrak{N}$ sind.

Definition 2.3.2. H ist *reduziert*, wenn keine Kante aus $\mathfrak{E}$ eine Teilmenge einer anderen und jeder Knoten in einer Kante enthalten ist. Die Reduktion von H, geschrieben $RED(H)$, ist H ohne die Einzel-Knoten und Teil-Kanten.

Definition 2.3.3. A und B sind Knoten aus $\mathfrak{N}$. Ein *Pfad* zwischen A und B ist eine Sequenz von Kanten $E_1, E_2 ... E_k, k \leq 1$, mit $A \in E_1$, $B \in E_k$ und $E_i \cap E_{i+1} \neq \emptyset$ für $1 \leq i < k$. Diese Sequenz ist außerdem ein Pfad zwischen den Kanten E_1 und E_k. Zwei Knoten oder Kanten sind *verbunden*, wenn es einen Pfad zwischen ihnen gibt.

Definition 2.3.4. Ein $\mathfrak{M}$-*induzierter* Hypergraph $H_{\mathfrak{M}}$ von H ist der Hypergraph $RED(\{\mathfrak{M}, \mathfrak{E}_{\mathfrak{M}}\})$, mit $\mathfrak{M} \subset \mathfrak{N}$ und $\mathfrak{E}_{\mathfrak{M}} = \{E \cap \mathfrak{M} | E \in \mathfrak{E}\}$.

Für die α-Azyklität werden noch die folgenden Definitionen aus [Mai83] benötigt:

Definition 2.3.5. Eine Menge $F \subseteq \mathfrak{N}$ ist ein *articulation set* für H, mit $F = E_1 \cap E_2$ für beliebige Kanten $E_1, E_2 \in \mathfrak{E}$, wenn es Kanten gibt, die in H und nicht in $H_{\mathfrak{N}-F}$ verbunden sind. Das Entfernen der Knoten aus F trennt manche Knoten, welche vorher miteinander verbunden waren.

Definition 2.3.6. Ein *block* ist ein induzierter Teilgraph, welcher keine *articulation sets* besitzt.

Das *articulation set* ist also die einzige Verbindung zwischen diesen Kanten. Ist in einem induzierten Teilgraph also kein *articulation set*, muss es nach dem Entfernen der gemeinsamen Knoten weiterhin einen Pfad zwischen den zwei Kanten geben. Ein *block* ist also ein Hypergraph, bei dem zwischen alle Kanten auch nach dem Entfernen noch ein Pfad besteht. Ein *block* stellt also eine Art Zyklus dar.

Definition 2.3.7. Ein Hypergraph und seine reduzierte Form sind *azyklisch*, wenn die reduzierte Form keine *blocks* enthält.

Es ist zu beachten, dass in [Fag83] unterschiedliche Arten der Azyklität von Hypergraphen vorgestellt wurden. Die hier definierte Azyklität entspricht der α-Azyklität. Diese ist die schwächste Form. Durch die Definitionen ergeben sich azyklische Hypergraphen, welche nicht intuitiv als solche aufgefasst werden.

Beispiel 2. Ein zyklischer Hypergraph kann durch Hinzufügen weiterer Kanten azyklisch werden. Ein sehr kurzes Beispiel dafür wurde unter anderem in [Fag83] gezeigt. In der Abbildung 2.1a ist der Hypergraph des Datenbankschemas *{ABC,CDE,AFE}* zu sehen. Hier ist der Zyklus klar zu erkennen. Die Kanten *ABC* und *CDE* sind nach dem Entfernen von *C* weiterhin über *AFE* miteinander verbunden. Somit hat dieser Hypergraph einen *block* und ist zyklisch.

(a) {ABC, CDE, AFE} (b) {ABC, CDE, AFE, ACE}

Abbildung 2.1: Hypergraphen für ein zyklisches und ein azyklisches Schema

Wird dieses Datenbankschema jedoch erweitert auf *{ABC, CDE, AFE, ACE}* ist es nicht länger zyklisch. Die *ACE*-Kante hat je zwei gemeinsame Knoten mit allen anderen Kanten. Damit der Zyklus erhalten bleibt, müssen mindestens die Knoten *A, C* und *E* im induzierten Teilgraph enthalten sein. Also muss die *ACE* Kante auch enthalten sein. Durch das Entfernen der gemeinsamen Knoten mit der Kante *ACE*, bleiben jeweils *B,D* und *F*, welche nicht mehr mit dem Restgraphen verbunden sind. Sollten *B, D* und *F* nicht enthalten sein, so sind ihre Kanten eine Teilmenge der *ACE* Kante und werden entfernt. Dadurch gibt es mehrere *articulation sets*. Es gibt keinen induzierten Teilgraphen, welcher einen *block* enthält. □

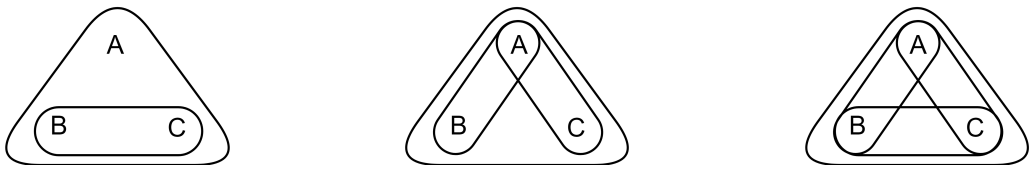
Das Beispiel entspricht einer anderen Form von Zyklus, welche in [Fag83] definiert werden. Die α-Azyklität des Hypergraphen eines Datenbankschemas ist jedoch ausreichend, um gewisse Eigenschaften des Datenbankschemas zu garantieren. Nach [MSH17] ist die α-Azyklität äquivalent zu:

1. Die Graham-Reduktion ist erfolgreich
2. R hat einen join tree
3. R hat einen full reducer
4. Die Reduktion von R ist eine 4NF-Dekomposition

Diese und noch weitere äquivalente Eigenschaften sind von Interesse, da sie die Arbeit mit dem Datenbankschema erleichtern. Ein *full reducer* wird eingesetzt, um bei einem Verbund eine Vorauswahl von Tupeln zu treffen. Diese Vorauswahl enthält nur Tupel, welche für den Verbund relevant sind. Das ist vor allem dann hilfreich, wenn die Übertragung von Tupeln langsam ist, weil die Daten an unterschiedlichen Orten sind.

Andere Arten von Zyklen

Neben den beschriebenen α-Zyklen wurden auch andere Arten definiert. Diese sind für diese Arbeit von Bedeutung. In [Bra16] wurde ein graphisches Beispiel angeführt, um diese leichter zu vergleichen. Die Abbildung 2.2 zeigt Teile dieses Beispiels. Die Betrachtung der anderen Zyklen eignen sich nur bedingt im Zusammenhang von Hypergraphen und Datenbankschemata. Da diese dennoch einen Einfluss auf die Verbundpfaderstellung der Graham-Reduktion haben, werden diese, ohne auf die genauen Definitionen einzugehen, kurz vorgestellt. Die Abbildung lässt auch die Zusammenhänge zwischen den Zyklen erkennen, welche in [Fag83] gezeigt wurden.



(a) berge-zyklisch, γ-azyklisch (b) γ-zyklisch, β-azyklisch (c) β-zyklisch, α-azyklisch

Abbildung 2.2: Unterschiedliche Arten von Zyklen

In der Abbildung 2.1b handelt es sich um einen β-Zyklus. Dieser besteht, wenn durch Entfernen von Kanten ein α-Zyklus entsteht. Ein γ-Zyklus ist vorhanden, wenn mindestens drei Kanten paarweise mindestens einen gemeinsamen Knoten haben. Der Berge-Zyklus besteht besteht schon, wenn zwei Kanten zwei Knoten teilen. Dieser Fall ist in Datenbanken häufiger vorhanden, wenn beispielsweise ein Schlüssel aus zwei Attributen besteht und diese eigentlich als eine Einheit angesehen werden müssen. Im Gegensatz zum α-Zyklus ist die Kenntnis über das Vorhandensein dieser Zyklen weniger hilfreich für einen Datenbankentwurf.

2.4 Join Tree und Graph

Der Ansatz des Paleo-J Frameworks ähnelt dem des *join graph*. In diesem werden Schlüsselbeziehungen zwischen Relationen mittels Kanten dargestellt. Dabei verwenden sie jedoch abgewandelte Definitionen. Vor allem die des *join tree* hat kaum noch eine Ähnlichkeit zu der aus [Mai83]. Letzterer definiert einen *join tree* so, dass mit ihnen leicht Verbundpfade gefunden werden können. Ein *join graph* kann in einer ähnlichen Weise wie die Graham-Reduktion zur Erstellung von Verbundpfaden genutzt werden. Ein in

[YO79] vorgestellter Algorithmus verwendet sogar die gleichen Schritte wie die Graham-Reduktion, nur mit *join graph* als Grundlage. Da diese Technik dem gewählten Ansatz stark ähnelt, wird sie kurz vorgestellt.

Definition 2.4.1. Ein *complete intersection graph* I_R für ein Datenbankschema R ist ein unidirektionaler Graph, dessen Knoten je eine Relation repräsentieren. Eine Kante zwischen zwei Knoten besteht, wenn die Relationen eine gemeinsame Menge von Attributen haben. Die Kanten werden mit diesen Attributen beschriftet.

Definition 2.4.2. Ein *intersection graph* ist ein Teilgraph eines I_R, der nur durch Entfernen von Kanten erzeugt wurde.

Definition 2.4.3. Ein *A-Pfad* für ein Attribut A beschreibt einen Pfad zwischen zwei Knoten, bei dem jede Kante mit dem Attribut A beschriftet wurde.

Definition 2.4.4. Ein *join graph* für ein Datenbankschema ist ein *intersection graph*, für den gilt, dass es für je zwei Knoten für jedes gemeinsame Attribut A einen A-Pfad gibt.

Abbildung 2.3: *join graph* des Schemas {ABC, CDE, ACE, AFE}

Beispiel 3. In Abbildung 2.3a wird der *complete intersection graph* für das Datenbankschema *{ABC, CDE, ACE, AFE}* gezeigt. Jede Relation, welche gemeinsame Attribute mit einer anderen hat, besitzt auch eine Kante mit dieser. Der in Abbildung 2.3b dargestellte *join graph* gehört zu dem selben Schema, hat aber weniger Kanten. Dennoch gibt es immer einen Pfad zwischen den Relationen mit einem gemeinsamen Attribut. Beispielsweise hat *ABC* einen A-Pfad zu *AFE* über *ACE*. □

Die Definition des *join graph* entspricht etwa der des Paleo-J Frameworks. Aus diesem Graphen können direkt die Attribute abgelesen werden, welche für den Verbund zweier Relationen nötigt ist. Sind mehr Relationen beteiligt, muss ein Pfad gesucht werden, welcher alle Relationen enthält. Die Suche danach wird jedoch erschwert, da es in einem Graphen potentiell viele unterschiedliche Pfade geben kann.

Definition 2.4.5. Ein *join tree* ist ein *join graph*, welcher eine Baumstruktur besitzt.

(a) {ABC, CDE, ACE, AFE} (b) {ABC, CDE, AFE}

Abbildung 2.4: *join graph* für ein zyklisches und ein azyklisches Schema

Beispiel 4. Die Abbildung 2.4a zeigt einen weiteren *join tree* des Schemas *{ABC, CDE, ACE, AFE}*. Dieser weist eine Baumstruktur auf und somit hat das Schema einen *join tree*. Der *join graph* des Schemas *{ABC, CDE, AFE}* (Abbildung 2.4b) kann nicht in eine Baumstruktur umgewandelt werden, weil die Knoten sonst keinen A-Pfad mehr besitzen. □

Bei einem *join tree* hingegen handelt es sich um einen Baum, welcher nur einen direkten Pfad zwischen zwei Knoten besitzt. Dieser kann dann leicht in einen Verbundausdruck übersetzt werden. Für eine Anfrage sind also nur die gesuchten Relationen wichtig, da der Verbund schnell berechnet werden kann. Der in [YO79] vorgestellte Algorithmus ähnelt der Graham-Reduktion so stark, dass häufig der gemeinsame Name GYO-Reduktion verwendet wird. Der grundlegende Unterschied ist, dass das Ergebnis der erfolgreichen Reduktion ein *join tree* ist. Aufgrund der Äquivalenz der Existenz eines *join tree* und dem Erfolg der Graham-Reduktion gibt es also einen direkten Zusammenhang zwischen den Verfahren. Diese Arbeit beschränkt sich jedoch auf die Graham-Reduktion mit Hypergraphen.

2.5 Tableaus

Eine weitere Alternative zur Graham-Reduktion, welcher mit Hypergraphen umgesetzt werden kann, ist die Reduktion mithilfe von Tableaus. Die Grundlagen dazu können in [Mai83] gefunden werden. Kurz zusammengefasst wird eine Tabelle erstellt, die für jedes Attribut eine Spalte besitzt. Dann wird pro Relation eine Zeile erstellt und die Zellen mit ihren Attributen erhalten sogenannte ausgezeichnete Variablen. Die restlichen Zellen werden mit nicht ausgezeichneten Variablen aufgefüllt. Um mit dieser Tabelle einen Verbundpfad zu finden, werden alle ausgezeichneten Variablen, die nicht Teil der Projektion sind, zu nicht ausgezeichneten. Dann werden mittels Tableau-Abbildungen, auch Homomorphismen genannt, wie in [Mai83] gezeigt, die Zeilen entfernt, welche nicht an einem Verbund beteiligt sind. Zeilen, die für diesen benötigt werden, können nicht entfernt werden, da diese nicht auf eine andere Zeile abgebildet werden können. Da diese Tabellen viel Platz in Anspruch nehmen, wird im Anhang A das Beispiel 7 mit dieser Technik durchgeführt. Dadurch werden die Ähnlichkeiten zur Graham-Reduktion ersichtlich. Dieser Zusammenhang wurde in [MU84] untersucht. Dort wurde bewiesen, dass das Ergebnis des

Tableau-Ansatzes dem der in dieser Arbeit genutzten Modifikation der Graham-Reduktion entspricht. Dies gilt jedoch nur bei azyklischen Hypergraphen.

Kapitel 3

Stand der Technik

Das Auffinden von Verbundpfaden nur mit Hilfe von vorgegebenen Attributen ist nichts Neues. Wie in [MHH93] gezeigt, gibt es unterschiedliche Verfahren, die dieses Problem lösen sollen. Diese sind meist auf eine erleichterte Nutzerinteraktion ausgelegt und eigentlich konzipiert, um genau eine Anfrage zu erzeugen. Das in dieser Arbeit beschriebene Problem des Reverse Engineering von Top-K-Anfragen erfordert jedoch einen anderen Ansatz. Deshalb lassen sich die bekannten Techniken nur bedingt darauf anwenden. In diesem Kapitel wird das Paleo Framework kurz und das Paleo-J Framework ausführlicher vorgestellt. Als alternativer Ansatz wird die Graham-Reduktion weiter erläutert. Mit dem TPC-H-Benchmark und dessen Datenbankschema und Anfragen wird eine Referenz aufgezeigt, welche für Beispiele und den späteren Vergleich verwendet wird.

3.1 Paleo Framework

Das Paleo [PM16] Framework wurde entwickelt, um mittels Reverse Engineering eine Top-K-Anfrage zu rekonstruieren. Dafür nötig ist das Ergebnis der Anfrage sowie die Daten zur Zeit der Anfrage. Das Ergebnis muss hierfür in Form einer binären Liste vorliegen, mit einem Identifikationsmerkmal und dem jeweiligen Score. Das Problem besteht aus mehreren Teilen. Einerseits muss die korrekte Projektion, also das Ranking-Kriterium, gefunden werden. Außerdem werden mögliche Selektionsbedingungen gesucht, damit die korrekten Tupel genutzt werden. Dann werden verschiedene Anfragen ausgeführt und ihr Ergebnis evaluiert. Da solche Anfragen meist eine zeitintensive Ausführung haben, werden verschiedene Optimierungen eingesetzt, welche die komplexen Anfragen mit bereits ausgeführten vergleichen und ihre Ergebnisse schätzen. Diese Schätzungen werden wiederum genutzt, um zuerst die vielversprechenden Anfragen auszuführen. Es wurde auch ein probabilistisches Modell entwickelt, welches nur ein Teil der originalen Daten benötigt. Für diese Arbeit ist vor allem Paleo-J [PWM17] von Interesse. In diesem wird nach allen möglichen Verbundpfaden gesucht. Der Ansatz basiert auf [ZEPS13]. Die Grundlage

ist eine Breitensuche über den Schemagraphen. Mittels der Fremdschlüsselbeziehungen wird nach allen möglichen Verbunden gesucht.
Das Ergebnis des Frameworks ist eine *select-project-join query*. Diese folgt immer dem vorgegebenen Template mit der Form:

```
SELECT L.e, agg(value)
FROM R1, R2,...
WHERE P1 and P2...
GROUP BY L.e
ORDER BY agg(value) DESC LIMIT k
```

An diesem Template können gut die notwendigen Schritte abgeleitet werden. Zuerst werden Attribute für die Identifikation (*L.e*) und Ranking (*value*) gesucht. Zusätzlich werden Filter benötigt (*P1...*). Der für diese Arbeit wichtigste Teil ist jedoch das Finden der Relationen (*R1, R2,...*), welche für die Verbunde nötig sind. Das Paleo-J Framework nutzt für die Suche nach den Verbundprädikaten die Schritte:

1. Schema Exploration
Das Datenbankschema wird analysiert und es wird nach Entity- und Ranking-Attributen gesucht. Für das Entity-Attribut kann meist einfach geprüft werden, ob die Werte aus der Ergebnisliste mit in den Attributwerten enthalten sind. Bei den Ranking- und Filter-Attributen hingegen wird auf die Domäne der Attribute geachtet, da diese meist nicht dem Ranking-Wert entsprechen. Außerdem werden alle Primär- (PK) und Fremdschlüssel (FK) Beziehungen untersucht, welche für den Aufbau der *join trees* genutzt wird.

2. Building Join Trees
Ausgehend von den Informationen aus Schritt 1 wird ein *join tree* erzeugt. Dieser entspricht nicht der Definition 2.4.5. Diese *join trees* werden ausgehend von einer Relation mit möglichen Enitity-Attributen aufgebaut. Für jeden Knoten werden dafür alle Relationen, welche mit der Relation des Knotens eine PK/FK-Verknüpfung haben, als Child-Knoten hinzugefügt. Für jeden Knoten, außer der Wurzel und den Blatt-Knoten, ist der Parent-Knoten auch wieder ein Child-Knoten. Durch diese Mehrfachverwendung von Relationen werden die Namen mit Indizes markiert. Die Erstellung des Baumes wird ab einer festgelegten Tiefe abgebrochen. Danach werden alle Knoten markiert, welche als Ranking-Kriterium in Frage kommen.

3. Building Join Chains
Aus dem *join tree* werden ausgehend von jedem Knoten sogenannte *join chains* erstellt. Dieser lineare *query graph* wird erzeugt, indem vom Knoten aus alle Parent-Knoten bis zur Wurzel hinzugefügt werden. Durch die Art der Erstellung des *join trees* gleichen sich manche der *join chains*.

4. Building Lattices
Diese *join chains* stellen schon Verbundpfade dar. Diese sind die Ausgangsmenge der *lattices*. Durch das schrittweise Zusammenfassen von Knoten, welche die gleiche Relation referenzieren, werden weitere erstellt. Dabei bleiben die Indizes erhalten. So werden beispielsweise C1 und C2 zu C12. Hatte der *join tree* die richtige Tiefe, ist der Verbund der Top-K-Anfrage in den *lattices* vorhanden.

5. Query Building
Aus den *lattices* können SQL-Anfragen erstellt werden. Das ermittelte Entity-Attribut wird als *L.e* verwendet und ein markierter Knoten für die Aggregatfunktion genutzt. Es werden alle Relationen des *lattice* verwendet. Die Verbundkriterien können aus den Kanten zwischen den Knoten abgeleitet werden. Die Selektion wird erst in einem späteren Schritt hinzugefügt, da aufgrund der Komplexität erst überprüft werden muss, welche Anfragen plausibel sind. Jede Anfrage erhält einen Kosten-Wert, welcher aus der Menge der Tupel und der Kardinalität der Verbundattribute bestimmt wird.

6. Instance Verification
In diesem Schritt werden die erstellten Anfragen ausgeführt und gegen die Liste geprüft. Um effizient zu arbeiten, wird eine Vorauswahl getroffen. Alle Anfragen, welche auch mit Filter nicht der Ergebnisliste entsprechen, werden entfernt. Alle potentiell gültigen Anfragen erhalten wieder eine Bewertung und werden dann vom Rest des Paleo Frameworks weiterverarbeitet.

Beispiel 5. Damit das Vorgehen besser verständlich ist, wird es an einem theoretischen Beispiel verdeutlicht. Es wurde eine Top-K-Anfrage an die Datenbank mit dem Schema 3.7 gestellt. Gesucht wurden die zehn Kunden, welche am meisten Geld ausgegeben haben. Das entspricht der SQL-Anfrage:

```
SELECT c_name, SUM(o_totalprice)
FROM customer, orders
WHERE c_custkey = o_custkey
GROUP BY c_name
ORDER BY SUM(o_totalprice)
DESC LIMIT 10
```

Im ersten Schritt werden alle möglichen Ranking-Kandidaten gefunden. Dazu gehören alle Attribute, welche den Bestellungskosten ähneln. Diese sind in den Relationen *lineitem*, *partsupp* und *part*. Das Entity-Attribut wird erkannt als *c_name*.
Nun wird ausgehend von der *customer*-Relation der *join tree* erstellt, in dem die Ranking-Relationen markiert werden (gestrichelte Umrandung):
Aus diesem Baum werden die *join chains* erstellt. Diese starten ausgehend von allen Blättern, welche mindestens eine Ranking-Relation enthalten. Das sind alle außer dem sechsten, siebten und neunten

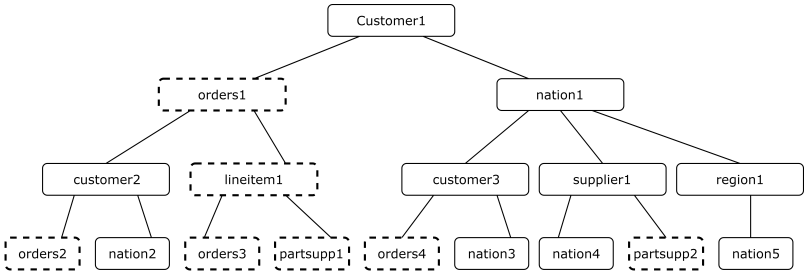


Abbildung 3.1: TPC-H join tree mit Tiefe 3

Blatt. Im vierten Schritt werden alle möglichen *lattices* erstellt. Spätestens ab einer Suchtiefe von zwei werden manche *join trees* den gleichen *lattice* erstellen. Das ist bei den Zweigen 2 und 5 der Fall.

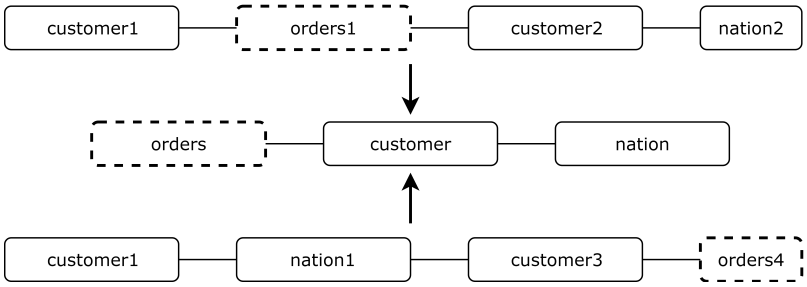


Abbildung 3.2: Join Chains der Zweige 2 und 5

Im fünften Schritt wird dann aus den zusammengefassten *join chains* eine SQL Anfrage erzeugt. Die benötigten Relationen und deren Verbunde können aus dem *lattice* abgelesen werden und die relevanten Attribute wurden bereits in der *schema exploration* ermittelt. Die SQL-Anfrage des ersten Zweigs entspricht der ursprünglichen Anfrage. □

Für das Auffinden der Verbundpfade sind vor allem die Schritte zwei bis vier wichtig. Das Ergebnis sind alle Verbundpfade, welche mindestens ein Ranking-Kriterium enthalten. Außerdem können diese auch Relationen enthalten, welche nur für die Selektion genutzt werden. Da die Suche ab einer bestimmten Tiefe abgebrochen wird, kann es jedoch sein, dass der gesuchte Verbund nicht gefunden wird. Das hat zur Folge, dass erneut mit einer größeren Tiefe gesucht werden muss. Dabei ist zu bedenken, dass der Suchbaum exponentiell in der Anzahl der Fremdschlüsselbeziehungen und Suchtiefe wächst.

3.2 Graham-Reduktion

Die Graham-Reduktion wurde laut [Fag83] und [MU84] vorgestellt in [Gra79] und in ähnlicher Form unabhängig in [YO79] beschrieben, weshalb sie häufig als GYO-Reduktion bezeichnet wird. Es handelt sich um einen Algorithmus, mit dem überprüft werden kann, ob ein Hypergraph α-azyklisch ist,

im Folgenden häufig als azyklisch bezeichnet. Nach [Mai83] handelt es sich dabei um die wiederholte Anwendung der zwei Reduktionsregeln:

Definition 3.2.1. *node removal*: Wenn ein Knoten A in maximal einer Kante enthalten ist, entferne A aus dieser Kante und der Knoten-Menge.

Definition 3.2.2. *edge removal*: Wenn eine Kante E komplett in einer anderen Kante F enthalten ist, entferne E von der Kanten-Menge.

Führt keine der beiden Regeln mehr zu einer Veränderung, ist die Reduktion abgeschlossen. In jedem Schritt werden Knoten und Kanten entfernt, welche für die Verbindungen im aktuellen Hypergraphen nicht von Bedeutung sind. Ein Knoten, welcher nur in einer Kante vorkommt, kann nicht für die Verbindung zweier Kanten relevant sein. Eine Kante, welche eine andere vollständig enthält, besitzt mindestens die selben Nachbarkanten. Also kann die kleine Kante ersetzt bzw. entfernt werden. Es gibt auch andere Definitionen der Graham-Reduktion, in der sogenannte Ohren entfernt werden. Dies sind Kanten, welche nur eine Verbindung mit dem Hypergraphen haben. Bleiben Kanten übrig, dann gibt es mindestens einen Zyklus im Hypergraphen. Liegt eine leere Menge vor, ist der Hypergraph azyklisch.

Beispiel 6. Zur Verdeutlichung der Reduktion wird diese exemplarisch an einem Teil des TPC-H-Datenbankschemas durchgeführt. Dieser besteht aus der *region*, *customer*, *nation*, *supplier* und *order*-Relation. Für die erfolgreiche Ausführung wurden die restlichen Relationen entfernt, da diese einen Zyklus bilden. Damit der Hypergraph übersichtlicher ist, wurden alle Schlüssel mit einem Buchstaben abgekürzt und alle einzigartigen Attribute einer Relation mit drei Punkten dargestellt. Abbildung 3.3

(a) node removal (b) edge removal

Abbildung 3.3: erster Reduktionschritt

zeigt den ersten Reduktionsschritt. Nach der ersten *node removal* sind nur Knoten übrig, welche in einer Schlüsselbeziehung genutzt werden, da alle anderen nur einmal vorkommen. Außerdem werden alle Kanten entfernt, welche vollständig in einer anderen enthalten sind. Das sind *order*, *region* und *supplier*. Diese sind entweder in *customer* oder *nation* enthalten. In Abbildung 3.4 wird der zweite Reduktionsschritt demonstriert. Im zweiten *node removal* werden der *customerkey* und *regionkey* entfernt. Danach bleibt in beiden Kanten nur noch der Knoten des *nationkey* übrig. Im *edge removal* wird noch eine der beiden Kanten entfernt. Damit ist nur noch eine Kante übrig, welche automatisch entfernt werden kann.

(a) node removal (b) edge removal

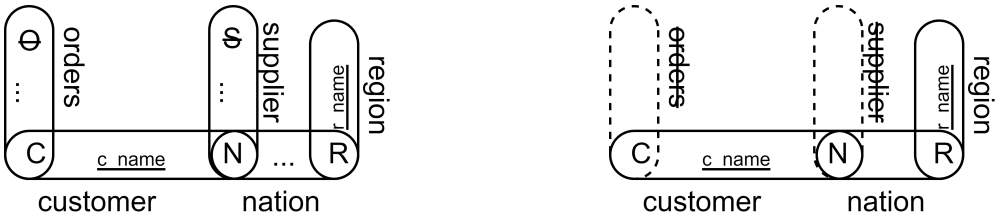
Abbildung 3.4: zweiter Reduktionschritt

Das Ergebnis der Graham-Reduktion ist eine leere Kantenmenge und somit erfolgreich. Dieser Teil des Datenbankschemas enthält keinen Zyklus. □

Erstellung von Verbundpfaden

In [MU84] wurde eine Erweiterung vorgeschlagen, mit der die Graham-Reduktion auch für die Erstellung von Verbundpfaden genutzt werden kann. Dafür werden alle Knoten, welche von Interesse sind, markiert als *sacred* Knoten. Diese werden beim *node removal*-Schritt nicht entfernt. Damit bleiben sie und auch alle Kanten erhalten, welche für die Verbindung zwischen ihnen nötig sind. Es bleibt zu beachten, dass diese Technik auch die Zyklen übrig lässt. Es müssen also bei zyklischen Hypergraphen noch weitere Techniken, bzw. Modifikationen vorgenommen werden, damit ein sinnvolles Ergebnis entsteht. In einem azyklischen Datenbankschema ist es eine einfache Methode, um den Verbundpfad zwischen beliebig vielen Attributen zu berechnen. Da nur relevante Knoten und Kanten erhalten bleiben, kann aus dem Hypergraphen leicht eine Anfrage erzeugt werden. Alle übrigen Knoten, die vorher nicht markiert wurden, werden für den Verbund zwischen den Relationen benötigt.

Beispiel 7. Es soll eine Anfrage gestellt werden, die allen Kunden den Namen ihrer Region zuordnet. Es werden also die entsprechenden Knoten der Attribute der *customer* und der *region*-Relation als *sacred* markiert. Dadurch verändert sich das Ergebnis der Reduktion.



(a) node removal (b) edge removal

Abbildung 3.5: erster Reduktionschritt

Wie die Abbildung 3.5 zeigt, unterscheidet sich der erste Reduktionsschritt im Vergleich zur Abbildung 3.3. Die beiden markierten Knoten werden nicht entfernt. Dadurch ist die *region*-Kante nicht in der *nation*-Kante enthalten und bleibt deshalb erhalten. Da die *region*-Kante weiterhin im Hypergraphen ist,

(a) node removal (b) edge removal

Abbildung 3.6: zweiter Reduktionschritt

bleibt der *regionkey* im zweiten Reduktionsschritt erhalten. In Abbildung 3.6 ist erkennbar, dass, nach der Entfernung von *customerkey*, weder Knoten noch Kanten entfernt werden können. Die Reduktion ist abgeschlossen. Aus den übrigen Knoten und Kanten kann eine SQL-Anfrage erstellt werden.

```
SELECT c.c_name, r.r_name
FROM customer c, region r, nation n
WHERE c.c_nationkey = n.n_nationkey AND n.r_regionkey = r.r_regionkey
```

Das gleiche Ergebnis kann mit dem Einsatz eines Tableaus erreicht werden. Um dies zu zeigen, wurde dieses Beispiel im Anhang A erneut mit Tableaus durchgeführt. □

3.3 TPC-H-Benchmark

Die TPC [TPC] ist eine Non-profit-Kooperation, welche Benchmarks für Datenbankmanagementsysteme veröffentlicht. Diese bestehen aus Schemata, Daten und Anfragen, welche typisch für bestimmte Anwendungsbereiche sein sollen. Dafür werden Tools bereitgestellt, welche unterschiedlich viele Datensätze erzeugen können. Der TPC-H-Benchmark ist für den Bereich *decision support* entwickelt worden. Dabei handelt es sich um komplexe Anfragen, welche Informationen zur Entscheidungshilfe bereitstellen sollen, wie z.B. Top-K-Anfragen. Die Geschwindigkeit des Systems wird in Anfragen pro Stunden gemessen. Zusätzlich wird die Größe der verwendeten Datenmenge angegeben, um die Geschwindigkeit besser einschätzen zu können. Damit eignet er sich gut als Ausgangslage für das Top-K Reverse Engineering.
Das Datenbankschema, dargestellt in Abbildung 3.7, entspricht dem einer Lieferfirma, welche Bestellungen von Teilen entgegen nimmt, die sie von anderen Firmen erhalten und an Kunden ausliefern. Es besitzt einen sehr großen Zyklus, welcher fast alle Relationen umfasst. Dadurch stellt es ein großes Problem für Verfahren dar, welche auf der Graham-Reduktion beruhen.

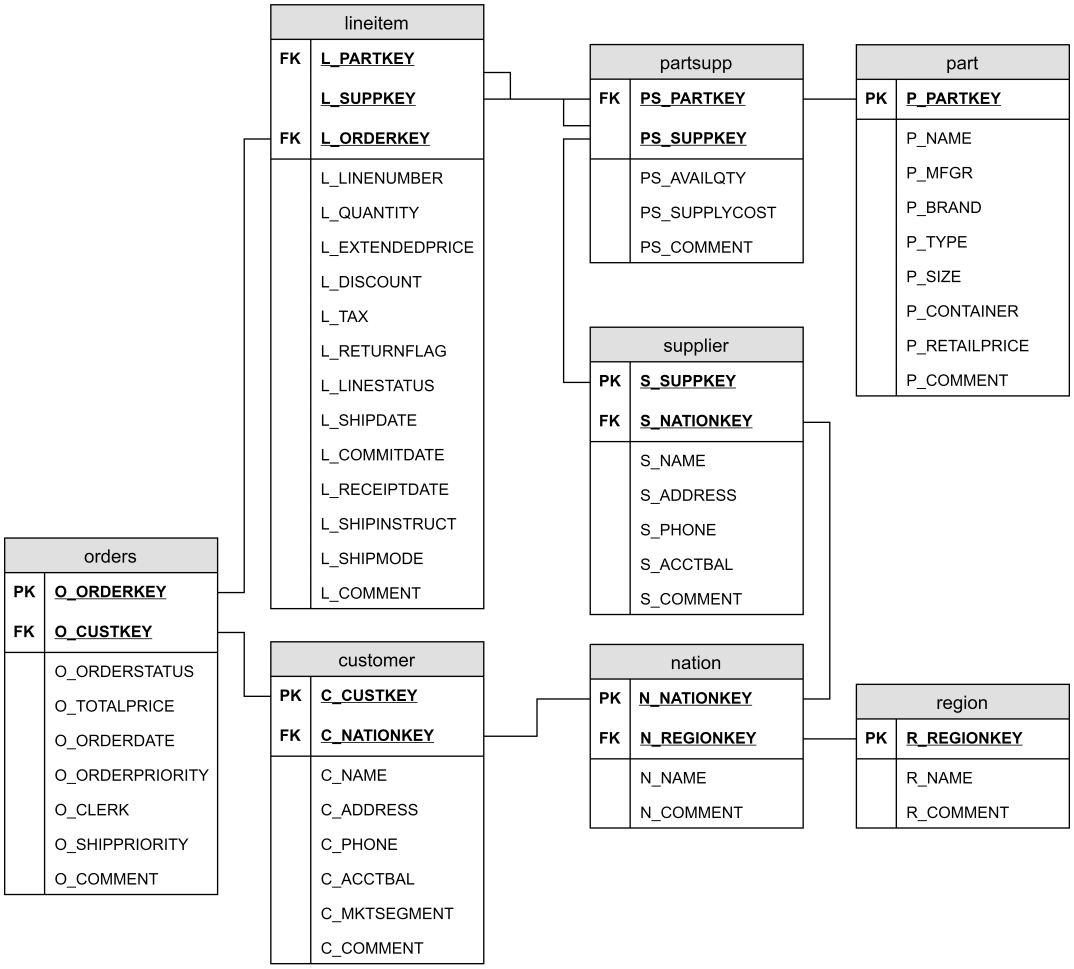


Abbildung 3.7: TPC-H-Datenbankschema

Kapitel 4

Konzept

Das Paleo-J Framework wurde gezielt dafür entwickelt, alle relevanten Verbundpfade in einem Datenbankschema zu finden. Diese müssen ihrem Template entsprechen. Die einzige Anforderung, die an das Datenbankschema gestellt wird, ist, dass die Fremdschlüssel korrekt implementiert wurden. Um das zu erreichen, setzen sie eine Breitensuche ein, welche alle Fremdschlüsselbeziehung nutzt. Das hat zwei Nachteile. Zum einen entscheidet die Tiefe des Suchbaums maßgeblich, ob der Verbundpfad gefunden werden kann. Zum anderen wächst der Suchbaum exponentiell mit der Tiefe und der Verzweigung der Relationen. Dadurch muss abgewogen werden, ob eine Suche mit größerer Tiefe sinnvoll ist. In der Praxis ist es also denkbar, dass nicht alle Verbundpfade gefunden werden.

Die Graham-Reduktion hingegen hatte ursprünglich einen ganz anderen Zweck. Diese wurde genutzt, um die Azyklität eines Datenbankschemas zu überprüfen. Durch die Erweiterung der *sacred nodes* können aber auch Verbundpfade ermittelt werden. Dazu müssen die gesuchten Attribute angegeben werden. Es wird gezielt nach einem Verbundpfad gesucht, welcher diese Attribute miteinander verbindet. Das Ergebnis ist eine Menge von Relationen, von denen alle unbedingt für den Verbund benötigt werden. Datenbankschemata mit Zyklen stellen die einzige Ausnahme dazu dar. Bei diesen ist es nicht ohne weiteres möglich, einen vernünftigen Verbundpfad zu finden, da der Zyklus immer erhalten bleibt.

Die Graham-Reduktion und Paleo-J unterscheiden sich grundsätzlich in ihrer Anforderung und ihren Ergebnissen. Paleo-J verarbeitet die Ausgabe einer Anfrage und versucht mit den originalen Daten diese Anfrage zu rekonstruieren. Ziel ist es, Anfragen zu finden, welche das gleiche Ergebnis erzeugen. Dafür wird aus den Werten der Ergebnisliste eine Menge von potentiellen Attributen berechnet. Dann werden alle möglichen Verbundpfade untersucht und ob diese die berechneten Attribute beinhalten. Die Graham-Reduktion hingegen wird genutzt um gezielt einen Verbundpfad mit gegebenen Attributen zu finden. Es wird also nur einer statt vieler Pfaden gesucht. Ein weiterer wichtiger Aspekt von Paleo-J ist auch, dass Verbundpfade mit Filter-Relationen untersucht werden. Im Gegenteil dazu sollen bei der

Graham-Reduktion keine unbeteiligten Relationen enthalten sein. Außerdem spielt es keine Rolle, ob Relationen einfach oder mehrfach verwendet werden. Das ist bei vielen klassischen Ansätzen nicht vorgesehen, da so die Menge möglicher Verbundpfade enorm steigt.

Der Vorteil eines Ansatzes mit Graham-Reduktion ist, dass Verbundpfade zwischen gesuchten Attributen schnell gefunden werden können, ohne alle möglichen Verbundpfade erstellen zu müssen. Es gibt jedoch zwei große Nachteile. Zyklen verhindern die Verarbeitung der Datenbankschemata und bei der Verbundpfaderstellung ist es nicht vorgesehen, dass eine Relation mehrfach verwendet wird. Um ein vergleichbares System zu erstellen, ist also noch weitere Arbeit nötig. Nachfolgend werden die Probleme und deren Auswirkungen analysiert. Es werden Modifikationen und Verfahren untersucht, welche diese lösen sollen, und ein gesamter Entwurf vorgestellt, welcher vergleichbar mit der Verbundpfaderstellung des Paleo-J sein soll.

4.1 Probleme der Graham-Reduktion

Wie schon beschrieben, kann die Graham-Reduktion nur ein Teil des Konzeptes sein. Da die Graham-Reduktion ursprünglich dazu gedacht war Zyklen zu finden, bleiben diese erhalten. Das betrifft zwar nur die beschriebenen α-Zyklen, jedoch zeigt sich, dass auch andere Zyklen Probleme bereiten. Auch die Darstellung von Datenbankschemata in Form von Hypergraphen benötigt etwas Arbeit. Es muss sichergestellt werden, dass die Knoten die entsprechenden Attribute auch korrekt repräsentieren. Vor allem die Mehrfachverwendung einer Relation bereitet so einem Ansatz große Probleme.

4.1.1 Datenbankschema mit α-Zyklus

Zwei Relationen in einem zyklischen Datenbankschema, wie *customer* und *supplier* in Abbildung 3.7, besitzen mehrere Verbundpfade zueinander. Bei der Graham-Reduktion bleibt jedoch immer der gesamte Zyklus erhalten, obwohl dies meist nicht von Interesse ist. Es sollen alle möglichst minimalen Verbundpfade gefunden werden. Deshalb sind Pfade, welche nur Teile des Zyklus enthalten, viel interessanter. Dabei müssen zunächst die zwei Fälle betrachtet werden:

Der Zyklus wird nicht benötigt: Obwohl keine Relation des Zyklus für den gesuchten Verbundpfad nötig wäre, bleiben diese im Hypergraphen erhalten. Außerdem können zusätzliche unnötige Relationen erhalten bleiben, welche den eigentlichen Verbundpfad mit dem Zyklus verbinden. Diese sind nur in seltenen Fällen die Lösung und können allgemein als falsch angesehen werden. Der Zyklus sollte in diesem Fall nicht im Verbundpfad sein.

Der Zyklus wird benötigt: Wenn Relationen des Zyklus gesucht sind, wird dennoch der gesamte Zyklus im Verbundpfad erhalten bleiben. Es gibt aber mindestens zwei Verbundpfade, welche teils

unterschiedliche Relationen des Zyklus verwenden. Diese bilden die Ausgangslage für eine nachträgliche Erweiterung um weitere Relationen, falls noch Filterrelationen gesucht werden.

Im ersten Fall kann der Zyklus vor der Reduktion entfernt, aufgetrennt oder in einer Kante zusammengefasst werden. Durch das Entfernen wird der Hypergraph aufgeteilt und es können alle Verbundpfade gefunden werden, welche keine Relation des Zyklus enthalten würden. Dadurch kann der zweite Fall nicht gelöst werden. Durch das Auftrennen geht einer der Verbundpfade verloren und es muss überhaupt eine richtige Kante oder Knoten zum Entfernen gefunden werden. Der Zyklus muss vor der Reduktion verändert oder der Verbundpfad nach der Reduktion entsprechend weiterverarbeitet werden. Klassische Ansätze, wie [Hal86], suchen nach dem kürzesten Pfad oder lassen dem Nutzer die Auswahl. Dabei wird nur nach einem Verbundpfad gesucht. In dieser Arbeit werden aber alle Verbundpfade gesucht. Deshalb sollen auch die unterschiedlichen Pfade durch den Zyklus gefunden werden.

4.1.2 Datenbankschema mit anderen Zyklen

Ebenso problematisch können auch die anderen Arten der Zyklen sein. In bestimmten Fällen werden Relationen durch die Graham-Reduktion immer entfernt. Der TPC-H-Benchmark hat gezeigt, dass α-Zyklen nicht untypisch sind. Auch die Existenz eines β-Zyklus ist nicht unwahrscheinlich. In manchen Datenbanken werden aus Performance-Gründen materialisierte Sichten eingesetzt, welche häufig die Schlüssel mehrerer Relationen enthalten. Es kann also durchaus gewollt sein, dass diese Sichten bei der Erstellung von Anfragen berücksichtigt werden sollen. Diese Sichten führen aber zu β- oder γ-Zyklen. An einem Beispiel soll verdeutlicht werden, welche Folgen ein β-Zyklus für die Graham-Reduktion hat.

Beispiel 8. Wird die Graham-Reduktion mit *sacred nodes* am Schema *{ABC, CDE, AFE, ACE}* aus Abbildung 2.1b durchgeführt, bleibt nur die *ACE*-Relation übrig, außer *B*, *D* oder F sind die *sacred nodes*. Es sollen aber alle Verbundpfade gefunden werden. Eine Top-K-Anfrage, welche *A* zur Identifikation und *E* als Ranking nutzt, zeigt diesen Nachteil deutlich. Im Paleo-J-System wird der Baum ausgehend von den *A*-Relationen erstellt. Das sind *ABC*, *AEF* und *ACE*. Für das Ranking werden *CDE*, *AEF* und *ACE* verwendet.

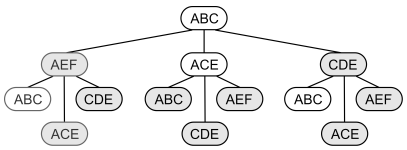


Abbildung 4.1: Paleo-Suchbaum von ABC

Abbildung 4.1 zeigt den Suchbaum, der von Paleo-J erzeugt werden sollte. Die daraus resultierenden Verbundpfade sind: *ABC-AEF*, *ABC-CDE*, *ABC-AEF-ACE* und so weiter. Es fehlen nur *ABC-ACE*, welche im *ACE*-Baum erstellt werden, und der Verbund aller Relationen, welcher bei einem höheren Limit erzeugt wird.

Bei der Graham-Reduktion werden alle Relationen durch die *ACE*-Relation entfernt. Da die *sacred nodes* nur vor dem *node removal* schützen, werden die Relationen trotzdem entfernt. Letztendlich wird immer nur *ACE* übrig bleiben. Kanten mit *sacred nodes* müssen auch entfernbar sein, weil sonst alle Kanten erhalten bleiben, was auch nicht hilfreich wäre. □

Laut [CZ91] ist es möglich, β-Azyklität in polynomieller Laufzeit zu finden. Der Einsatz von weiteren Verfahren neben der Graham-Reduktion erhöht jedoch die Komplexität bei der Suche nach den Verbundpfaden. Es ist fraglich, wie häufig diese Zyklen in einem Datenbankschema auftauchen. Außerdem bereiten auch die anderen Arten der Zyklen Probleme. Bei einem γ-Zyklus werden ähnlich dem β-Zyklus Kanten immer entfernt. Dieser hat jedoch einen anderen Einfluss auf Verbundpfade. Deshalb wäre es auch hier nötig, diesen mit einem Algorithmus zu erkennen. Manche Probleme mit einem Berge-Zyklus zeigen sich schon bei Relationen mit mehr als einem gemeinsamen Attribut. In diesem Fall ist die Unterscheidung nötig, ob es nur einen zusammengesetzten Schlüssel gibt oder ob es auch mehrere Schlüssel sein können. Bei der Erstellung einer Anfrage ist es wichtig, ob alle oder nur ein Teil der Attribute für den Verbund verwendet werden.
Da jede Art eines Zyklus spezifische Folgen für das Ergebnis hat, müsste also für jeden eine Erkennung durchgeführt werden. Dazu gehört auch die Betrachtung vieler unterschiedlicher Kombinationen. Im Fall der Graham-Reduktion verhindern nur α-Zyklen die grundsätzliche Funktionalität. Deshalb liegt die Priorität zunächst darauf, mit diesen umgehen zu können. Ist das erfolgreich, ist es denkbar, dass statt einer Erkennung und speziellen Verarbeitung der anderen Zyklen ein Verfahren eingesetzt wird, um die verlorenen oder alternativen Verbundpfade zu finden. Dieses könnte die ermittelten Verbundpfade untersuchen und überprüfen, ob es Alternativen gibt.

4.1.3 Benennung von Attributen

Die erste Hürde bei einer Umsetzung mittels Graham-Reduktion ist die URSA. Vor der Erzeugung des Hypergraphen muss sichergestellt werden, dass dieser auch die Beziehungen im Datenbankschema korrekt darstellt. Das Datenbankschema muss also in einer Form vorliegen, sodass gemeinsame Attribute von Relationen den gleichen Namen haben. Dafür muss es wahrscheinlich modifiziert werden. Ohne eine Umbenennung wird ein Ansatz mit Hypergraphen nur funktionieren, wenn bei der Erstellung die URSA bedacht wurde. Das TPC-H-Schema zeigt jedoch, dass auch professionell entwickelte Schemata diese nur bedingt beachten. Zwar ist jedes Attribut einzigartig benannt, das trifft aber auch auf die Schlüssel zu. Diese müssen also noch umbenannt werden. Ein anderes Problem ist häufig, dass mehrere Attribute den gleichen Namen haben, aber nicht die gleiche Bedeutung.

Beispiel 9. Im TPC-H-Benchmark ist gut erkennbar, dass es häufig Attribute mit ähnlichem oder gleichem Namen gibt. So hat jede Relation ein *comment*-Attribut. Diese sind immer nur genau für diese

Relation gedacht. Hätten sie nicht den eindeutigen Präfix ihrer Relation, würden sie von einem Hypergraphen als gemeinsame Knoten erkannt. Dann würden Verbunde über gleiche Kommentare erstellt, was Unsinn ist und nur falsche Ergebnisse liefert. □

Ein verwandtes Problem zu dieser Thematik ist das Auffinden von Verbunden über Nichtschlüsselattribute. So ist es im TPC-H-Benchmark denkbar, dass nach Kunden gesucht werden, die auch Lieferanten sind. Dazu werden die Namen und Adressen verglichen. Leider ist aus dem Namen nicht ableitbar, ob das Attribut für einen Verbund nutzbar ist oder nicht. Deshalb kann nicht einfach bei gleichem oder ähnlichem Attributnamen angenommen werden, dass auch die Informationen gleich sind. Außerdem ist es möglich, dass Attribute, welche für einen Verbund genutzt werden könnten, völlig unterschiedliche Namen haben. Die Erkennung solcher Zusammenhänge ist nicht einfach lösbar und auch nicht Ziel dieser Arbeit und wird deshalb nicht weiter beachtet.

4.1.4 Self Joins

Anfragen mit *self joins*, also Anfragen, die Relationen mehrfach enthalten, sind eine besondere Herausforderung für einen Ansatz mit Hypergraphen. Dieser enthält genau eine Kante für jede Relation. Es ist mit der Graham-Reduktion nicht vorgesehen, dass eine Relation mehrfach an einem Verbund beteiligt ist. Können Relationen beliebig oft an einem Verbund teilnehmen, erweitert sich die eigentlich begrenzte Anzahl der Verbundpfade auf unendlich viele. In der Praxis sollte eine Relation nur zwei- oder dreimal verwendet werden. Es ist also sinnvoll, nur von solchen Fällen auszugehen. Trotzdem handelt es sich um eine Aufgabe, welche mit diesem Ansatz nur schwer gelöst werden kann. Denkbar wäre es, den Hypergraph zu vervielfältigen oder mehrere Ebenen zu erstellen. Dann muss die Graham-Reduktion entsprechend angepasst werden. Außerdem entstehen bei solch einer Lösung viele Zyklen, die wiederum andere Probleme verursachen. Alternativ können auch Verbundpfade erstellt und mit der Paleo-J-Technik erweitert werden. Dadurch können die Vorteile der Graham-Reduktion weiterhin genutzt werden. Dieser Ansatz verlässt sich jedoch darauf, dass der korrekte Verbundpfad erweitert wird. Werden alle Verbundpfade zufällig bearbeitet, dann macht es im Falle einer *self join* Anfrage viel Aufwand. Ein anderer Ansatz wäre, die Kanten der erstellten Verbundpfade zu vervielfältigen. Auch in diesem Fall macht das nur Sinn, wenn eine Vorauswahl plausible Verbundpfade zuverlässig erkennt. Bis auf einen mehrschichtigen Hypergraphen entsprechen die Lösungen etwa dem Paleo-J-Ansatz und haben entsprechende Nachteile. Ein erweiterter Hypergraph erzeugt jedoch Probleme, welche erst mit dieser Arbeit gelöst werden. Es ist auch möglich, die Ansätze zu kombinieren. So kann für eine Erweiterung oder Verdoppelung der Verbundpfade der mehrschichtige Hypergraph verwendet werden.

Zusammenfassung

Um ein vergleichbares System zu entwickeln, müssen diese Probleme zumindest teilweise überwunden werden. Zunächst muss ein korrekter Hypergraph erstellt werden. In klassischen Ansätzen werden immer die Namen der Attribute zur Erstellung der Knoten genutzt. Diese setzten jedoch die URSA voraus. Da nicht angenommen werden kann, dass diese umgesetzt wurde, kann der Hypergraph nicht direkt aus dem Namen der Attribute erstellt werden. Außerdem wird durch die Azyklität eine spezifische Anforderung an das Datenbankschema gestellt. Auch wenn diese positive Eigenschaften garantiert, muss sie nicht erfüllt sein. Der TPC-H-Benchmark zeigt, dass es durchaus üblich ist, dass zyklische Schemata eingesetzt werden. Der dort vorhandene α-Zyklus verhindert eine Arbeit mit der Graham-Reduktion. Zusätzlich können auch die anderen Arten von Zyklen enthalten sein. Diese bereiten auch Probleme, werden aber nicht von der Graham-Reduktion erkannt. Dazu kommen noch weitere kleinere Probleme. Beispielsweise verwendet Paleo-J die Schlüssel, welche bei der Erstellung des *join tree* genutzt wurden. Im Hypergraphen-Ansatz werden dafür die gemeinsamen Attribute verwendet. Diese müssen nicht immer nur einen, sondern können auch mehrere Schlüssel enthalten. Zuletzt bleibt noch das Auffinden von Verbundpfaden mit Mehrfachverwendung der Relationen. Da die anderen Probleme den Einsatz der Graham-Reduktion blockieren oder einschränken, haben diese Priorität.

Der Nutzen der Graham-Reduktion mit azyklischen Datenbankschemata, die der URSA folgen, wurde schon in [MU84] gezeigt. Dabei handelt es sich aber um Annahmen, welche in der Praxis häufig nicht erfüllt werden. In dieser Arbeit wird nach einer Lösung gesucht, die α-Zyklen aufzulösen und auch die anderen Arten von Zyklen zu berücksichtigen. Außerdem wird die Benennung der Attribute untersucht. Zusammen soll dies ermöglichen, wie bei Paleo-J, keine speziellen Anforderungen an ein Datenbankschema stellen zu müssen.

4.2 Anpassen der Attributnamen

Als erster Schritt der Umsetzung dieser Arbeit müssen die Attributnamen umbenannt werden. Es ist allgemein sinnvoll, alle Attribute so zu benennen, dass sie einzigartig sind, wenn es sich nicht um Schlüssel handelt. Damit werden ungewollte Beziehungen vermieden, welche zusätzliche falsche Ergebnisse liefern oder nicht vorhandene Zyklen erzeugen. Sind alle Attribute einzigartig, müssen die Fremdschlüsselbeziehung umgesetzt werden. Dafür sind zwei Ansätze denkbar:

1. die Gleichbenennung von Knoten, welche die Schlüssel bzw. Fremdschlüssel darstellen
2. die Erzeugung zusätzlicher Knoten, welche je in die beiden Kanten der Relationen eingefügt werden

Diese Ansätze erzeugen unterschiedliche Hypergraphen, deren Verbundpfade sich unterscheiden können. Bei der Gleichbenennung werden die Fremdschlüssel nach dem dazugehörigen Primärschlüssel benannt. Ein Nebeneffekt ist, dass indirekte Fremdschlüsselbeziehungen zu direkten Beziehungen werden. Das hat den Vorteil, dass die ursprüngliche Relation des Schlüssels für einen Verbund nicht benötigt wird. Es gibt jedoch Schemata, bei der die Gleichbenennung Fehler erzeugt bzw. nicht direkt umsetzbar ist. Sollten in einer Relation zwei Fremdschlüssel den selben Primärschlüssel referenzieren, würden die beiden den gleichen Namen erhalten.

Beispiel 10. Nach einer Gleichbenennung der Schlüssel im TPC-H-Datenbankschema, haben die Relationen *customer* und *supplier* ein gemeinsames Attribut und sind miteinander verbunden. Dies wird in Abbildung 6.1 gezeigt. Diese indirekten Fremdschlüsselbeziehungen haben den Vorteil, dass bei manchen Anfragen weniger Relationen im Verbund benötigt werden. Wird zum Beispiel nach Kunden gesucht, welche die selbe Nationalität wie ein Lieferant haben, ist der Verbund mit der *nation*-Relation unnötig, da diese das Ergebnis nicht beeinflusst. □

Beispiel 11. Angenommen, es handelt sich um ein Datenbankschema eines Versandhauses. Kunden können anderen Kunden Produkte empfehlen. Wird dann eine Bestellung getätigt, erhalten sie eine Provision. In der Bestellungsrelation wird einmal die Kundennummer des Bestellenden und die des Vermittlers gespeichert. Es gibt also zwei Fremdschlüssel, welche die Kundennummer referenzieren. Nach einer Gleichbenennung wird eines der Attribute überschrieben. □

Im zweiten Ansatz wird für jede Fremdschlüsselbeziehung ein zusätzlicher Knoten erstellt, welcher nur in den beiden Kanten der Relationen enthalten ist. Der Vorteil ist, dass doppelte Fremdschlüssel erhalten bleiben. Dafür werden die Fremdschlüsselbeziehungen nicht zu direkten Beziehungen.

Durch die Gleichbenennung werden kürzere Verbundpfade erstellt. Das ist eine Folge aus den zusätzlichen Verbindungen der Kanten. Das bedeutet auch, dass der zweite Ansatz eventuell Verbundpfade nicht findet. Diese sind um die Relation gekürzt, aus der die Fremdschlüssel stammen. Der Verbund mit dieser Relation reduziert die Anzahl der Tupel nicht und hat auch sonst keinen Einfluss. Um ein mit Paleo-J vergleichbares System zu entwickeln, müssen sowieso auch die Verbundpfade mit Filterrelationen erzeugt werden. Nach der Erstellung der Verbundpfade folgt also noch ein Schritt, welcher zusätzliche Relationen hinzufügt. Im Ergebnis sollten dann auch die Verbundpfade des zweiten Ansatzes enthalten sein. Deshalb wird die Gleichbenennung bevorzugt eingesetzt. Sollte es zu einem Doppelschlüssel-Fall kommen, wird dieser mit der zweiten Technik aufgelöst.

4.3 Umgang mit Zyklen

Zyklen verfälschen die Reduktion, weil dadurch unbeteiligte Relationen nicht entfernt werden. Leider ist der Umgang mit Zyklen schwierig. Bei einem Zyklus gibt es mindestens zwei Pfade zwischen zwei Knoten. Diese unterschiedlichen Pfade sind essentiell, um alle Verbundpfade finden zu können und somit vergleichbare Ergebnisse zu Paleo-J zu liefern. Da es jedoch viele verschiedene Formen von Zyklen gibt, ist es schwierig einen allgemein gültigen Ansatz zu finden, welcher immer alle Pfade eines Zyklus findet.

4.3.1 Einfache Zyklen

Da die Definition von α-Zyklen kaum einen Rückschluss auf ihre Struktur gibt, wird zunächst nur die simpelste Form eines Zyklus betrachtet. Das soll dabei helfen die Verbundpfadberechnung zu erleichtern. Dabei handelt es sich um einen α-zyklischen Hypergraphen, bei der jede Kante exakt zwei Nachbarn hat. Diese Zyklen werden fortan als *einfacher* Zyklus bezeichnet. Es gibt ähnliche Definitionen, z.B. werden sie in [CZ91] als *pure cycles* oder in [Fag83] als *graham cycle* bezeichnet. Der Vorteil so eines Graphen ist, dass es zwischen zwei Knoten immer nur zwei Pfade gibt. Diese sind leicht auffindbar, da eine Kante nur zwei Nachbarn hat. Für die folgenden Erklärungen werden zunächst Definitionen benötigt. So werden Kanten, die mit dem Zyklus verbunden sind, aber nicht zu ihm gehören, Brücken genannt. Die gemeinsamen Knoten der Brücken mit dem Zyklus sind Eingangs- bzw. Ausgangsknoten. Die Kanten dieser Knoten, die innerhalb des Zyklus liegen, sind die Eingangs- bzw. Ausgangskanten des Algorithmus. Um beide Pfade zu finden, sind die folgenden Schritte nötig:

1. Entferne die Eingangskante des Eingangsknoten aus dem Hypergraphen. Bei zwei Eingangskanten entferne deren gemeinsame Knoten. Der Ausgangsknoten wird als *sacred node* makiert.
2. Kopiere den Hypergraphen. Gibt es nur eine Eingangskante, markiere je einen ihrer Nachbarn als *sacred*. Bei zwei Eingangskanten, markiere je eine von diesen.
3. Führe in beiden Hypergraphen die Graham-Reduktion durch.
4. Füge die entfernte Kante bzw. Knoten aus dem ersten Schritt wieder zum Hypergraphen hinzu.

Durch dieses Vorgehen werden die beiden Pfade gefunden. Statt der Graham-Reduktion kann auch eine andere Suche nach einem Pfad verwendet werden, da in den beiden Graphen je nur ein minimaler Pfad zwischen den Knoten vorhanden ist. Welche der beiden Kanten die Eingangs- oder Ausgangskante ist, spielt keine Rolle. Es ist jedoch wichtig, wie viele innere Nachbarn der Eingangsknoten im Zyklus hat. Damit der Zyklus korrekt aufgetrennt werden kann, ist es bei einer einzelnen Eingangskante nötig, diese komplett zu entfernen. Dies spielt bei der Ausgangskante keine Rolle, da hier nur relevant ist, dass der Knoten zur Brücke erhalten bleibt. Das Problem mit dieser Idee ist jedoch, dass α-Zyklen zwar in dieser

Form vorliegen können, aber nicht müssen. Mit diesem Vorgehen wird also nur eine spezielle Form oder ein Teil des Zyklus abgedeckt.

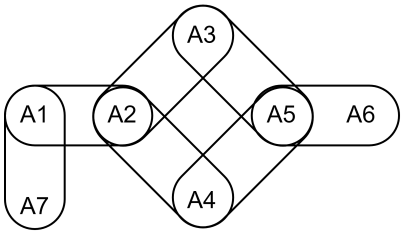


Abbildung 4.2: Hypergraph mit Zyklus

Beispiel 12. Es soll ein Verbundpfad für ein Schema gefunden werden, welches durch den Hypergraph in Abbildung 4.2 dargestellt wird. Die gesuchten Attribute sind A1 und A6. Es gibt zwei direkte Pfade, welche durch den Zyklus führen:

1. {A2, A3} und {A3, A5}.
2. {A2, A4} und {A4, A5}.

Diese können über den vorgeschlagenen Algorithmus gefunden werden, indem zwei Hypergraphen mit den Kanten des Zyklus erstellt werden. A2 wird als *sacred* makiert. Dann werden in beiden Graphen die gemeinsamen Knoten der Kanten von A5 entfernt, welches nur A5 selber ist. {A3, A5} bzw. {A4, A5} wird als *sacred* makiert. Nach der Reduktion stellen die Hypergraphen je einen der Pfade dar. □

Es gibt noch mehr Verbundpfade, die mit einem Zyklus erzeugt werden können. Dazu gehört beispielsweise der gesamte Zyklus oder ein Pfade und eine Teilmenge des anderen. Ziel bei der Auflösung der Zyklen ist es jedoch, die kürzesten beiden Pfade zu finden. Alle weiteren Pfade werden in einem anderen Schritt erzeugt.

4.3.2 Superkanten als Lösung einfacher Zyklen

Die Verbundpfade innerhalb des *einfachen* Zyklus zu finden ist leicht. Dennoch bleiben die Zyklen in der Graham-Reduktion erhalten. Diese müssen durch eine Vorverarbeitung entfernt werden. In Anlehnung an die *hinge decomposition* aus [Gys86] wird eine *Superkante* eingesetzt, welche den Zyklus zusammenfassen soll. Sie enthält alle Knoten ihrer untergeordneten Kanten und somit auch alle ihre Nachbarn. Um solch eine Superkante zu erzeugen, wird zuerst der *einfache* Zyklus ermittelt. Dann werden alle Kanten und Knoten zu einer übergeordneten Superkante hinzugefügt. In nachfolgenden Reduktionen wird diese als normale Kante behandelt. Ist die Superkante am Verbundpfad nicht beteiligt, wird sie einfach entfernt. Falls sie doch beteiligt ist, müssen bei der Auswertung die unterschiedlichen Pfade erzeugt werden. Da

eine Superkante immer einen *einfachen* Zyklus enthalten soll, können mit dem beschriebenen Verfahren die beiden Verbundpfade ermittelt werden.

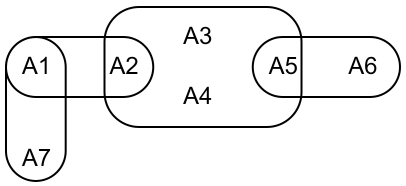


Abbildung 4.3: Hypergraph mit Superkante

Beispiel 13. Damit ein Datenbankschema wie in Abbildung 4.2 mit der Graham-Reduktion bearbeitbar sein kann, muss der Zyklus bestehend aus {A2, A3}, {A2, A4}, {A3, A5} und {A4, A5} entfernt werden. Diese werden zu einer großen gemeinsamen Kante hinzugefügt. Dadurch entsteht ein Hypergraph wie in Abbildung 4.3. Bei einer Reduktion, welche den Zyklus nicht betrifft, wird eine Superkante einfach entfernt. Sind A2 und A7 *sacred nodes*, bleiben nur die Relationen {A1, A2} und {A1, A7} übrig. □

Es bleibt noch, die entscheidenden Knoten und Kanten im Zyklus zu finden. Ist die Reduktion mit *sacred nodes* abgeschlossen, sind in der Superkante nur noch die Knoten enthalten, die für die Verbindung mit den Brücken verantwortlich sind. Diese können wiederum den ursprünglichen Kanten zugeordnet werden. Diese stellen die Eingangs- und Ausgangskanten für den Algorithmus dar. Es entstehen unterschiedliche Fälle, die von der Anzahl der Eingangskanten im Zyklus abhängen. Bei einer einzelnen Eingangskante, muss diese entfernt werden um den Zyklus aufzutrennen. Bei zwei müssen die gemeinsamen Knoten entfernt werden. Außerdem muss unterschieden werden, wie viele Eingangsknoten es gibt. *sacred nodes* müssen immer als Eingangsknoten berücksichtigt werden, auch wenn diese keine Brücke haben. Es kann also vorkommen, dass keine oder nur eine Brücke ermittelt wird. In diesem Fall liegt einer der *sacred nodes* im Zyklus. Es bleibt die Unterscheidung, ob es zwei oder noch mehr Eingangsknoten gibt:

Zwei Eingangsknoten: Dieser Fall entspricht dem Algorithmus.

Mehr Eingangsknoten: Hat ein Zyklus mehr als zwei Eingangsknoten, dann muss der Algorithmus mehrfach durchgeführt werden. Dabei gilt, dass jeder Eingangsknoten einmal den Eingangsknoten/-kanten stellt und alle anderen Eingangsknoten als Ausgangsknoten fungieren. Im Algorithmus müssen also mehrere Ausgangsknoten berücksichtigt werden. Die Anzahl der erstellten Verbundpfade entspricht den Eingangsknoten.

Es gibt also mehrere Fälle, welche unterschiedliche Eingaben für den Algorithmus erzeugen. Der einfachere Fall mit einer Eingangskante wird wahrscheinlich seltener eintreten, da es sich bei dem Eingangsknoten meist um einen Schlüssel handelt, welcher auch andere Relationen verbindet. Dafür kann es mehr

als zwei Eingangsknoten nur geben, wenn es mehr als zwei *sacred nodes* gibt. Der häufigste Fall sollte also zwei Eingangsknoten mit zwei Eingangskanten sein.

4.3.3 Komplexere Zyklen und Markierung / Rekonstruktion

Das beschriebene Verfahren funktioniert nur, wenn der beschriebene *einfache* Zyklus vorliegt. Es können jedoch komplexere Strukturen nach der Graham-Reduktion übrig bleiben. Dazu gehören zum einen komplexere Zyklen, welche mehrfach mit sich selbst verbunden sind, zum anderen auch Hypergraphen, welche mehrere Zyklen besitzen. Zwischen diesen bleibt nach der Reduktion eine Verbindung aus eigentlich unbeteiligten Kanten zurück. Diese sind nicht Teil des Zyklus und müssen deshalb auch nicht gesondert betrachtet werden, bleiben aber bei der Reduktion erhalten und können nicht direkt als solche identifiziert werden.

(a) unbearbeitet (b) mit Superkante

Abbildung 4.4: Zwei einfache Zyklen mit Verbindung

Beispiel 14. Die Abbildung 4.4a zeigt einen Hypergraphen, welcher zwei einfache Zyklen enthält. Dieser Hypergraph kann nicht zu einer Superkante zusammengefügt werden, weil diese nicht mit dem Algorithmus ausgewertet werden könnte. Es gibt hier keine zwei definierten Wege. Es bietet sich jedoch an, zumindest Teile dieser Graphen als Superkante zusammenzufassen. Werden die beiden Zyklen zu Superkanten, wie in Abbildung 4.4b, kann korrekt mit ihnen gearbeitet werden. □

(a) unbearbeitet (b) mit Superkante

Abbildung 4.5: Dreierzyklus

Beispiel 15. Ein weiteres Beispiel wird in Abbildung 4.5a gezeigt. Im Gegensatz zum vorhergehenden Beispiel, überschneiden sich zwei einfache Zyklen und bilden so eine komplexere Struktur. Dennoch

kann durch das Hinzufügen einer Superkante dieser Zyklus vereinfacht werden. Wie die Abbildung 4.5b zeigt, entsteht dadurch ein einfacher Zyklus, welcher wiederum zu einer Superkante hinzugefügt wird. □

Ziel ist es, die komplexen, zyklischen Strukturen in einfache Zyklen aufzuteilen. Diese werden zu Superkanten hinzugefügt und lösen die Struktur schrittweise auf. Dazu müssen diese zunächst aufgefunden werden. Dies wird mit einer Variante des Dijkstra-Algorithmus versucht:

Algorithmus 1: Markier-Algorithmus

```
Data: Hypergraph H nach Graham-Reduktion
Result: markierter Hypergraph
Markiere alle Knoten und Kanten von H mit -1;
sn = ein Konten mit geringster Kantenanzahl;
queue = {sn};
while not empty queue do
    node1 = dequeue(queue);
    for edge ∈ Kanten(node1) do
        if edge.marker == -1 then
            edge.marker = node1.marker + 1;
            for node2 ∈ Knoten(edge) \ node1 do
                if (node2.marker > -1) && (node2.parent != node1.parent) then
                    return H;
                end
                if (node2.parent != node1.parent) then
                    node2.marker = node1.marker + 1;
                    node2.parent = edge;
                    queue.add(node2);
                end
            end
        end
    end
end
```

Über die Nachbarbeziehung werden solange Knoten und Kanten markiert, bis ein Knoten gefunden wird, welcher schon markiert ist. Zu diesem muss es einen alternativen Pfad geben. Das Ergebnis ist ein Hypergraph, in dem ein Zyklus markiert wurde. Damit ein Knoten nicht von der Kante markiert wird, die von ihm markiert wurde, werden alle schon markierten Kanten übersprungen. Außerdem kann die Schnittmenge zweier Kanten mehrere Knoten enthalten. Für diesen Fall wird überprüft, ob die Knoten von der selben Kante markiert wurden. Solche Knoten werden übersprungen.

In diesem markierten Hypergraph soll nun ein einfacher Zyklus gefunden werden. Dafür wird eine spezielle Rekonstruktion angewandt. Diese soll sicherstellen, dass es genau zwei Pfade gibt. Dafür wird folgender Algorithmus eingesetzt:

Algorithmus 2: Rekonstruktion-Algorithmus

```
Data: markierter Hypergraph H = {N, E}
Result: ein Hypergraph bestehend aus einem einfachen Zyklus
e = letzte markierte Kante;
neigh1 = Nachbar(e): mit kleinster Markierung;
neigh2 = Nachbar(e): mit zweit kleinster Markierung;
neigh1.parent = e, neigh2.parent = e;

cycle = {e};
queue = {neigh2, neigh1};

while not empty queue do
    edge = dequeue(queue);
    for neighbour ∈ Nachbarn(edge) do
        if (edge ∈ cycle) && (neighbour != edge.parent) then
            return {N,cycle};
        end
    end
    neighbour = Nachbar(edge) mit kleinster Markierung;
    neighbour.parent = edge;
    queue.add(neighbour);
end
```

Ausgehend von der letzten markierten Kante, werden zwei Nachbarkanten zum Teilgraphen hinzugefügt. Für jede neue Kante wird überprüft, ob sie einen Nachbarn im Teilgraphen hat und falls nicht, wird die Kante mit einer kleineren Markierung hinzugefügt. Das geschieht solange, bis einer der Kanten einen Nachbarn hat. Damit ist der Zyklus geschlossen und kann als Superkante übernommen werden. Damit keine unbeteiligten Kanten hinzugefügt werden, müssen immer erst die Kanten mit größerer Markierung ausgewertet werden.

In [Fag83] wird gezeigt, dass ein Hypergraph mit α-Zyklus auch β-zyklisch ist. Das wiederum bedeutet, dass es einen *graham cycle* gibt, welcher der Definition des einfachen Zyklus entspricht. Es existiert also mindestens ein einfacher Zyklus nach der Graham-Reduktion, welcher durch diese Algorithmen gefunden werden soll.

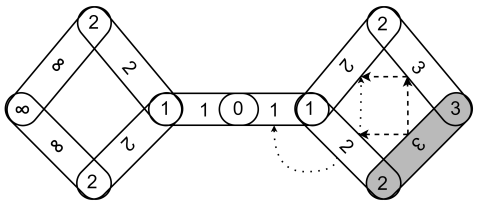


Abbildung 4.6: Markierung und Rekonstruktion

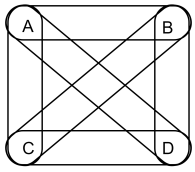
Beispiel 16. Beide Algorithmen werden an einem Beispiel durchgeführt. Dieses entspricht dem Hypergraphen aus Abbildung 4.4a. Der Knoten mit der 0 wurde zufällig als der Startknoten ausgewählt. Dieser gehört nicht zu einem Zyklus. Ausgehend von diesem Knoten werden alle seine Kanten markiert und dann deren Knoten. Das geschieht so lange, bis die graue Kante einen Knoten erneut markieren soll. Diese Kante ist die erste im Rekonstruktion-Algorithmus. Diese und ihre beiden Nachbarn werden zum Ergebnis hinzugefügt. Dann wird für jede Kante eine weitere hinzugefügt, welche eine kleinere Markierung hat. Sobald eine Kante hinzugefügt wird, welche schon einen Nachbarn im erstellten Hypergraphen hat, wird die Rekonstruktion abgebrochen. Das Beispiel zeigt, dass zuerst die Kanten mit der größeren Markierung ausgewertet werden sollten, da sonst unbeteiligte Kanten hinzugefügt werden, welche dann wieder entfernt werden müssen. □

Die Definition 2.3.7 der Azyklität lässt leider kaum einen Rückschluss auf die Struktur eines α-Zyklus zu. Es ist zwar bekannt, dass ein einfacher Zyklus vorhanden ist, aber die Form der restlichen Strukturen nicht. Nach der Entfernung des einfachen Zyklus können diese Strukturen verloren gehen, welche zusätzliche Informationen zu Verbundpfaden enthalten.

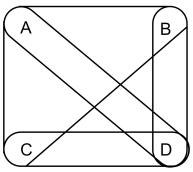
Superkanten in Superkanten

Wird eine Superkante erstellt, die eine andere Superkante ersetzen soll, muss darauf geachtet werden, dass diese Superkante nicht zu groß wird. Je größer die Superkanten werden, desto mehr andere Kanten werden komplett in ihr enthalten sein. Um diesen Effekt zu mindern, kann stattdessen bei komplexen Zyklen der kürzeste Pfad innerhalb der zu ersetzenden Superkante verwendet werden. Damit sind die Kanten des Zykus in mehreren Superkanten.

Beispiel 17. Das Verschlucken von anderen Kanten kann sehr gut am Hypergraphen aus Abbildung 4.7a beobachtet werden. Nach der Erstellung der ersten Superkante entsteht der Hypergraph in Abbildung 4.7b. Es wird ein weiterer Zyklus zwischen der Superkante, {A, D} und {B, D} gefunden.
Wird bei der Erstellung der zweiten Superkante die erste vollständig ersetzt, dann entsteht ein Graph wie in Abbildung 4.8a. Da die Kante {C, D} vollständig in der Superkante enthalten ist, wird sie immer entfernt und nimmt somit nie an einem Verbundpfad teil. Wird stattdessen der kürzeste Pfad der ersten

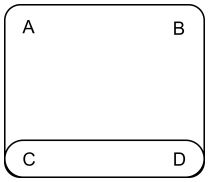


(a) mit zweier Kanten

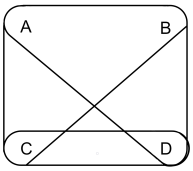


(b) mit einer Superkante

Abbildung 4.7: Vollverbundener Hypergraph



(a) mit einer verschachtelten Superkante



(b) mit zwei Superkanten

Abbildung 4.8: Vollverbundener Hypergraph

Superkante verwendet, dann entsteht ein Hypergraph wie in Abbildung 4.8b. Dieser enthält beide Superkanten. Es handelt sich dabei um einen einfachen Zyklus, welcher vollständig in einer Superkante verwendet wird. Mit dieser dritten Superkante werden wieder alle Verbundpfade gefunden. □

Das Beispiel zeigt, dass die Superkanten β-, γ- oder Berge-Zyklen erzeugen können, die verhindern können, dass Verbundpfade gefunden werden. Das zweite Vorgehen erzeugt zwar mehr Verbundpfade, aber auch nicht alle. Beide Varianten sind nicht ideal und es wird auf jeden Fall ein weiteres Verfahren benötigt, um alle Verbundpfade zu finden. Letztlich zeigt dieses Beispiel, dass Superkanten nur dazu geeignet sind, die Funktionalität der Graham-Reduktion in einem zyklischen Datenbankschema zu ermöglichen. Liegt mehr als ein einfacher Zyklus vor, ist nicht sicher, dass die gesuchten Verbundpfade gefunden werden. Die Entfernung des Zyklus reicht also nicht aus.

4.3.4 Eliminierungshypergraph

Je nach Struktur eines Datenbankschemas können viele Verbundpfade nicht mit der Graham-Reduktion gefunden werden. Vor allem, wenn es große Überlappungen der Kanten gibt, werden die kleineren durch den *edge removal* Schritt entfernt.
Damit diese Verbundpfade nicht verloren gehen, kann mit den entfernten Kanten ein Eliminierungshypergraph (EHG) erstellt werden. Es entsteht ein Hypergraph, welcher alternative Teilpfade besitzen kann. Die Informationen gehen bei der Reduktion also nicht verloren. Im Gegensatz zum originalen Hypergraph wird der EHG nicht komplett verbunden sein und unverbundene Teilgraphen haben. Außerdem können neue α-Zyklen auftreten. Bevor also alternative Pfade ermittelt werden, müssen die Zyklen zuerst zusammengefasst werden. Dann kann überprüft werden, ob Teile des Verbundpfades auch im

EHG erstellt werden können. Dazu müssen alle Knoten ermittelt werden, welche weniger Kanten als im originalen Hypergraphen haben. Das sind alle Knoten der Kanten, die im Verbundpfad sind. Um die verbundenen Teilgraphen des EHG zu finden, werden all diese Knoten als *sacred* markiert und es wird eine Graham-Reduktion durchgeführt. In diesem reduzierten EHG werden dann noch alle Kanten entfernt, welche nur einen Knoten haben, da diese nicht für einen Verbund relevant sind. Dann müssen über die Nachbarbeziehung die jeweiligen verbundenen Teilgraphen ermittelt werden. Jeder dieser Teilgraphen kann einen Teil des Verbundpfades ersetzen. Der jeweilige Abschnitt des Verbundpfades kann mit den *sacred nodes* des Teilgraphen bestimmt werden. Bei der Ersetzung muss darauf geachtet werden, dass die Knoten, für die der Verbundpfad erstellt wurde, nicht entfernt werden. Gibt es einen Zyklus im Hypergraphen, kann es auch sein, dass keine Kanten ersetzt werden, sondern statt der direkten Verbindung zweier Kanten, der alternative Pfad des Zyklus gefunden wird. Dieser sollte jedoch schon bei der Auswertung der Superkante gefunden werden.
Eine einmalige Ausführung ist noch einfach umzusetzen. Das Verfahren muss aber mehrfach durchgeführt werden. Außerdem kann dieser verkleinert werden, wenn es Superkanten gibt. Wie die mehrmaligen Ausführungen durchgeführt und wie die Superkanten berücksichtigt werden müssen, kann aufgrund der fehlenden Zeit leider nicht untersucht werden. Deshalb kann auch keine Aussage darüber getroffen werden, ob alle Verbundpfade mit diesem Ansatz gefunden werden. Trotzdem ermöglicht dieses Vorgehen das Auffinden weiterer Verbundpfade, wenn nicht sogar aller. Damit ist es eine gute Ergänzung, um den Ansatz der Graham-Reduktion vergleichbar mit Paleo-J zu machen. Der EHG eignet sich auch gut in Kombination mit anderen Verfahren, welche einen Pfad zwischen zwei Knoten berechnen.

(a) vor der Reduktion

(b) nach der Reduktion

Abbildung 4.9: ein Eliminierungshypergraph des TPC-H

Beispiel 18. Es wird ein Verbundpfad für das TPC-H-Datenbankschema (Abbildung 6.2) erstellt. Dieser soll *part* und *supplier* verbinden. Ein Ergebnis ist der Verbundpfad {*part,lineitem, supplier*}. Der Hypergraph in Abbildung 4.9a entspricht dem EHG dieses Verbundpfades. Alle Knoten, welche den EHG mit dem Verbundpfad verbinden, sind fett und unterstrichen. Nach der Graham-Reduktion hat der Hypergraph die Form wie in Abbildung 4.9b. Es gibt zwei Teilgraphen, die verbunden sind. Der Teilgraph mit der *partsupp*-Kante (PS) ersetzt im Verbundpfad *lineitem*. Es wird also ein weiterer Verbundpfad gefunden: {*part, partsupp, supplier*}. Der andere Teilgraph ist der Rest des Zyklus. Dieser Teilgraph

erzeugt keinen neuen Verbundpfad, da dieser den zweiten Pfad durch den Zyklus erstellt, welcher schon mit der Superkante gefunden wurde. □

4.4 Entwurf

Wie auch beim Paleo-J Framework, bedarf es einer Analyse des Schemas und der Daten. Dieser Teil wird übernommen. Die Ausgangslage ist also eine Menge von Attributen, welche Kandidaten des Identifikationsmerkmals sind und eine Menge von potentiellen Ranking-Attributen. Gesucht sind alle Verbundpfade, welche mindestens ein Entity- und ein Ranking-Attribut enthalten. Dafür kann die Graham-Reduktion eingesetzt werden. Alle erzeugten Verbundpfade sind dann die Grundlage für eine weitere Verarbeitung. Im nächsten Schritt müssen diese erweitert werden.

Da in dieser Arbeit nur die Verbundpfaderstellung untersucht wurde, fehlt eine Lösung für die Berechnung der Plausibilität. Teile des Entwurfes machen nur dann Sinn, wenn diese Berechnung gut funktioniert. Die Entwicklung solch einer Berechnung benötigt wiederum andere Auswertungen, wie die Erstellung der Anfragen mit unterschiedlichen Aggregaten. Letztlich würde das den zeitlichen Rahmen dieser Arbeit übersteigen. Leider liegt weder eine ausführliche theoretische, noch eine praktische Berechnung der Plausibilität von Paleo vor. Der Programmcode, der zur Verfügung gestellt wurde, ist eine Entwicklungsversion, die auf dem getesteten System leider nicht korrekt funktioniert. Deshalb können auch keine Annahmen dazu getroffen werden, wie sinnvoll die Umsetzung der folgenden Erweiterung ist. Ohne eine sortierte Überprüfung der Anfragen ist es fraglich, ob Mehrfachverwendung von Relationen berücksichtigt werden können. Für die Umsetzung dieser ist die Kenntnis wichtig, ob Anfragen zuverlässig und frühzeitig als potentielle Kandidaten erkannt werden können. Für eine Umsetzung mit einer Erweiterung des Hypergraphen fehlte die Zeit. Deshalb wird an einer geeigneten Stelle in der Erweiterung ein Hinweis für eine Umsetzung mit Mehrfachverwendungen gegeben. Diese ist nur eine Notlösung, nicht optimiert und möglicherweise zu aufwendig.

Wie beschrieben, ist es für das Auffinden der Verbundpfade nötig, die Plausibilitäts- und Ranking-Berechnungen von Paleo-J zu verwenden. Alle folgenden Annahmen setzten voraus, dass der Verbundpfad mit Paleo-J gefunden werden kann und dass die Überprüfung der Aggregatfunktionen korrekt und vollständig ist. Dass die COUNT-Funktion in Paleo nicht berücksichtigt wird, zeigt jedoch, dass schon die Funktionen mit einem einzigen Attribut unpraktisch viele Berechnungen benötigen. Die Komplexität nimmt bei zwei Attributen nur noch weiter zu. In Paleo-J werden nur die Multiplikation und Addition von zwei Attributen untersucht. Es können also nur Anfragen gefunden werden, welche diesen Berechnungen entsprechen. Das Auffinden der Verbundpfade kann also daran scheitern, dass die korrekte Projektion nicht gefunden wurde. Dies ist kein Problem der Verbundpfaderstellung, sondern

der Erstellung und Auswertung der Anfragen.

4.4.1 Erstellung eines Hypergraphen

Als Arbeitsgrundlage für die Graham-Reduktion wird eine genaue Umsetzung des Datenbankschemas in Form eines Hypergraphen benötigt. Dafür werden alle Attribute zu Knoten und die Relationen zu Kanten. Falls Knoten die gleichen Namen haben sollten, werden diese so umbenannt, dass sie einzigartig sind. Nur die Schlüssel und Fremdschlüssel werden gleich benannt, um die Beziehungen der Relationen zu erhalten. Im fertigen Hypergraphen müssen alle gleich benannten Attribute entweder direkte oder indirekte Schlüsselbeziehungen repräsentieren.
Der Hypergraph muss dann auf Azyklität getestet werden. Dafür wird die Graham-Reduktion angewandt. Ist diese nicht erfolgreich, müssen die übrigen Kanten weiterverarbeitet werden. Mit dem Markier- und Rekonstruktion-Algorithmus wird eine Superkante erstellt. Diese ersetzt die Kanten im Hypergraphen. Diese Schritte werden so lange wiederholt, bis die Graham-Reduktion erfolgreich ist. Anschließend kann der Hypergraph genutzt werden, um Verbundpfade zu finden.

4.4.2 Verbundpfade mit Ranking-Kriterium

Die erste Phase der Suche nach Verbundpfaden besteht darin, alle Pfade zwischen den Knoten der Entity- und Ranking-Attribute zu erstellen. Dafür werden jeweils zwei Knoten, welche ein Attribut ihrer Menge darstellen, als *sacred node* markiert. Mittels Graham-Reduktion werden auf dem modifizierten Hypergraphen die Kanten ermittelt, welche die beiden Knoten miteinander verbinden. Dieser reduzierte Hypergraph stellt den Verbundpfad dar und enthält alle Information, um diesen in eine Anfrage umzuwandeln. Sind mehrere einzigartige Ranking-Attribute in einer Relation, reicht es, nur eins zu überprüfen, da die Verbundpfade in solchen Fällen gleich sind. Falls Superkanten enthalten sind, werden diese aufgelöst. Dabei können pro Superkante mehrere Verbundpfade entstehen. Außerdem kann mit dem EHG überprüft werden, ob die Verbundpfade noch alternative Teilpfade besitzen.

Die erstellten Verbundpfade sollten auf ihre Plausibilität überprüft werden, da die Erweiterung der Pfade um Filterrelationen sehr aufwendig ist. Das Ergebnis dieser ersten Phase ist eine Menge von möglichen Verbundpfaden, die entsprechend ihrer Plausibilität bewertet werden. Falls keiner der Verbundpfade das Ergebnis erzeugt, müssen die Verbundpfade noch erweitert werden.

4.4.3 Erweiterung der Verbundpfade

In der zweiten Phase sollen die Verbundpfade so erweitert werden, dass der korrekte Pfad gefunden wird. Dass keiner der Verbundpfade der ersten Phase das Ergebnis erzeugt hat, kann mehrere Gründe haben: Entweder es fehlen noch Filterrelationen in der Selektion oder noch ein weiteres Ranking-Attribut oder eine Relation muss mehrfach verwendet werden. Ersteres ist wahrscheinlich dann der Fall, wenn die Ergebnisse ähnlich der Ergebnisliste sind. So sollten die Werte der Liste bei einem MAX-Ranking immer größer sein. Sind diese kleiner, ist es ein falscher Verbund oder ein falsches Ranking. Ähnliche Regeln lassen sich auch für die anderen Aggregatfunktionen aufstellen. Sollte jedoch keine Anfrage der Ergebnisliste ähneln, dann sind die anderen Gründe wahrscheinlicher.

Alle Ursachen können mit einer schrittweisen Erweiterung der vorhandenen Verbundpfade gelöst werden. Fehlt noch ein Attribut in der Ranking-Funktion, kann es sinnvoller sein, einen Verbundpfad mit einem zweiten Ranking-Attribut mit der Graham-Reduktion zu ermitteln. Dazu müssen statt einem Ranking-Attribut zwei als *sacred* markiert werden. Die erstellten Verbundpfade sollten dann korrekt sein oder müssen noch erweitert werden.

Ähnlich dem Paleo-J-Ansatz, sollen schrittweise Relationen hinzugefügt werden, die gemeinsame Schlüssel mit dem Verbundpfad haben. Im Hypergraphen sind das jeweils die Kanten, welche eine nicht leere Schnittmenge mit der Knotenmenge des Verbundpfades besitzen. Solch eine Erweiterung wird definitiv nach einer gewissen Tiefe beendet. Damit werden alle Verbundpfade ermittelt, welche den Ausgangspfad um Filter erweitern. Da eine Kante mehrere Nachbarn im Verbundpfad haben kann, muss bedacht werden, dass dadurch unterschiedliche Verbundkriterien entstehen können. Eine Lösung wäre, die Filter mit der Kante zu markieren, durch die sie hinzugefügt wurde und dann wie bei Paleo-J, die Kanten nach und nach zusammenzufassen. Das Problem kann aber auch bei der Anfragenerstellung berücksichtigt werden. Bei einer Berücksichtigung von Mehrfachverwendungen müssen auch die Kanten des Verbundpfades verwendet werden. Diese erhalten wie bei Paleo einen Index. Selbst wenn die Verwendung gleicher Relationen limitiert wird, erhöht sich die Menge der erzeugten Verbundpfade exponentiell in der Anzahl der zusätzlich betrachteten Kanten. Dieses Vorgehen ist fast identisch zum Paleo-J-Ansatz.

4.4.4 Erstellung der Anfragen

Prinzipiell ist es leicht, aus einem Hypergraphen eine Datenbankanfrage zu erzeugen. Für die Projektion werden die *sacred nodes* verwendet, da es sich bei diesen um die gesuchten Attribute handelt. Alle Kanten entsprechen einer Relation, deren Verbund über die gemeinsamen Knoten bestimmt wird. Je-

der Knoten kann, entsprechend der Kante aus der er entnommen wird, einem entsprechendem Attribut zugeordnet werden. Eine SQL Anfrage hat die Form:

```
SELECT [sacred nodes]
FROM Kante1, Kante2,...
WHERE P1 and P2...
```

Dabei stehen P's für die Verbundbedingung zwischen den Relationen. Diese setzen die gemeinsamen Attribute gleich. Da im Hypergraphen alle gemeinsamen Attribute erhalten bleiben, liegen diese auch in der Auswertung vor. Da diese mehrere Schlüssel enthalten können, sollten mehrere Anfragen erzeugt werden, die je Teilmengen dieser Schlüssel berücksichtigen. Dieser Fall sollte eher selten eintreten und dann auch nur mit sehr wenigen Schlüsseln. Da die Erstellung des Verbundpfade ungerichtet ist, werden alle möglichen Verbunde erstellt. In einem zyklischen Verbundpfad werden aber nur selten alle Verbunde verwendet. Für diesen Fall müssen alle Anfragen in Betracht gezogen werden, die Teile des Pfades voneinander trennen. Um eine Zyklenerkennung zu vermeiden, sollten stattdessen gerichtete Verbundpfade erstellt werden. Diese sind im aktuellen Entwurf nicht vorgesehen.

Im Kontext der Top-K-Anfragen muss die Projektion mindestens einen Ranking-Wert besitzen. In Paleo-J wird zuerst die hier beschriebene Projektion der möglichen Entity- und Ranking-Attribute durchgeführt und mit dem Ergebnis werden anschließend die unterschiedlichen Ranking-Funktionen überprüft. Entspricht das Ergebnis einer der Funktionen der Ergebnisliste, wird abschließend eine Anfrage erzeugt, die diese Funktion enthält. Es gibt also einen zweistufigen Erstellungsprozess. Da das Ziel dieser Arbeit das Auffinden der Verbundpfade ist, werden nur diese mit denen von Paleo-J verglichen und dessen Anfragenerstellung übernommen.

Kapitel 5

Implementierung

In diesem Kapitel wird der eigene Prototyp vorgestellt. Dies geschieht mittels Ausschnitten des Java-Codes und Pseudocodes. Es wird eine Struktur eines Hypergraphen gezeigt, welche sich für die Graham-Reduktion eignet. Außerdem wird die Implementierung der Superkanten vorgestellt und wie diese gefunden und aufgelöst werden können. Die Umsetzung des EHG liegt nur in einer einfachen, unvollständigen Form vor. Auch die Erweiterung konnte nicht vollständig implementiert werden.

5.1 Graham-Reduktion

Die Struktur von Hypergraphen ist komplexer als die normaler Graphen. Kanten bestehen aus beliebig vielen Knoten. Um das in Java zu repräsentieren, werden die Knoten als Set für jede Kante gespeichert. Da nicht nur die Kanten mit den Knoten verlinkt werden sollen, sondern auch anders herum, wird auch noch eine Funktion für das Entfernen einer Kante benötigt. Die Klasse der Kanten hat somit die Form:

```
class kante {
  hypergraph graph
  knoten[] knoten

  public boolean entferne_aus_knoten() {...}

  //Gibt die Kanten aller Knoten zurück
  public edge[] nachbarn () {...}
}
```

Knoten können in mehreren Kanten enthalten sein. Auch wenn aus der Kantenmenge abgeleitet werden kann, in welchen Kanten sich der Knoten befindet, wird diese Information zusätzlich im Knoten gespei-

chert. Das erleichtert die Reduktion, weil für jeden Knoten die Anzahl seiner Kanten bekannt ist. Hat diese Menge die Größe eins oder kleiner, kann die Kante entfernt werden. Eine zusätzliche Variable **sacred** beschreibt, ob dieser Knoten überhaupt entfernbar ist. Wenn die Variable *true* ist, dann wird dieser Knoten im *node removal* nicht entfernt. Entsprechend werden die Knoten wie folgt implementiert:

```
class knoten {
  kante[] kanten
  boolean sacred

  //Entfernt den Knoten aus allen Kanten
  public void entferne_aus_kanten () {...}
  public boolean einzigartig() {
    return (kanten.size()) <= 1 && !sacred;
  }
}
```

Der Hypergraph besteht aus Kanten und Knoten. Diese werden je als Menge gespeichert. Theoretisch wäre es hier möglich, nur eine der beiden Mengen zu speichern, jedoch erleichtert auch diese Speicherung die Reduktion.

```
class hypergraph {
  kante[] kanten;
  knoten[] knoten;
}
```

Die *node removal* umzusetzen ist leicht. Für jeden Knoten wird überprüft, ob er einzigartig ist. Falls das der Fall ist, wird er aus dem Graphen und allen Kanten entfernt.

```
FOR n ∈ knoten
  IF n.einzigartig() THEN
    knoten.entferne(n)
    n.entferne_aus_kanten()
ENDFOR
```

In der *edge removal* muss für jede Kante überprüft werden, ob diese vollständig in einer anderen enthalten

ist. Um dabei Arbeit zu sparen, wird, statt der Teilmengenüberprüfung von allen Kanten, nach dem Schnitt der verbundenen Kanten gesucht. Dafür werden für jede Kante die Nachbarmengen erstellt und dann geschnitten. Eine Nachbarmenge ist die Menge aller Kanten, welche gemeinsame Knoten mit der Kante haben. Diese kann direkt von jedem Knoten abgefragt werden. Ist der Schnitt nicht leer, bedeutet dies, dass die Kante vollständig in einer anderen enthalten ist und entfernt werden kann.

```
FOR e ∈ kanten
  IF ∩(e.nachbarn).size() > 0 THEN
    kanten.entferne(e)
    e.entferne_aus_knoten();
ENDFOR
```

Für die Ausführung der gesamten Reduktion wird nur eine Schleife benötigt, welche solange ausgeführt wird, bis sich durch *node removal* und *edge removal* keine Änderungen mehr ergeben.

```
boolean changed
public void graham_reduktion() {
  while(changed)  {
    node_removal()
    edge_removal()
  }
}
```

Dies ist eine direkte, aber nicht optimale Implementierung. Statt immer die Schleife zu durchlaufen, sollten nur die Teile betrachtet werden, welche sich geändert haben. Wird eine Kante entfernt, so sollten deren Knoten auf Einzigartigkeit überprüft werden. Wird ein Knoten entfernt, sollten dessen Kanten überprüft werden. Dadurch können vor allem die Teilmengenüberprüfungen der Kanten auf die wesentlichen reduziert werden.

5.2 Superkanten

Superkanten werden als Kanten betrachtet, aber ähneln zugleich einem Hypergraphen. Sie besitzen eine Menge von Kanten, welche sie ersetzt haben, und deren Knoten. In der Graham-Reduktion wird eine Superkante wie eine normale Kante behandelt. Außerdem gibt es eine Funktion, welche Pfade innerhalb der Superkante berechnet. Diese wird eingesetzt, wenn die Superkante in einem Verbundpfad benötigt wird. Die Auflösung entspricht dem Algorithmus aus Abschnitt 4.3.1. Wenn Superkanten Teil anderer

Superkanten sein können, muss die Auflösung solange wiederholt werden, bis keine Superkante mehr im Verbundpfad ist.

```
class superkante extends kante {
  kante[] ersetzte_kanten;

  public kante[] innere_pfade(knoten eingang, knoten[] ausgang) {}
}
```

Um die Zyklen zu finden, wird die Graham-Reduktion auf einer Kopie des Hypergraphen durchgeführt. Bei Misserfolg wird der reduzierte Graph untersucht. Hat jede Kante genau zwei Nachbarn, dann gibt es nur einen einzigen einfachen Zyklus. Falls es Kanten mit mehr Nachbarn gibt, werden die Markier- und Rekonstruktion-Algorithmen aus dem Abschnitt 4.3.3 verwendet. Mit jeder Ausführung wird ein Zyklus aufgelöst. Wenn die Graham-Reduktion erfolgreich ist, kann der Hypergraph verwendet werden, um Verbundpfade zu finden.

```
class hypergraph {
  public void markier( )           //Markier-Algorithmus
  public kante[] rekonstruktion() //Rekonstruktion-Algorithmus
  public void erstelle_superkante(kante[] kanten)

  public void zyklus_entfernen() {
    hypergraph kopie = this.copy();
    if(kopie.graham_reduktion() == false) {
      kopie.markier();
      kante[] zyklus_kanten = kopie.rekonstruktion();
      this.erstelle_superkante(zyklus_kanten);
      this.zyklus_entfernen();
    }
  }
}
```

5.3 Eliminierungshypergraph

Der erste Eliminierungshypergraph entspricht dem originalen Hypergraphen, aus dem alle Kanten des erstellten Verbundpfades entfernt wurden. Alle Knoten, deren Kanten-Anzahl sich verkleinert hat, werden als *sacred* markiert. Zusätzlich müssen im EHG alle Zyklen mit Superkanten ersetzt werden. Nach einer Graham-Reduktion bleiben nur Kanten übrig, die einen *sacred node* enthalten oder an einem Pfad zwischen diesen beteiligt sind. Davon werden alle entfernt, die nur einen Knoten haben. Dann werden alle Kanten über ihre Nachbarschaftsbeziehung in Teilgraphen unterteilt, in denen alle Kanten miteinander verbunden sind. Der Schnitt der Knoten eines solchen Teilgraphen mit den *sacred nodes* ergibt die Knoten, für die der Teilgraph einen alternativen Pfad besitzt. Jeweils zwei der so ermittelten Knoten können einen Teil des Verbundpfades ersetzen. Besitzt der Teilgraph nur zwei dieser Knoten, dann stellt er einen alternativen Teilpfad dar. Sind mehr der *sacred nodes* enthalten, muss je der Verbundpfad von zwei Knoten berechnet werden. Dieser ersetzt den entsprechenden Teilpfad im Verbundpfad. Um diesen zu ermitteln, müssen die selben zwei Knoten markiert und dann die Graham-Reduktion durchgeführt werden. Die übrig gebliebenen Kanten sind der zu ersetzende Teilpfad.
Nach diesen Schritten müsste noch überprüft werden, ob der EHG des ersten EHG noch weitere Pfade enthält. Auf Grund von Zeitmangel wurde jedoch nur eine sehr primitive Variante des EHG umgesetzt. Statt einer korrekten Ersetzung der Teilpfade, werden alle Möglichkeiten erstellt und nachträglich überprüft. Dadurch entstehen auch Verbundpfade, welche zwar korrekt sind, aber Relationen enthalten können, die nicht benötigt werden. Eine korrekte Implementierung würde die entfernten Kanten des *edge removal* zum EHG hinzufügen und die beschriebenen Schritte durchführen. Außerdem kann der Einsatz des EHG noch weiter optimiert werden. Statt einen kompletten Hypergraphen zu erstellen, sollte dieser bei großen Schemata limitiert werden. Hilfreich wäre dafür das Wissen über *articulation sets*, also wichtige Kanten, welche für die Verbindung des Hypergraphen nötig sind.

5.4 Erweiterung um Nachbarkanten

Eine Erweiterung um Nachbarkanten ist prinzipiell einfach, berücksichtigt jedoch nicht die Mehrfachverbindung und hat weitere Probleme. So sollte eine Art gerichteter Verbundpfad erstellt werden. Die gleiche Kante kann also in unterschiedlichen Nachbarschaftsbeziehungen hinzugefügt werden und stellt ein anderes Verbundkriterium dar. Leider ist für die zweite Phase eine Bewertung der Verbundpfade der ersten Phase nötig. Diese fehlt und die verschiedenen Ansätze können deshalb nicht getestet werden. Zwar können mit einer ähnlichen Erweiterung wie in Paleo-J alle Verbunde gefunden werden, dies erfolgt aber mit einem höheren Aufwand. Eine bessere Lösung könnte ein mehrschichtiger Hypergraph sein. Dafür wird der Hypergraph zweimal kopiert und die Relationen miteinander verknüpft. Dadurch entstehen aber sehr viele Zyklen, die aufgelöst werden müssen. Es ist also besser Superkanten hinzuzufügen. Es fehlte jedoch die Zeit, um diese Idee zu testen und umzusetzen. Ein einfache Lösung, die

aber Kanten nur einmal verwenden kann, ist eine simple Erweiterung über Nachbarschaftsbeziehungen. Dieser Ansatz fügt schrittweise benachbarte Kanten hinzu, bis alle Kanten im Verbundpfad enthalten sind. Dadurch ist dieses Verfahren begrenzt. Die Mehrfachverwendung von Kanten wird in diesem Fall nicht berücksichtigt. Auch können Fehler bei der Verbundpfaderstellung auftreten, wenn es Cliquen mit drei oder mehr Kanten oder einen Zyklus gibt. Der einzige Vorteil dieses Ansatzes ist, dass er begrenzt ist. Das spielt in der Praxis aber kaum eine Rolle, da meist nur wenige zusätzliche Relationen benötigt werden. Außerdem werden nicht alle Verbundpfade gefunden.

5.5 Gesamter Prototyp

Mit den beschrieben Verfahren ist es möglich, von einer gegebene Menge von Entity- und Ranking-Attributen eine Menge von Verbundpfaden zu erstellen. Nach der Erstellung des Hypergraphen werden dessen Zyklen behandelt. Dann werden Verbundpfade erstellt, je für ein Paar mit einem Entity- und einem Ranking-Attribut. In diesen können noch Superkanten enthalten sein, die aufgelöst werden. Außerdem wird der EHG eingesetzt, um alternative Pfade zu ermitteln. Für jedes Attributpaar können also mehrere Verbundpfade entstehen. Falls keiner dieser Verbundpfade korrekt ist, müssen die vorhandenen Pfade erweitert werden. Dies geschieht in einer sortierten Reihenfolge, in der zuerst vielversprechende Verbundpfade bearbeitet werden. Der entsprechende Code wird in Anhang B gezeigt.

Kapitel 6

Evaluierung

In diesem Kapitel soll der eigene Ansatz mit Paleo-J verglichen werden. Dafür werden verschiedene Anfragen vorgestellt und die Ergebnisse beider Systeme ausgewertet. Da der eigene Ansatz noch sehr funktional und nicht optimiert ist, wurde auf einen umfassenden Geschwindigkeitsvergleich verzichtet. Auch hatte die getestete Paleo-Version Probleme mit der Datenbank und konnte erst nach ein paar Veränderungen und dann nur teilweise eingesetzt werden. Als Referenz-Datenbankschema für den Vergleich wird das TPC-H-Datenbankschema genutzt. Dieses stellt durch den α-Zyklus eine Herausforderung für einen Ansatz mit der Graham-Reduktion dar. Außerdem werden Anfragen vorgestellt, welche auch in der Praxis eingesetzt werden könnten. Zusätzlich wird noch eine modifizierte Variante des TPC-H verwendet. Die getesteten Anfragen haben verschiedene Schwerpunkte und wurden so erstellt, dass das Paleo-System die Aggregatsfunktion erkennen kann. Für jede Anfrage wurde überprüft, ob die beiden Systeme einen korrekten Verbundpfad finden.

6.1 Vorbereitungen

Vor der Verbundpfaderstellung muss ein entsprechender Hypergraph erstellt werden. Das Datenbankschema des TPC-H eignet sich gut für einen Vergleich, da es mit Paleo-J getestet wurde und außerdem sehr praxisnah sein soll. Da es einen Zyklus enthält, ist es ein Beispiel, das die Schwächen der Graham-Reduktion zeigt. Um den eigenen Ansatz und dessen Umsetzung zu verdeutlichen, werden die einzelnen Vorbereitungsschritte exemplarisch TPC-H demonstriert.

Erstellung des Hypergraphen

Im Datenbankschema des TPC-H ist jedes Attribut einzigartig benannt. Selbst Fremdschlüssel haben immer einen anderen Namen. Dadurch müssen keine Nichtschlüsselattribute umbenannt werden. Alle Fremdschlüssel werden an ihren Primärschlüssel angepasst. Dadurch ändert sich das Schema und entspricht der Abbildung 6.1. Die dabei neuentstandenen Beziehungen zwischen den Relationen werden

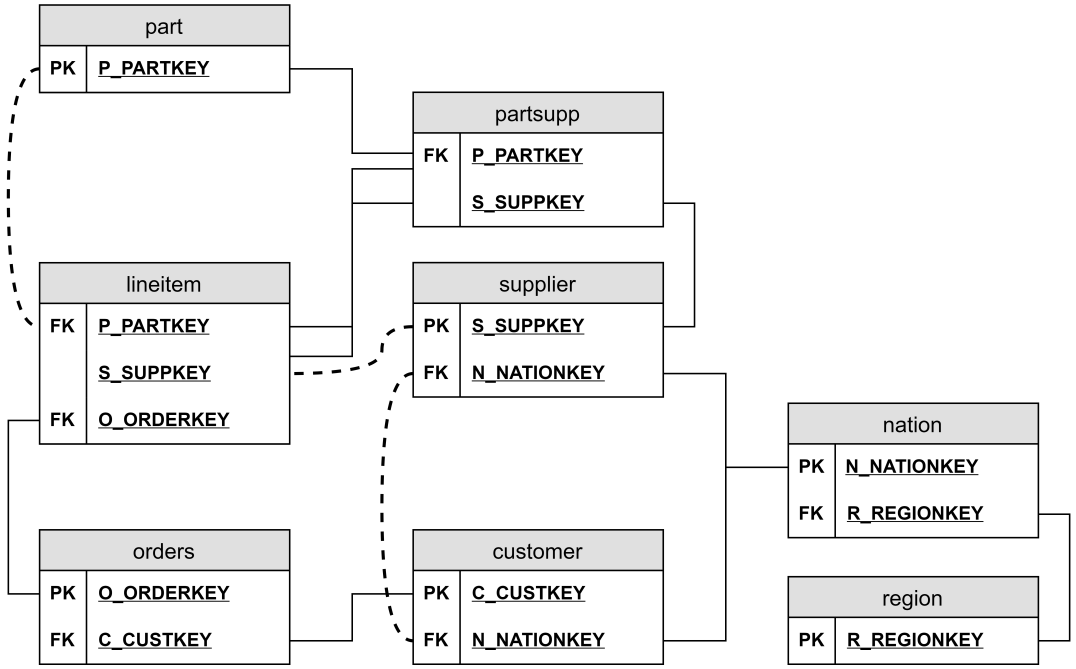


Abbildung 6.1: Verkürztes TPC-H-Datenbankschema nach Umbenennung

mit gestrichelten Linien markiert und die Relationsschemata auf ihre Schlüsselattribute verkürzt. Die Abbildung 6.2 zeigt den dazugehörigen verkürzten Hypergraph. Dort ist gut erkennbar, dass ein Zyklus zwischen *lineitem*, *supplier*, *customer* und *orders* besteht. Wäre statt der Gleichbenennung der Fremdschlüssel je ein extra Knoten hinzugefügt worden, wäre der Zyklus noch um die Kanten *partsupp* und *nation* größer. Im Hypergraphen ist auch gut erkennbar, dass zwischen *partsupp*, *lineitem* und *supplier* kein α-Zyklus besteht, da die Schlüssel von *partsupp* vollständig in *lineitem* enthalten sind.

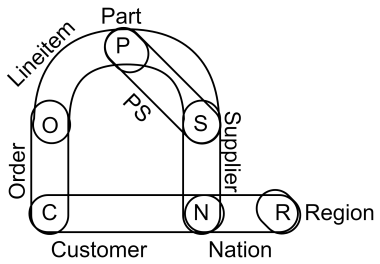


Abbildung 6.2: Verkürzter TPC-H-Hypergraph nach Umbenennung

Superkanten

Um mit dem Schema überhaupt eine Anfrage erstellen zu können, muss noch der Zyklus entfernt werden. Dafür wird die Graham-Reduktion auf dem Hypergraphen durchgeführt. Übrig bleibt der Hypergraph mit

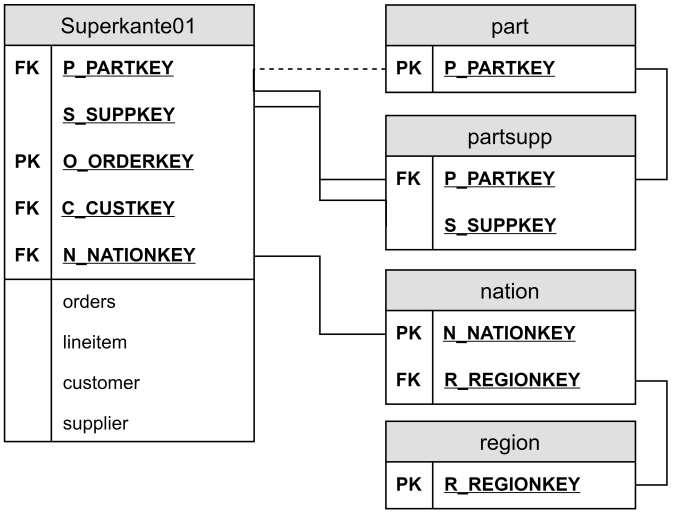


Abbildung 6.3: Verkürztes TPC-H-Datenbankschema mit Superkante

den Kanten *customer*, *orders*, *lineitem* und *supplier*. Jede dieser Kanten hat noch genau zwei Nachbarn. Damit handelt es sich um den beschriebenen einfachen Zyklus. Dieser wird vollständig zu einer Superkante zusammengefügt. Das neue Datenbankschema wird in Abbildung 6.3 dargestellt. In diesem wirkt es, als gäbe es zwischen der Superkante, *part* und *partsupp* einen weiteren Zyklus, aber in Abbildung 6.4 ist deutlich erkennbar, dass diese je eine Teilmenge der Superkante sind und entsprechend im *edge removal* entfernt werden.

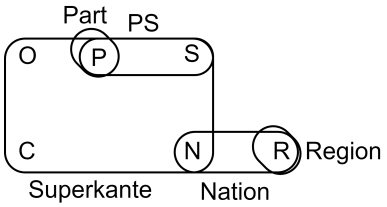


Abbildung 6.4: Verkürzter TPC-H-Hypergraph mit Superkante

Der Hypergraph ist nun azyklisch und kann eingesetzt werden, um Verbundpfade zu ermitteln.

6.2 Anfragen

Leider wurden in [PWM17] keine Beispielanfragen bereit gestellt. Für die Tests wurden daher selbst erstellte Anfragen verwendet, welche denen des TPC-H-Benchmarks ähneln. Da es leider Probleme mit der Auswertung der getesteten Paleo-J-Version gab, wird nachfolgend nur verglichen, ob die Verbundpfade gefunden werden.

Anfrage 1

Die erste Anfrage ist eine sehr simple Top-K-Anfrage. Es sind nur zwei Relationen beteiligt und es wurden keine Filterprädikate eingesetzt.

```
SELECT  s.s_name AS entity, SUM(ps.ps_supplycost)
FROM  partsupp ps, supplier s
```

Paleo-J erstellt und untersucht den korrekten Pfad.
Auch der eigene Ansatz hat den korrekter Pfad ermittelt. Dies geschieht in der ersten Phase der Verbundpfaderstellung.

Anfrage 2

Die zweite Anfrage hat mehr beteiligte Relationen. Sie wurde gezielt ausgewählt, um den Vorteil der indirekten Schlüsselbeziehungen aufzuzeigen.

```
SELECT  o.o_clerk AS entity, AVG(p.p_retailprice) as ranking
FROM  lineitem l, part p, orders o
```

Paleo-J findet einen Pfad, welcher ein korrektes Ergebnis liefert. Dieser enthält aber zusätzlich die Relation *partsupp*. Da nur die direkten Fremdschlüsselbeziehungen genutzt werden, kann nur dieser Pfad gefunden werden. Das Ergebnis wird dadurch nicht beeinflusst, es kann jedoch eine längere Ausführung zur Folge haben.
Vom eigenen Ansatz wird der korrekte Pfad gefunden. Auch in diesem Fall ist es nicht nötig, die Verbundpfade zu erweitern oder einen EHG zu verwenden.

Anfrage 3

Diese Anfrage entspricht einer etwas abgewandelten Form der *query 5* des TPC-H. In der originalen Anfrage wird der *revenue* mit einem Abzug berechnet, welchen das Paleo-System aber nicht testet. In dieser Anfrage gibt es außerdem Filterprädikate.

```
SELECT  n_name,  sum(l_extendedprice) as revenue
FROM  customer, orders, lineitem, supplier, nation, region
WHERE   o_orderdate >= date '1995-10-23' AND r_name = 'AFRICA'
```

Die getestete Paleo-J-Version konnte diese Anfrage nicht finden. Da es sich um eine Anfrage mit vielen Relationen handelt, ist die empfohlene Suchtiefe von fünf nicht ausreichend. Wird die Suchtiefe erhöht, wird der Verbundpfad gefunden.
Das Ergebnis des eigenen Ansatz hängt stark von der Auswertung der Plausibilität und der eingesetzten Erweiterung ab. Ein Verbundpfad, der für *n_name* und *l_extendedprice* erstellt wird, verläuft über *customer, lineitem, nation, orders*. Es fehlen also nur *region* und *supplier*. Diese können in der zweiten Phase hinzugefügt werden. Damit sollte der Pfad theoretisch gefunden werden. Bei der Anfrageerstellung gibt es jedoch ein Problem. Es sind zwar alle Relationen enthalten, aber im Gegensatz zu Paleo-J liegen keine Information vor, welche Relationen verbunden werden müssen. Deshalb werden alle Relationen miteinander verbunden, welche gemeinsame Knoten besitzen. Ein Verbund der *orders*- und *lineitem*-Relation ist jedoch nicht vorgesehen. Im eigenen Ansatz fehlt für diesen Fall noch eine Lösung. So müsste im Erweiterungsschritt gespeichert werden, welche Kanten miteinander verbunden sind. Dadurch würde dieser aber entsprechend komplexer. Alternativ kann auch jeder Verbund innerhalb des Zyklus einmal aufgetrennt werden. Der Verbundpfad kann zwar gefunden, aber die Anfrage noch nicht erstellt werden.

Anfrage 4

Diese Anfrage soll das Problem der verschluckten Kanten verdeutlichen. Das trifft auf die Relation *partsupp* zu, welche ohne *sacred nodes* immer Teilmenge der Superkante ist.

```
SELECT  s_name,  sum(p_size)
FROM  part, partsupp, supplier
```

Paleo-J findet diesen Verbundpfad.
Diese Anfrage zeigt ein Problem der Graham-Reduktion. Der Verbundpfad, welcher der Anfrage am meisten ähnelt, ist *part, lineitem, supplier*. Da *partsupp* nach einem *node removal* vollständig in der Superkante enthalten ist, wird die Kante immer entfernt und ist nie Teil eines erstellten Verbundpfades. Der Verbundpfad kann ohne EHG gefunden werden, wenn sowohl *part*, als auch *partsupp sacred nodes* enthalten sind. Dabei handelt es sich aber um einen Zufall. Mit dem EHG wird ein alternativer Teilpfad gefunden, welcher *lineitem* durch *partsupp* ersetzt und damit den gesuchten Verbundpfad findet. Dies wurde in Beispiel 18 gezeigt.

Anfrage 5

Die Anfrage 5 enthält die *nation*-Relation mehrfach. Da unterschiedliche Nationen gesucht werden, ist es nötig, diese Relation zweimal zu verwenden. Diese Anfrage ist an *query 7* angelehnt.

```
SELECT  s_name,  sum(l_extendedprice) as revenue
FROM  customer, orders, lineitem, supplier, nation n1, nation n2
WHERE   n1.n_name = 'AFRICA' AND n2.n_name = 'GERMANY'
```

Paleo-J findet diese Anfrage ab einer Suchtiefe von sieben und enthält dann die *partsupp*-Relation.
Diese Anfrage ist ein Extrembeispiel für die Probleme des eigenen Ansatzes. *supplier* ist die Relation mit dem Identifikationsmerkmal. Der Verbundpfad dieser Tabelle zu der Ranking-Relation *lineitem* enthält nur diese beiden Relationen. Damit handelt es sich bei den anderen vier um Filterrelationen. Der Verbundpfad hängt stark von der gewählten Erweiterung ab. Leider ist unbekannt, wie sich diese in so einem Fall verhalten würde.

6.3 modifiziertes TPC-H-Datenbankschema

Da ein guter Datenbankentwurf meistens darauf abzielt, ein azyklisches Schema zu entwickeln, wird noch eine modifizierte Variante des TPC-H überprüft. Um den Zyklus aufzulösen, wird der Fremdschlüssel der *nation*-Relation aus *customer* entfernt. Außerdem wurden noch zwei zusätzliche Relationen hinzugefügt. *quality_test* enthält Informationen über Produktbewertungen in Form einer Benotung. Diese wurden von Firmen ausgeführt, welche in *company* enthalten sind. Beide Relationen enthalten je einen Namen und einen Kommentar. Dadurch hat das Schema eine sternartige Form. Durch die Azyklität müssen keine Superkanten erstellt werden.

Anfrage 6

Diese Anfrage enthält nur zwei Relationen, welche für den Verbund benötigt werden. Alle anderen werden nur für die Selektion eingesetzt oder um einen Verbund zu diesen Relationen herzustellen.

```
SELECT  p_name, SUM(ps_availqty) as ranking
FROM  partsupp, part, quality_test, company, supplier, nation
WHERE co_name = 'ADANAC'  AND n_name = 'GERMANY'
```

Paleo-J benötigt auch in diesem Beispiel eine Suchtiefe von sieben, um den korrekten Pfad zu finden.
Für den eigenen Ansatz macht diese Anfrage keinen Unterschied zu Anfrage 5. Durch das azyklische

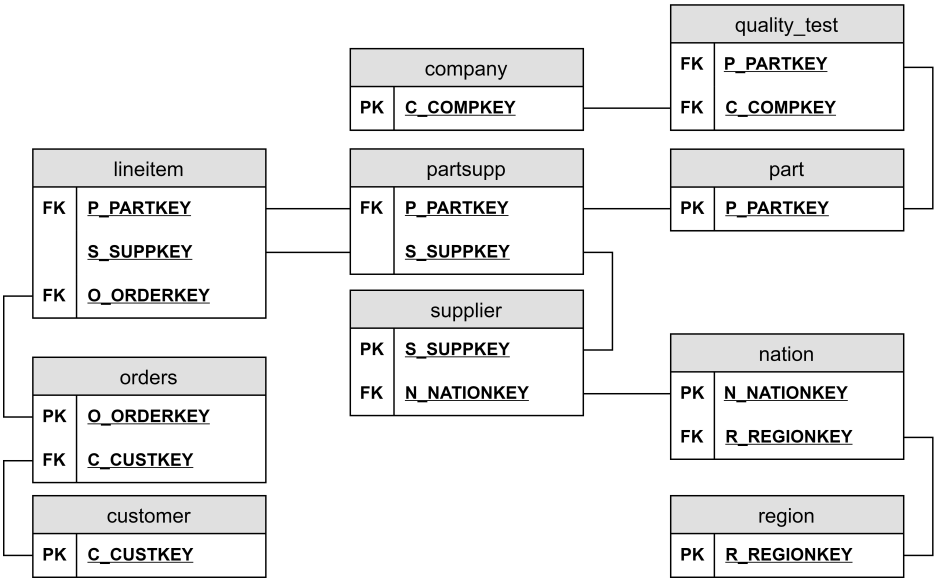


Abbildung 6.5: verkürztes, modifiziertes TPC-H-Datenbankschema nach Umbenennung

Datenbankschema sind kaum Vorteile entstanden. Da es sehr wenige Relationen gibt, ist die Zyklen-Erkennung und -Entfernung kaum ein Problem.

Anfrage 7

Diese Anfrage ist angelehnt an *query 21*. Diese enthält die selbe Relation dreimal. Das ist die maximale Anzahl von Mehrfachverwendung in den TPC-H-Anfragen. l2 und l3 werden als Filter eingesetzt.

```
SELECT  l_commitdate, SUM(l1.l_extendedprice)
FROM  lineitem l1, lineitem l2, lineitem l3
WHERE   ...
```

Obwohl der Verbundpfad nur drei Relationen enthält, wird eine Suchtiefe von fünf benötigt. Es kann ein ähnlicher Verbundpfad erstellt werden. Dieser enthält eine der Nachbartabellen. Das Zusammenfassen im Paleo-J-Ansatz setzt voraus, dass es neben *lineitem* noch eine weitere Relation gibt. Es gibt spezielle Anfragen, die dadurch nicht gefunden werden können. Diese führen den Verbund nicht so aus, wie Paleo-J es vorsieht.
Wie auch in den anderen Fällen, kann dieser Verbundpfad im eigenen Ansatz nur mit der zweiten Phase gefunden werden.

6.4 Ergebnisse des eigenen Ansatzes

Trotz des α-Zyklus kann mit dem eigenen Ansatz die Graham-Reduktion dazu eingesetzt werden, um Verbundpfade für das TPC-H-Schema zu finden. Leider hat die Zeit nicht gereicht, um eine guten Ansatz für *self joins* zu finden. Auch fehlen noch Untersuchungen und Beweise dafür, dass alle Verbundpfade mit den eingesetzten Techniken gefunden werden. Auf einen umfassenden Geschwindigkeitsvergleich zwischen Paleo-J und dem eigenen Prototypen wurde verzichtet, da es technische Probleme mit der Ausführung der getesteten Paleo-J-Version gab. Außerdem würde der entwickelte Prototyp Teile der Aufbereitung und Auswertung des Paleo-J Frameworks nutzen, die nicht für diesen Einsatz optimiert sind. Insgesamt wurde die eigene Software noch nicht optimiert und für einen korrekten Einsatz fehlen noch Funktionen, wie der mehrstufige EHG und die korrekte Erkennung der Verbundattribute in Verbundpfaden mit Zyklus. Nachfolgend werden die beiden Ansätze und ihre Stärken und Schwächen verglichen.

Der eigene Ansatz:
Da die Graham-Reduktion aufwendiger wird, wenn mehr Knoten und Kanten vorhanden sind, ist diese in kleineren Datenbankschemata schneller. Werden diese zu groß, muss in Erwägung gezogen werden, nur Teilgraphen zu berücksichtigen, bei denen alle Kanten direkte oder nahe Nachbarn des Entity-Attributes sind. Die Länge von Verbundpfaden, die in Phase eins gefunden werden, spielt keine Rolle. Ungewiss ist, wie sich der eigene Ansatz in der zweiten Phase im Vergleich zu Paleo-J verhält. Auch wenn zyklische Schemata verarbeitet werden können, benötigen diese zusätzliche Arbeit. Je mehr Zyklen ein Schema enthält, desto aufwendiger wird die Erkennung, da der entwickelte Prototyp nach jeder neuen Superkante überprüft, ob der Hypergraph azyklisch ist. Als eine Lösung ist die gleichzeitige Erkennung mehrerer Zyklen denkbar, die aber weitere Entwicklung benötigt. Das Auffinden der Superkanten und des EHG ist auch abhängig von der Menge der Kanten. Auch hier muss im Zweifel ein Teilgraph erstellt und genutzt werden. Die Mehrfachverwendung von Relationen stellt die klassischen Ansätze vor ein großes Problem. Werden diese berücksichtigt, erweitert sich die begrenzte Menge der Verbundpfade auf unendlich viele. Diese können nicht ohne Umwege berechnet werden und es muss ein Limit eingeführt werden, das den Suchraum beschränkt. Auch ohne Mehrfachverwendung ist die Korrektheit der Verfahren der eigenen Umsetzung nicht bewiesen. Fehler sind vor allem bei vielen Zyklen, die sich überlappen, denkbar. Diese Fälle sind meist unrealistisch und es ist fraglich, ob diese in der Praxis überhaupt auftauchen.

Paleo-J:
Der Ansatz von Paleo-J hat keine Einschränkungen an das Datenbankschema oder die Zusammensetzung des Verbundpfades. Die Menge der Kanten beeinflusst nur die Dauer der *schema exploration* und nicht die Verbundpfaderstellung. Diese ist nur von der Verzweigung der Kanten und der Suchtiefe abhängig. Letztere bestimmt maßgeblich, wie lang die gefunden Verbundpfade sein können. Bei einer Suchtiefe von d können nur Pfade mit bis zu $d + 1$ Relationen gefunden werden. Außerdem ist nur sichergestellt, dass alle Verbundpfade mit $d/2$ Relationen gefunden werden. Das trifft dann zu, wenn die Startrelation

der Suche mindestens $d/2$ unterschiedliche Nachbarrelationen besitzt. Dabei handelt es sich zwar um ein Extrembeispiel, zeigt aber die Abhängigkeit von der Verzweigung der Relationen. Dabei ist irrelevant, welche Rolle die Relationen im Verbundpfad spielen. Entscheidend ist letztlich vor allem die Suchtiefe. Ist diese zu hoch, werden zu viele Verbundpfade berechnet. Da die Suche mit Δ^d (Δ ist der Faktor der Verzweigung) exponentiell wächst, ist es wichtig, diese so niedrig wie möglich zu halten. Ist sie jedoch zu niedrig, wird der Verbundpfad nicht gefunden. Damit tritt die klassische Frage auf, ob nicht tief genug gesucht wurde oder die Anfrage nicht gefunden werden kann. Da nicht auszuschließen ist, dass die Anfrage ein Ranking nutzt, welches nicht überprüft wird, muss abgewogen werden, ob eine tiefere Suche sinnvoll ist.

Vergleich von Paleo-J mit dem eigenen Ansatz:
Beide Ansätze unterscheiden sich in ihrer Herangehensweise und bieten unterschiedliche Vor- und Nachteile. Der *join tree* von Paleo-J muss immer erstellt und ausgewertet werden. Der eigene Ansatz sucht hingegen erst die vielversprechenden Verbundpfade und versucht, diese zu erweitern. Der Vorteil des eigenen Ansatzes liegt vor allem in Anfragen, die keine Filterrelationen enthalten. Diese werden sehr schnell in der ersten Phase gefunden und das unabhängig von ihrer Länge. Da Paleo-J alle möglichen Verbundpfade berechnet, spielt nur die Menge und nicht die Funktion der Relationen ein Rolle. Sind mehrere Filterrelationen oder mehrfach verwendete Relationen im gesuchten Verbundpfad, dann ist der eigene Ansatz stark davon abhängig, ob früh der richtige Kandidat gefunden werden kann. Dadurch ist die Erweiterung weniger aufwendig, da nur von einem oder wenigen Verbundpfaden ausgegangen werden muss. Leider fehlen die Informationen, wie gut solch eine Überprüfung in diesem Kontext arbeitet. Für Paleo-J ist die Größe des Datenbankschemas fast irrelevant, während der Berechnungsaufwand der Graham-Reduktion mit dieser zunimmt.
Wie beschrieben, wurde kein umfassender Geschwindigkeitstest durchgeführt. Dennoch lassen die Ausführungszeiten vermuten, dass zumindest die erste Phase mit der Graham-Reduktion schneller sein kann. In Anfrage 4 dauerte die *join tree*-Erstellung etwa 6 Sekunden, mit dem *chainbuilding* etwa 7 Sekunden. Der eigene Ansatz hat für die Untersuchung der Zyklen und die Erstellung der Verbundpfade mit EHG etwa 0.06 Sekunden benötigt. Da die Paleo-J-Pfade schneller in Anfragen umgewandelt werden können und die eigene Umsetzung noch unvollständig ist sowie nicht alle Verbundpfade findet, ist ein direkter Vergleich unter diesen Umständen nur bedingt aussagekräftig. Außerdem ist das TPC-H-Datenbankschema relativ klein im Gegensatz zu produktiven System. Es handelt sich also um ein Szenario, welches den eigenen Ansatz deutlich bevorzugt. Dennoch lassen diese Zahlen vermuten, dass ein Ansatz mit Graham-Reduktion durchaus schneller sein kann. Auch interessant ist, dass Paleo-J für die Anfrage 4 nur sehr wenige SQL-Anfragen testet. Obwohl die Anfragen eine der wenigen ist, bei der die getestete Version korrekt funktionierte, heißt das, dass eine Erweiterung basierend auf der Plausibilitätsberechnung gut funktionieren könnte. Zusätzlich wurde der eigene Ansatz mit einem Datenbankschema getestet, welches dem modifizierten TPC-H entspricht, nur mit siebenfach kopierten Knoten und Kanten.

Statt der 0.06 Sekunden wurden etwa 0.23 Sekunden benötigt. Das Schema hat 80 Relationen und keine Zyklen. Die Verlangsamung war zu erwarten und zeigt auch, dass der eigene Ansatz ab einer gewissen Anzahl von Relationen nur mit einer Teilgraph-Untersuchung Sinn macht.

Analyse der TPC-H-Anfragen:
Im eigenen Ansatz können momentan nicht alle Verbundpfade gefunden werden. Um die Menge der gefundenen besser bestimmen zu können, wurden die TPC-H-Anfragen analysiert. Diese sollen Datenbanksysteme testen und haben deshalb gezielt unterschiedliche Schwerpunkte. Dennoch liefern sie einen guten Eindruck über mögliche Anfragen. Für die Untersuchung sind mehrere Faktoren von Interesse: die Anzahl der verwendeten Relationen, der Filterrelationen und der Anfragen mit Mehrfachverwendungen. Im Schnitt sind etwa 4 Relationen an Verbund und Selektion beteiligt. Immerhin 7 der 22 Anfragen verwenden sechs oder mehr Relationen. Das sollten die einzigen sein, die Paleo-J Probleme bereiten. Dieses kann davon höchstens einen finden, wenn die voreingestellte Suchtiefe von fünf verwendet wird. In 12 Anfragen wurde mindestens eine Filterrelation verwendet. Davon waren 7 nur mit einer zusätzlichen Relation. Wiederum 5 von diesen 7 kann der eigene Ansatz selbst mit einer simplen Erweiterung leicht finden. In 8 Anfragen wurden Relationen mehrfach verwendet. Diese können nur mit einer besseren Erweiterung gefunden werden. Da die Doppelungen meist in der Selektion verkommen, besteht die Möglichkeit, dass es äquivalente Anfrage gibt, die alle Relationen nur einmal enthalten. Diese wiederum können auch mit einer einfachen Erweiterung gefunden werden.
In 8 der 22 Anfragen waren alle Relationen an der Projektion oder am Verbund beteiligt. Das sind Verbundpfade, welche in Phase eins gefunden werden. Alle weiteren Verbundpfade können nur in Phase zwei gefunden werden. Wird von einer einfachen Erweiterung ausgegangen, die nur Nachbarn hinzufügt, können weitere 5 Anfragen leicht gefunden werden, da diese nur eine Filterrelation enthalten. Lässt die Erweiterung Relationen mehrfach zu, dann können weitere 3 Anfragen, die nur eine Relation mehrfach enthalten, auch leicht gefunden werden. Insgesamt ist davon auszugehen, dass mit einer guten Umsetzung der Erweiterung 16 der 22 Anfragen schnell gefunden werden können. Die restlichen Anfragen enthalten mehr als eine Filterrelationen oder eine doppelte Relation. Diese sind stark abhängig von einer frühen Kandidaten-Ermittelung und deshalb können keine Aussagen über diese getroffen werden.

Kapitel 7

Zusammenfassung und Ausblick

Nach einer Untersuchung des Paleo-J-Ansatzes und einem theoretischen Vergleich mit der Graham-Reduktion, sprechen mehrere Gründe gegen den Einsatz der klassischen Verbundpfaderstellungen im Reverse Engineering von Anfragen. Ein Ansatz mit der Graham-Reduktion hat zwei große Probleme. Zum einen wurde diese für Universalrelationen entwickelt und war ursprünglich für die Erkennung von Zyklen gedacht. Dadurch können unter anderem nur azyklische Datenbankschemata genutzt werden. Zum anderen sind *self joins* nicht vorgesehen. Nur mit diesen Einschränkungen kann das Verfahren eingesetzt werden und sinnvolle Ergebnisse liefern. Bei einem Vergleich der beiden Ansätze mit dem Datenbankschema des TPC-H, würde die Graham-Reduktion nicht einen einzigen korrekten Verbundpfad finden. Ähnlich verhalten sich auch die anderen klassischen Ansätze.

Dennoch wurde in dieser Arbeit versucht, einen vergleichbaren Ansatz zu Paleo-J zu entwickeln. Dafür ist es nötig, auch beliebige Datenbankschemata verarbeiten zu können. Für diesen Zweck wurden Superkanten vorgestellt, welche α-Zyklen entfernen. Dadurch ist die Ausführung der Graham-Reduktion auch unabhängig von der Struktur des Schemas möglich. Mit dem Eliminierungshypergraphen können alternative Verbundpfade gefunden werden, welche sonst unauffindbar wären. Leider fehlt noch eine geeignete Lösung, um Relationen mehrfach zu berücksichtigen. Durch die Umsetzung der vorgestellten Techniken, konnte der eigene Prototyp viele Verbundpfade finden. Auch wenn dies nicht alle Verbundpfade waren, so war er teilweise deutlich schneller als die getestete Paleo-Version.

Das Ziel eines vergleichbaren Ansatzes wurde also nur teilweise erreicht. Neben dem Hinzufügen von Filterrelationen fehlt noch, dass *self joins* berücksichtigt werden. Auch ist eine Überarbeitung der Verbundpfaderstellung nötig, um die Verbundprädikate korrekt zu ermitteln. Außerdem muss getestet werden, ob die Plausibilitätsberechnung, wie in Paleo, mit solch einem Ansatz möglich ist. Diese entscheidet maßgeblich darüber, ob der Geschwindigkeitsvorteil wirklich nutzbar ist. Offen ist auch der

Einfluss von sehr großen Datenbankschemata und wie mit diesen umgegangen werden muss.

Die Graham-Reduktion hat andere Anforderungen und auch die Ergebnisse unterscheiden sich grundsätzlich von Paleo-J. Ein vergleichbarer Einsatz kann nur mit weiteren Verfahren erreicht werden. Die vielen Einschränkungen und der Aufwand diese zu überwinden, lassen die Graham-Reduktion als ungeeignet erscheinen. Dennoch zeigt der theoretische und praktische Vergleich dieser Arbeit, dass diese auch gewisse Vorteile hat und eine weitere Untersuchung lohnenswert ist.

Literaturverzeichnis

[Bra16] BRAULT-BARON, JOHANN: *Hypergraph Acyclicity Revisited.* ACM Comput. Surv., 49(3):54:1–54:26, 2016.

[CZ91] CHEUNG, TO-YAT und YUN-ZHOU ZHU: *Recognizing Different Types of Beta-Cycles in a Database Scheme*. Theor. Comput. Sci., 81(2):295–304, 1991.

[Fag83] FAGIN, RONALD: *Acyclic Database Schemes (of Various Degrees): A Painless Introduction*. In: AUSIELLO, GIORGIO und MARCO PROTASI (Herausgeber): *CAAP'83, Trees in Algebra and Programming, 8th Colloquium, L'Aquila, Italy, March 9-11, 1983, Proceedings*, Band 159 der Reihe *Lecture Notes in Computer Science*, Seiten 65–89. Springer, 1983.

[FMU82] FAGIN, RONALD, ALBERTO O. MENDELZON und JEFFREY D. ULLMAN: *A Simplified Universal Relation Assumption and Its Properties*. ACM Trans. Database Syst., 7(3):343–360, 1982.

[Gra79] GRAHAM, M.H.: *On the Universal Relation*. Technical Report, Univ. of Toronto, Canada, 1979.

[Gys86] GYSSENS, MARC: *On the Complexity of Join Dependencies*. ACM Trans. Database Syst., 11(1):81–108, 1986.

[Hal86] HALL, GARY W: *Querying Cyclic Databases in Natural Language*. Doktorarbeit, Theses (School of Computing Science)/Simon Fraser University, 1986.

[Mai83] MAIER, DAVID: *The Theory of Relational Databases*. Computer Science Press, 1983.

[MHH93] MANEGOLD, STEFAN, CARSTEN HÖRNER und ANDREAS HEUER: *Rückblick auf IRIS-Abschlußbericht eines Datenbank-Forschungsprojektes*. Technical Report, TU Clausthal, 1993.

[MSH17] MEYER, HOLGER, ALF-CHRISTIAN SCHERING und ANDREAS HEUER: *The Hydra.PowerGraph System - Building Digital Archives with Directed and Typed Hypergraphs*. Datenbank-Spektrum, 17(2):113–129, 2017.

[MU83] MAIER, DAVID und JEFFREY D. ULLMAN: *Maximal Objects and the Semantics of Universal Relation Databases*. ACM Trans. Database Syst., 8(1):1–14, 1983.

[MU84] MAIER, DAVID und JEFFREY D. ULLMAN: *Connections in Acyclic Hypergraphs*. Theor. Comput. Sci., 32:185–199, 1984.

[PM16] PANEV, KIRIL und SEBASTIAN MICHEL: *Reverse Engineering Top-k Database Queries with PALEO*. In: PITOURA, EVAGGELIA, SOFIAN MAABOUT, GEORGIA KOUTRIKA, AMÉLIE MARIAN, LETIZIA TANCA, IOANA MANOLESCU und KOSTAS STEFANIDIS (Herausgeber): *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, Seiten 113–124. OpenProceedings.org, 2016.

[PWM17] PANEV, KIRIL, NICO WEISENAUER und SEBASTIAN MICHEL: *Reverse Engineering Top-k Join Queries*. In: MITSCHANG, BERNHARD, DANIELA NICKLAS, FRANK LEYMANN, HARALD SCHÖNING, MELANIE HERSCHEL, JENS TEUBNER, THEO HÄRDER, OLIVER KOPP und MATTHIAS WIELAND (Herausgeber): *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme"(DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, Band P-265 der Reihe *LNI*, Seiten 61–80. GI, 2017.

[TPC] *Transaction Performance Council website (TPC).* `http://www.tpc.org`.

[YO79] YU, C. T. und M. Z. OZSOYOGLU: *An algorithm for tree-query membership of a distributed query*. In: *COMPSAC 79. Proceedings. Computer Software and The IEEE Computer Society's Third International Applications Conference, 1979.*, Seiten 306–312, Nov 1979.

[ZEPS13] ZHANG, MEIHUI, HAZEM ELMELEEGY, CECILIA M. PROCOPIUC und DIVESH SRIVASTAVA: *Reverse engineering complex join queries*. In: ROSS, KENNETH A., DIVESH SRIVASTAVA und DIMITRIS PAPADIAS (Herausgeber): *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Seiten 809–820. ACM, 2013.

Anhang A

Tableau-Beispiel

Nachfolgend wird das Tableau für des Beispiel 7 gezeigt. Für jedes Attribut wurde eine Spalte erstellt. Dabei wurden nur die Schlüssel und die gesuchten Attribute berücksichtigt, da das Tableau sonst zu groß werden würde. Für jede Relation wird dann eine Zeile erstellt, welche ausgezeichnete Variablen in den Zellen der enthaltenen Attribute hat. Diese bestehen aus einem a mit Index. Innerhalb einer Spalte haben die ausgezeichneten Variablen den selben Index.

Relation	O	O1	C	C1	N	N1	R	S1	S	R1
orders	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$							
customer			$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	$\boldsymbol{a}_5$					
nation					$\boldsymbol{a}_5$	$\boldsymbol{a}_6$	$\boldsymbol{a}_7$			
supplier					$\boldsymbol{a}_5$			$\boldsymbol{a}_8$	$\boldsymbol{a}_9$	
region							$\boldsymbol{a}_7$			$\boldsymbol{a}_{10}$

Die restlichen Zellen werden mit nicht ausgezeichneten Variablen belegt. Diese sind alle einzigartig.

Relation	O	O1	C	C1	N	N1	R	S1	S	R1
orders	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	b_1	b_2	b_3	b_4	b_5	b_6	b_7
customer	b_8	b_9	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	$\boldsymbol{a}_5$	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}
nation	b_{15}	b_{16}	b_{17}	b_{18}	$\boldsymbol{a}_5$	$\boldsymbol{a}_6$	$\boldsymbol{a}_7$	b_{19}	b_{20}	b_{21}
supplier	b_{22}	b_{23}	b_{24}	b_{25}	$\boldsymbol{a}_5$	b_{26}	b_{27}	$\boldsymbol{a}_8$	$\boldsymbol{a}_9$	b_{28}
region	b_{29}	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	$\boldsymbol{a}_7$	b_{35}	b_{36}	$\boldsymbol{a}_{10}$

Um nun einen Verbundpfad mit dem Tableau zu erstellen, werden alle ausgezeichneten Variablen, welche nicht für die Projektion relevant sind, durch nicht ausgezeichnete ersetzt. In diesem Fall sind die Projektionsattribute c1 und s1. Diese entsprechen den *sacred nodes* in der Graham-Reduktion und dürfen nicht ersetzt werden. Die b-Variablen hingegen können in einer Tableau-Abbildung, auch Homomorphismus

genannt, durch eine andere ersetzt werden. Ziel ist es, ein minimales Teiltableau zu erzeugen, welches äquivalent zum Ausgangstableau ist.

Relation	O	O1	C	C1	N	N1	R	S1	S	R1
orders	b_{37}	b_{38}	b_{39}	b_1	b_2	b_3	b_4	b_5	b_6	b_7
customer	b_8	b_9	b_{39}	$\boldsymbol{a}_4$	b_{40}	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}
nation	b_{15}	b_{16}	b_{17}	b_{18}	b_{40}	b_{41}	b_{42}	b_{19}	b_{20}	b_{21}
supplier	b_{22}	b_{23}	b_{24}	b_{25}	b_{40}	b_{26}	b_{27}	b_{44}	b_{45}	b_{28}
region	b_{29}	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{42}	b_{35}	b_{36}	$\boldsymbol{a}_{10}$

Alle Zeilen, die durch eine Abbildung einer anderen gleichen, können entfernt werden. Dieser Vorgang entspricht dem *edge removal*.

Relation	O	O1	C	C1	N	N1	R	S1	S	R1
customer	b_8	b_9	b_{39}	$\boldsymbol{a}_4$	b_{40}	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}
nation	b_{15}	b_{16}	b_{17}	b_{18}	b_{40}	b_{41}	b_{42}	b_{19}	b_{20}	b_{21}
region	b_{29}	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{42}	b_{35}	b_{36}	$\boldsymbol{a}_{10}$

Außerdem können alle b-Variablen, die nur einmal vorkommen, entfernt werden. Das entspricht dem *node removal*. Aus dem gekürzten Tableau können dann alle Informationen für eine Anfrage abgelesen werden. Die a-Variablen werden in der Projektion und die b-Variablen für den Verbund verwendet.

Relation	C1	N	R	R1
customer	$\boldsymbol{a}_4$	b_{40}		
nation		b_{40}	b_{42}	
region			b_{42}	$\boldsymbol{a}_{10}$

Anhang B

Prototyp Code

```
1  class hypergraph {
2
3    public verbundpfad[] erstelle_verbundpfade (knoten[] sacreds) {
4      hypergraph hg = this.copy();
5      hg.sacreds = sacreds;
6      hg.graham_reduktion();
7      queue<hypergraph> auflösen_queue = {hg};
8      verbundpfad[] gefundene_pfade = {};
9      while(auflösen_queue not empty) {
10       hypergraph hg2 = auflösen_queue.poll();
11       if(hg2.enthält_supperkante) {
12         auflösen_queue.add(hg2.superkante_auflösen());
13       }
14       else {
15         gefundene_pfade.add(hg2, hg2.ehg_pfade());
16       }
17     }
18     return gefundene_pfade;
19   }
20
21   //Bewertet die Verbundpfade, gibt einen Verbundpfad zurück, wenn
     dieser den maximalen Score hat
22   public verbundpfad bewerte_pfade (verbundpfad[] alle_pfade)
23
```

```
24   public verbundpfad finde_verbundpfade (knoten[] idents, knoten[]
     rankings ) {
25     verbundpfade[] alle_pfade = [];
26     //Phase 1
27     for(knoten ident in idents) {
28       for(knoten rank in rankings) {
29         hypergraph hg = this.copy_with_sacreds({ident, rank});
30         hg.graham_reduction;
31         Verbundpfade
32         alle_pfade
33       }
34     }
35     verbundpfad erfolg_pfad = bewerte_pfade(alle_pfade);
36     if(erfolg_pfad != null) return erfolg_pfad;
37
38     //Phase 2
39     int getestete_anfragen = 0;
40     while(getestete_anfragen < LIMIT_ANFRAGEN) {
41       verbundpfad[] erweitert = alle_pfade.first().erweiterung();
         //QUERY...
42       alle_pfade.addALL(pfad.erweiterung());
43       verbundpfad erfolg_pfad = bewerte_pfade(alle_pfade);
44       if(erfolg_pfad != null) return erfolg_pfad;
45     }
46
47   }
48 }
```

Selbständigkeitserklärung

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

Rostock, den 27. Mai 2019