Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Datenbanken und Informationssysteme
Prof. Dr. rer. nat. Volker Markl

Bachelorarbeit

GoOLAP
User Interaktion

Sebastian Arnold (310221)
sarnold@mailbox.tu-berlin.de

26.07.2011

Gutachter:
Prof. Dr. rer. nat. Volker Markl
Prof. Dr. Odej Kao
Betreuer:
Dr. Alexander Löser

Eigenständigkeitserklärung

Diese Arbeit ist Teil des Projektes „GoOLAP“ am Fachgebiet Datenbanksysteme und Informationsmanagement (DIMA) und wurde in Teilen als Demo-Paper unter dem Titel „GoOLAP: Interactive Fact Retrieval from Search Engines“ [2] zur CIKM2011 Glasgow, UK eingereicht und erscheint in Teilen im Rahmen der eBISS Summer School 2011 unter dem Titel „Web Scale Business Analytics“ in „Lecture Notes in Business Information Processing“. Inhalte dieser Arbeit wurden in Verbindung mit dem Projekt GoOLAP auf der „Langen Nacht der Wissenschaften“ am 28.05.2011 in der Technischen Universität Berlin präsentiert.
Die selbstständige und eigenhändige Ausfertigung versichere ich an Eides statt.

Berlin, den 26.07.2011

_____

(Sebastian Arnold)

Zusammenfassung

Bei der Suche nach Informationen im Web verfolgt ein Benutzer keinen geradlinigen Pfad durch eine gegebene Navigationsstruktur, sondern er verfolgt eine komplexe Suchstrategie über verschiedene Quellen und Navigationsebenen hinweg. Das Projekt GoOLAP verfolgt das Ziel, die im Web in Textform vorliegenden unstrukturierten Daten aufzuspüren, in eine strukturierte Faktenbasis zu extrahieren und darauf komplexe Anfragen zu ermöglichen. Dafür bietet es ein Web-Frontend mit einer Reihe von Operatoren für interaktive explorative Suche. Die vorliegende Arbeit hat zum Ziel, die Benutzerinteraktion mit diesem System zu nutzen, um die GoOLAP-Faktenbasis hinsichtlich ihrer Qualität und Abdeckung zu verbessern. Wir untersuchen dafür den Prozess der explorativen Suche und ermitteln typische Suchintentionen des Benutzers. Da der Benutzer dem System seine Suchintention explizit durch Interaktionen mitteilt, implementieren wir zur Erkennung der Intentionen eine detaillierte Protokollierung der Benutzerinteraktion. Wir werten die Protokolle in einem strukturierten parallel verteilten Prozess aus. Dabei beobachten wir typische Interaktionsmuster, mit deren Hilfe wir das Interesse an Objekten, Fakten und deren Beziehungen quantifizieren können und Hinweise auf ungesehene Fakten erhalten. Die Ergebnisse der Auswertung können wir verwenden, um die Faktenbasis mit einem nutzungsbasierten Ranking zu versehen und den Prozess der Faktenbeschaffung aus dem Web zielgerichtet auf ungesehene Fakten mit hohem Benutzerinteresse zu lenken.

Abstract

When searching the Web for information, the user does not follow a straight path inside a fixed navigational structure. Instead, he uses a complex search strategy, switching between different sources and navigation levels. The project GoOLAP aims at the retrieval and extraction of unstructured Web text data to a structured fact base that allows complex queries. GoOLAP provides a Web frontend with a set of operators for interactive exploratory search. The objective of this thesis is to utilize user interaction on the system to improve the quality and coverage of the fact base. Therefore we study the process of exploratory search and identify characteristic user search intentions. As the user explicitly communicates his search intention to the system by interacting with it, we implement detailed logging of user interaction in order to detect his intentions. We analyze the log files in a structured parallel distributed process. Thereby we observe typical interaction patterns, which we use to measure the interest in objects, facts and their relationships and to discover potentially unseen facts. We can later use the results of this analysis to augment the fact base with usage-based rankings und efficiently pilot the fact retrieval process to unseen facts with high user interest.

Inhaltsverzeichnis

1. Einführung

In diesem Kapitel geben wir zur Einführung einen Überblick über den Begriff der explorativen Suche. Wir zeigen, wie ein Benutzer mit einem explorativen Suchsystem interagiert und motivieren unser Vorgehen zur Analyse von Benutzerinteraktion. Abschließend geben wir einen Überblick über die Gliederung der vorliegenden Arbeit.

1.1. Interaktive explorative Suche im Web

Bei der Suche nach Informationen im Web verfolgt ein Benutzer keinen geradlinigen Pfad durch eine gegebene Navigationsstruktur. Seine Suchstrategie ist zu komplex, um sie als linearen Prozess zu definieren. Stattdessen nutzt er auf vielfältigste Weise die Möglichkeiten des Browsings über verschiedene Quellen und Navigationsebenen hinweg. Als Einstiegspunkt dienen oft die Ergebnislisten einer Suchmaschine für einfache, kurze Suchanfragen oder thematisch sortierte, oft von Benutzern generierte Verzeichnisse wie z.B. Wikipedia[1], Freebase[2] oder CrunchBase[3]. Aus diesen Ergebnissen erhält der Benutzer weitere Quellen oder Kandidaten für neue Suchanfragen. Die gewünschten Informationen bekommt er dabei nicht auf einer finalen Ergebnisseite präsentiert, sondern er sammelt sie auf seinem Pfad durch die verschiedenen Quellen allmählich zusammen – sein Ergebnis ist eine Aggregation über viele Teilergebnisse [3]. Diesen iterativen Prozess aus Anfragen und ergebnisoffenem Browsing nennen wir *explorative Suche* (vgl. [25]).

Der Benutzer teilt dem System seine Intentionen explizit durch Interaktionen mit, z.B. stellt er eine textuelle Anfrage über ein Suchobjekt an eine Suchmaschine und erhält eine von seiner Eingabe abhängige Antwort. Im Erfolgsfall enthält die Antwort neue Informationen über das Suchobjekt, meist zusammen mit redundanten Informationen, die bereits aus vorangegangenen Anfragen bekannt sind. Die neue Information richtet die Suchstrategie des Benutzers neu aus: sie wird erweitert oder spezifiziert, so dass er seine Suche nachfolgend breiter gestalten oder vertiefen kann. Durch neue Erkenntnisse – unabhängig davon, ob sie zufällig entdeckt oder zielgerichtet angefragt wurden – entsteht eine beidseitige Interaktion zwischen Benutzer und Webseite.

Die beschriebene Strategie besteht oft aus Selektion, Navigation und *Trial-And-Error* und ist im Web weit verbreitet. Der Benutzer erwartet von einem guten Suchsystem, dass er innerhalb des Systems explorativ interagieren kann [17]. Um diese Erwartung zu erfüllen, werden *Information Retrieval* (IR) Systeme mit *Human-Computer Interaction* (HCI) verwendet, deren Verknüpfung zu interaktiven IR-Methoden unser Forschungsthema darstellt.

1.2. Problemstellung

Wir entwickeln im Umfeld dieser Arbeit das IR-System GoOLAP, welches das Ziel verfolgt, die im Web in Textform vorliegenden unstrukturierten Daten aufzuspüren, in eine Faktenbasis zu extrahieren und dem Benutzer für komplexe strukturierte Anfragen zur Verfügung zu stellen. Dafür bietet GoOLAP ein Web-Frontend mit einer Reihe von Operatoren, die dem

[1] http://en.wikipedia.org
[2] http://www.freebase.com
[3] http://www.crunchbase.com

Benutzer eine interaktive explorative Suche auf der Faktenbasis ermöglichen. Einige Operatoren, darunter Funktionen zur Bewertung und Editierung von Fakten, wurden im Vorfeld dieser Arbeit neu entwickelt.

Dem System liegt ein Prozess zugrunde, der neue Fakten mit einer Crawlingstrategie aus dem Web in die strukturierte Faktenbasis extrahiert. Da dieser Retrieval-Prozess mit sehr hohen Kosten in der Ausführung verbunden ist, versuchen wir, diese Kosten zu minimieren. Diese Arbeit verfolgt den Ansatz, Benutzerinteraktion zu beobachten und mit den daraus resultierenden Erkenntnissen den Retrieval-Prozess gezielt auf für die Benutzer interessante ungesehene Fakten zu richten [2]. Z.B. können wir beobachten, welche Fakten oft nachgefragt werden, welche Fakten nicht korrekt sein könnten, und welche Fakten möglicherweise in einer bisher nicht erkannten Relation stehen. Unser Ziel ist es dabei, maschinell reproduzierbare Analysen zu entwickeln, welche später in den automatisierten iterativen Retrieval-Prozess eingebunden werden können.

1.3. Vorgehen und Gliederung der Arbeit

In der vorliegenden Arbeit analysieren wir den Prozess der explorativen Suche systematisch und entwickeln eine Methode, ihn aufzuzeichnen, darzustellen und zu evaluieren. Die Arbeit gliedert sich in drei Hauptteile, in denen unterschiedliche aufeinander aufbauende Aspekte der Problemstellung beleuchtet werden. Nach der Einführung über explorative Suche in Kapitel 1 folgt eine Übersicht über verwandte Arbeiten und bestehende Erkenntnisse in der Forschung in Kapitel 2. Der Hauptteil beginnt mit dem Konzept über die Definition von Interaktionen in einem IR-System in Kapitel 3. Wir werden die Interaktionen kategorisieren und in atomare Events zerlegen, die wir später messen können. Außerdem definieren wir Qualitätsmaße, um den Einfluss der Interaktion auf den Datenbestand messbar zu machen. Das zentrale Kapitel 4 beschäftigt sich mit der Umsetzung der Protokollierung von Benutzerinteraktion und ihrer Analyse auf großen parallel verteilten Systemen im Hinblick auf die ständige Verbesserung der Faktenbasis . Dabei werden die zuvor definierten Events auf Graphen abgebildet, mit deren Hilfe wir Navigationstrukturen innerhalb der Benutzerinteraktion erkennen können. In Kapitel 5 evaluieren wir die Erkenntnisse aus den vorausgehenden Kapiteln anhand realer Daten unseres Prototypen GoOLAP, die über den Zeitraum von zwei Monaten erhoben wurden. Wir evaluieren qualitative und quantitative Fragestellungen und zeigen auf, welche Maßnahmen möglich sind, um die Benutzerinteraktion in den Prozess zum Aufbau der Faktenbasis einfließen zu lassen. Zum Schluss fassen wir die Ergebnisse in Kapitel 6 zusammen und geben einen Ausblick über die weitere Entwicklung des interaktiven IR-Systems GoOLAP.

2. Verwandte Arbeiten

In diesem Kapitel geben wir einen Überblick über die bisherige Forschung in den Bereichen Web-Suche, Interaktion mit explorativen Suchsystemen und Analyse von Interaktion auf Webseiten. Wir werden uns im Verlauf der vorliegenden Arbeit auf die vorgestellten Konzepte und Begriffe beziehen. Wir stellen außerdem kurz einige Web-Systeme vor, die ähnliche Ansätze für die Verknüpfung von IR mit Benutzerinteraktion verfolgen.

2.1. Prozess der Web-Suche

Der Prozess der Suche wird schon für frühe IR-Systeme in Bibliotheken erforscht und die Ergebnisse sind in ihren Grundlagen auch relevant für die Suche nach Informationen im Web. Wir unterscheiden dabei den Prozess der Web-Suche in zwei Dimensionen: (1) auf der Ebene der Navigation und Interaktion in einem Suchsystem und (2) auf der Ebene der gesuchten Inhalte.

Bates [4] untersucht den Anteil von Benutzerinteraktion an einem Suchprozess und die Anzahl an Aktivitäten dabei. Sie beschreibt eine vierstufige Hierarchie: *Move* (eine einzelne Aktion als Teil der Informationssuche), *Tactic* (einige Aktionen, die die Inhalte der Suche vorantreiben), *Stratagem* (eine komplexere Folge von Aktionen, die die Struktur der Suche erweitern) und *Strategy* (ein festgelegter Plan für den gesamten Suchprozess).

Marchionini [17] beschreibt in detaillierter Form den Prozess der explorativen Suche und erkennt dafür wichtige Operatoren, die meist der Klasse der Stratagems angehören. Er beschreibt die Interaktionen während einer Suche als die Aktivitätstypen *Lookup* (nachschlagen), *Learn* (erschließen) und *Investigate* (erforschen). Lookup ist die grundlegende Aufgabe, die Datenbanksysteme und Suchmaschinen primär beherrschen. Das Ergebnis sind meist wohlstrukturierte, singuläre Daten. Learn und Investigate sind dagegen typisch explorative Aktivitäten, die das System dem Nutzer (in der Regel) nicht abnehmen kann. Durch die oftmals nicht-lineare Navigationsstruktur des Benutzers (z.B. führt er verschiedene Suchpfade parallel in mehreren Browser-Tabs aus, oder verschachtelt mehrere Suchanfragen ineinander) verschwimmen die Grenzen zwischen den Aktivitätstypen.

Aula and Russell [3] unterscheiden zwischen explorativer Suche und *komplexer* Suche aufgrund des Grades der Abstraktion und der Anzahl der Bewegungen. Sie bewegen sich damit zwischen den anfangs vorgestellten Klassen Tactic und Stratagem.

Auf inhaltlicher Ebene entwickelt Broder [7] ein Modell von drei Klassen von Suchintentionen: *Navigational* (das zielgerichtete Erreichen einer Seite), *Informational* (die Beschaffung von Information aus einer oder mehreren Quellen) und *Transactional* (die Nutzung eines Web-typischen Services).

Rose and Levinson [21] verknüpfen diese Klassen mit dem Bedürfnis, ein der Suche zugrunde liegendes Ziel zu erfüllen. Sie untergliedern die Klasse Informational feiner und führen an Stelle von Transactional eine breiter angelegte Klasse *Resource* (die Beschaffung und Verwendung von Ressourcen und Anwendungen im Web) ein.

Wir werden die vorgestellten Begriffe für verschiedene Klassen von Anfragen, Aktivitäten und Zielen im Verlauf der Arbeit wieder aufgreifen und benutzen.

2.2. Human Computer Interaction bei explorativen Suchintentionen

Hearst et al. [10] entwickeln aus der Beobachtung von verschiedenen Typen von Suchanfragen ein User-Interface zur Unterstützung von explorativer Suche. Sie untersuchen die *Usefulness* der verfügbaren Operatoren anhand der Häufigkeit ihrer Nutzung.

White et al. [25] erkennen, dass es keine geeigneten Metriken gibt, um den Grad der Unterstützung für explorative Suche in IR-Systemen zu beurteilen. Stattdessen schlagen sie vor, diese Beurteilung den Benutzern zu überlassen. Aus den Erkenntnissen und Modellen über Suchprozesse und Nutzung lassen sich bessere Interfaces für explorative Suche zu entwickeln, die eine engere Verknüpfung von HCI und IR erreichen [17, 26, 27].

Die vorliegende Arbeit verfolgt das Ziel, HCI in strukturierter Form zu erfassen und zu analysieren, um zielgerichtetes IR nach den Bedürfnissen der Benutzer zu ermöglichen.

2.3. Analyse von Interaktion auf Webseiten

Card et al. [8] führen eine Methode zur Analyse von Webprotokollen und der Visualisierung ihrer Nutzung als *Web Behaviour Graphs* ein. Sie beschreiben neben verschiedenen Techniken der Beobachtung (z.B. Videoaufzeichnung und Sprachprotokolle) die Protokollierung von Nutzungsdaten in Logfiles und der Analyse von *Clickstreams*.

White and Drucker [24] untersuchen die Variabilität von Verhaltensmustern bei einer Websuche. Sie unterscheiden dabei zwei Dimensionen der Nutzung (analog zur Unterscheidung in Abschnitt 2.1): die Ebene der verwendeten Operatoren und Seitentypen und die Ebene der Inhalte der eingegebenen Queries. Sie untersuchen auf jeder Ebene jeweils die Variabilität der Interaktionen eines einzelnen Benutzers und die Variabilität der Interaktionen aller Benutzer untereinander.

Wir werden die Vorgehensweise dieser beiden Studien im Verlauf der Arbeit ebenfalls aufgreifen und erweitern, und dabei die Graph-Notation übernehmen.

2.4. Existierende Projekte

Wir bieten einen kurzen Überblick über im Web existierende Projekte, die verschiedene Konzepte zur Unterstützung von explorativem Browsing, User-Feedback und benutzergesteuerten Einflüssen auf IR demonstrieren.

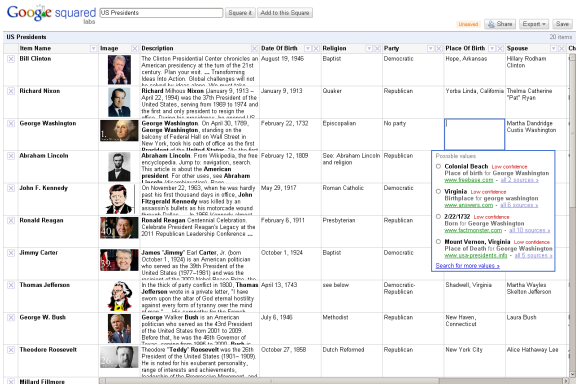


Abbildung 2.1.: Google Squared

Die kommerzielle Suchmaschine Google Squared[1] (Abbildung 2.1) extrahiert ihre Faktenbasis maschinell aus Informationen aus Tabellen und Listen aus den von Google indizierten Seiten im Web. In tabellarischer Form kann der Benutzer die Faktenbasis explorativ erforschen, indem er interaktiv Zeilen (Objekte) und Spalten (Attribute) zu einer Anfrage (z.B. „US Presidents“) hinzufügt, die dann automatisch mit Werten gefüllt werden.

[1] http://www.google.com/squared/

Die Webseiten Factual[2] und TrueKnowledge[3] integrieren neuartige, an die Suchintentionen angepasste Interfaces mit direktem User-Feedback in Form von Bewertung, Validierung und Editierfunktionen zum manuellen Einfügen von Informationen. Beide basieren zu einem Großteil auf angereichertem *User Generated Content*. Während das Interface von Factual rohe Daten in tabellarischer Form präsentiert und direkt editierbar macht, versucht TrueKnowledge textuelle Fragen der Benutzer mit Steckbriefartigen Ergebnisseiten zu beantworten.

2.5. Zusammenfassung

Die verwandten Arbeiten bieten die Grundlage für unser weiteres Vorgehen. Wir verwenden die Erkenntnisse aus vorausgehenden Arbeiten, um unsere eigenen Konzepte zu strukturieren: wir nutzen im Folgenden die hier eingeführten Begriffe zur Kategorisierung von Aktivitäten hinsichtlich ihrer Länge (*Tactic*, *Stratagem*, *Strategy*), ihrer Komplexität (*Lookup*, *Learn*, *Investigate*) und ihrer inhaltlichen Intention (*Navigational*, *Information*, *Resource*). Die Erkenntnisse über HCI und die Analyse von Benutzerinteraktion bilden die Grundidee dieser Arbeit. Wir orientieren uns für die Weiterentwicklung des GoOLAP-Frontends auch an den sichtbaren Konzepten für explorative Interaktionsmöglichkeiten in den vorgestellten existierenden Projekten. Im folgenden Kapitel werden wir die hier vorgestellten Erkenntnisse weiterentwickeln und das Konzept dieser Arbeit vorstellen.

[2] http://www.factual.com
[3] http://www.trueknowledge.com

3. Benutzerinteraktion bei explorativer Web-Suche

Ein Benutzer drückt seine Suchintentionen während einer explorativen Web-Suche durch eine Folge von Interaktionen mit dem System aus. In diesem Kapitel entwickeln wir ein Konzept, welches diese Interaktionen als iterative Prozesse für den Benutzer (Suche) und im System (Retrieval) beschreibt. Die Interaktionen im Suchprozess folgen typischen Mustern, welche wir messen und bewerten wollen, um den Retrieval-Prozess gezielt zu steuern. Dazu definieren wir Qualitätsmaße zur Bewertung von Relevanz, Korrektheit, Vollständigkeit, Interesse und Abdeckung und formalisieren Benutzerinteraktionen als Events, die wir später protokollieren und analysieren können. Unser Ziel ist, die beiden Prozesse zu einem Kreislauf zu vereinen, indem wir die Benutzerinteraktion dazu verwenden, die Qualität und Abdeckung der Faktenbasis zu erhöhen.

3.1. Welche Interaktionen sind typisch für explorative Web-Suche?

Wir betrachten den in Abschnitt 1.1 vorgestellten Prozess der explorativen Suche am Beispiel einer Web-Suchmaschine und erläutern einige typische Suchintentionen und ihre Übersetzung in Interaktionen.

Suchintentionen eines Benutzers. Bei einer Web-Suche kann der Benutzer verschiedene Suchintentionen verfolgen. Sie reichen von der direkten Suche nach einer bestimmten atomaren Information bis hin zu informellem Browsing über einem Themengebiet oder einer komplexen Fragestellung. Jede Intention lässt sich in einzelnen, klar definierten Anfragen oder abstrakt als Informationsbeschaffung ausdrücken. Z.B. können wir eine Web-Suchmaschine benutzen, um (in Anlehnung an [10]) folgende Fragen zu beantworten:

(1) Wie viele Mitarbeiter hat die Firma Boeing?

(2) Welche europäischen Firmen stellen Flugzeuge her?

(3) Welche Freizeitsportarten werden in meiner Gegend angeboten, bei denen man das Fliegen lernt?

(4) Welchen Zusammenhang gibt es zwischen zunehmendem Flugverkehr und der Erderwärmung?

Übersetzung der Intentionen in Interaktionen. Suchsysteme wie z.B. die Web-Suchmaschine Google[1] bieten eine Reihe von Interaktionsmöglichkeiten an. Die beschriebenen Intentionen teilt der Benutzer dem unterliegenden System explizit durch eine Folge von in diesem System möglichen Aktionen mit.

(1) Im ersten Fall genügt dazu eine einzelne Suchanfrage mit wenigen Keywords, die zu einem Dokument mit der gewünschten Information führt (z.B. `boeing "number of employees"`).

(2) Im zweiten Beispiel wird es im Allgemeinen nicht ausreichen, ein einzelnes Dokument zu finden. Stattdessen führt der Benutzer einen iterativen Suchprozess aus, in dem er analog nach

[1] http://www.google.com

einzelnen Flugzeugherstellern sucht und die Ergebnisse manuell zu einer Liste zusammenstellt. Dabei verzweigt sich seine Suche möglicherweise, da die Information über die Herkunft einer Firma nicht immer im selben Dokument zu finden sein wird.

(3) Während das Erstellen einer Liste von Flugzeugherstellern noch als einfache Informationsbeschaffung bezeichnet werden kann, geht das dritte Beispiel einen Schritt weiter. Die Freizeitangebote in einer Stadt lassen sich ohne Zuhilfenahme eines vorher erstellten durchsuchbaren Verzeichnisses nicht einfach auflisten. Selbst mithilfe einer solchen Liste liegt es dann am Benutzer selbst, seine persönliche Suche nach passenden Sportarten einzugrenzen. Dazu muss er durch die Ergebnisse browsen und sich selbst einen Überblick verschaffen. Er wird sich dabei möglicherweise entscheiden, ob er lieber motorisiert fliegen oder den Gleitflug lernen möchte und wird seine Suche entsprechend anpassen. Stößt er auf ein Wassersportangebot in der Nähe, könnte er sich möglicherweise auch für Wasserskifahren entscheiden und so seine Suchintention entscheidend verändern. Der Suchprozess hat kein bestimmtes Ende, sondern dauert so lange an, bis der Benutzer die Entscheidung trifft, dass er genügend Ergebnisse gefunden hat. Der Prozess folgt keiner Ordnung (z.B. vom Überblick zum Detail), sondern verzweigt sich und bewegt sich explorativ in verschiedene Richtungen.

(4) Das vierte Beispiel soll aufzeigen, wie weit eine Suchintention über reine Informationsbeschaffung hinausgehen kann. In einem solchen Prozess muss es möglich sein, Argumentationsketten zu verfolgen, Vergleiche vorzunehmen, Zusammenzufassen und die gefundenen Informationen auf verschiedenste Weise detailliert zu verarbeiten [10].

Einordnung der Beispiele in Aktivitätsklassen. Zusammenfassend können wir diese Beispiele in die im Abschnitt 2.1 vorgestellten Aktivitätsklassen von Marchionini [17] einordnen: Beispiel (1) beschreibt die Aktivität *Lookup*, bei der der Benutzer Fakten abfragt und verifiziert, nach bekannten Objekten sucht oder eine Frage beantworten möchte. Beispiel (2) beschreibt die Aktivität *Learn*, bei der der Benutzer einzelne Anfragen zur Wissensbeschaffung verbindet und die Ergebnisse zusammenfasst, aggregiert und integriert. Beispiel (3) beschreibt die Aktivität *Investigate*, bei der der Benutzer Ergebnisse anhäuft und sie anschließend analysiert, bewertet und transformiert. Er erforscht das Themengebiet dabei immer weiter und stößt auf Ergebnisse, die außerhalb der ursprünglichen Suchintention liegen und seine Suchstrategie neu ausrichten. Im Beispiel (4) verwendet der Benutzer alle beschriebenen Aktivitäten, um Informationen zu erhalten, die durch seine eigene kognitive Leistung weiterverarbeitet werden können.

Beobachtung von Interaktionsmustern. Aus der Folge von Interaktionen eines Benutzers mit dem Suchsystem lassen sich typische Muster erkennen, die auf einen bestimmten Benutzertyp oder eine bestimmte Aktivität hindeuten können. White and Drucker [24] schlagen anhand der Erkenntnisse ihrer empirischen Studie zwei extreme Benutzerklassen vor: *Navigators* (Navigierer) und *Explorers* (Stöberer). Sie beobachten die Variabilität in Interaktionsmustern ausgehend von den Ergebnisseiten einer Keyword-Suche. Die Interaktionsfolgen der Navigators weisen eine niedrige Variabilität auf. Sie folgen für unterschiedliche Suchobjekte und selbst bei unterschiedlichen Suchintentionen immer einem konsistenten Muster von ähnlichen Seitenaufrufen und ähnlicher Länge. Die Interaktionsfolgen der Explorers dagegen sind von hoher Variabilität. Sie weisen selbst für einfache Aktivitäten wie Lookup einen hohen Verzweigungsgrad über verschiedene Ergebnisseiten auf und sind unterschiedlich lang.

Wir können demnach von den Navigators lernen, wie sich bestimmte Suchintentionen abkürzen lassen, indem wir komplexe Operatoren anbieten, die ihre typischen Interaktionsmuster nachbilden und eine tiefe Suche mit wenigen Schritten ermöglichen. Von den Explorers können wir Erkenntnisse über die Diversität der Quellen folgern und die Breite der Suchergebnisse erhöhen.

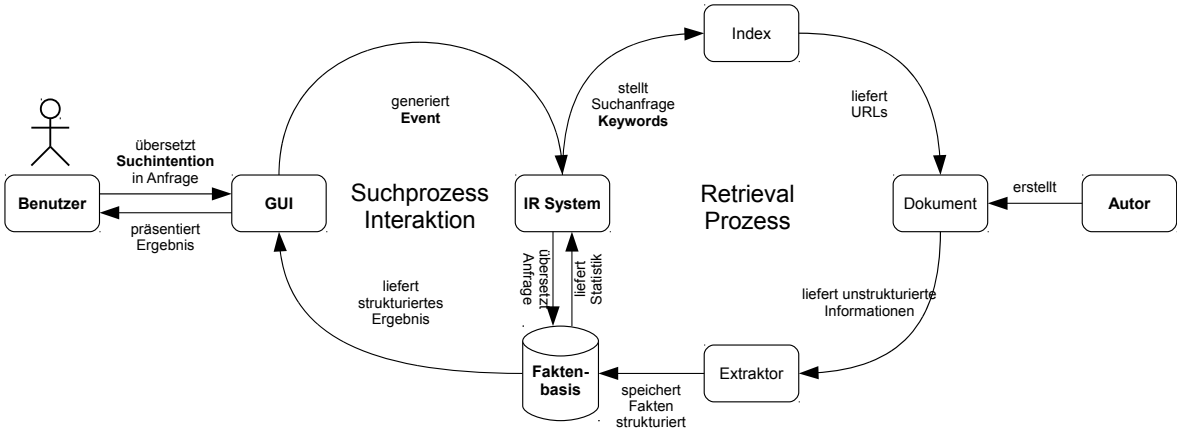


Abbildung 3.1.: Suchprozess und Retrieval-Prozess in einem Information Retrieval System

3.2. Definition und Kategorisierung der beteiligten Prozesse

Wir können in einem IR-System zwei Prozesse beobachten: die Anfragen eines Benutzers an das Suchsystem und die Auswertung der Anfragen durch das unterliegende System. Typischerweise wird die Auswertung nicht in Echtzeit durchgeführt, sondern Anfragen werden bis zu einem gewissen Grad vorberechnet, z.B. indem wir in einem Datenbanksystem eine indizierte Faktenbasis vorhalten, welche schnelle Anfragen auf einem repräsentativen Korpus ermöglicht. Abbildung 3.1 zeigt ein typisches Web Information Retrieval System, welches Suchanfragen von einem Benutzer annimmt und ein strukturiertes Ergebnis zurückliefert. Da die Informationen im Web in unstrukturierter Form als Text-Dokumente vorliegen, wird der Prozess der Auswertung als Retrieval-Prozess mit Faktenextraktion umgesetzt. D.h. die verfügbaren Informationen werden im Voraus angefragt, berechnet und strukturiert in einer Datenbank abgelegt. Der Suchprozess erhält seine Ergebnisse so direkt aus einer Datenbankabfrage.

Der Suchprozess beschreibt die Suchintention eines Benutzers als iterative Folge von Aktionen auf den Operatoren des IR-Systems. Dabei steuert der Benutzer über Events auf dem Graphical User Interface (GUI) die Anfragen an die Faktenbasis und setzt damit indirekt den Retrieval-Prozess in Gang. Als Ergebnis übersetzt das GUI das strukturierte Ergebnis in eine grafisch aufbereitete Ergebnisseite, die dem Benutzer angezeigt wird.

Der Retrieval-Prozess hat zum Ziel, die Faktenbasis mit ungesehenen Fakten durch automatisierte Anfragen anzureichern. Damit erhöhen wir die Abdeckung der im Web vorhandenen Dokumenten in unserer Datenbank. Je mehr für den Benutzer relevante Fakten aus diesen Dokumenten extrahiert werden können, umso höher wird die Qualität unserer Faktenbasis. Wir stellen in Abschnitt 3.3 Metriken vor, um diese Qualität messbar zu machen. Wir verfolgen in dieser Arbeit den Ansatz, diese Qualitätsmaße zu erhöhen, indem wir Benutzerinteraktion beobachten. So können wir z.B. ungesehene Fakten durch Analyse von Interaktion ermitteln und im Retrieval-Prozess dann gezielt nach diesen Fakten suchen.

Kategorisierung nach Nutzen. Wir wollen für jede Interaktion untersuchen, welchen erwartbaren Nutzen sie für den Benutzer und für das System hat. Die beiden Prozesse sind nur dann optimal ausgelastet, wenn möglichst viele Interaktionen sowohl dem Benutzer als auch dem System einen positiven Nutzen bieten. Das heißt, im optimalen Fall sind die Operatoren des Systems für den Benutzer attraktiv und generieren gleichzeitig neue Informationen für das unterliegende System.

Broder [7] erkennt, dass der positiven Nutzen für den Benutzer eines IR-Systems nicht allein in der Erfüllung seines Informationsbedürfnisses liegt. Der Benutzer verfolgt mit seiner Suchintention weitere Ziele, die an [7] anknüpfend in [21] zu den bereits vorgestellten Kategorien entwickelt werden: *Navigational* (Navigation), *Informational* (Information), *Resource* (Quellen und Angebote). Desweiteren wird in [19] die Motivation zum Beitrag von *User Generated Content* am Beispiel der Online-Enzyklopädie Wikipedia als Benutzerziel erkannt. Wir können zusammenfassend von positivem Nutzen sprechen, wenn eine Interaktion dem Benutzer zur Erfüllung eines dieser Ziele dient.

Das IR-System bietet mit seiner strukturierten Faktenbasis eine bessere Möglichkeit zur Quantifizierung von Nutzen. Wir werden im folgenden Abschnitt Qualitätsmaße definieren, mit denen wir positiven Nutzen für das System erkennen können. Auf einer höheren Ebene betrachtet, können wir von positivem Nutzen sprechen, wenn eine Interaktion das Interesse an einem Objekt oder Fakt messbar macht, sie die Korrektheit von Inhalten bewertet, sie unvollständige Ergebnisse aufzeigt oder sie Inhalte generiert.

3.3. Bestimmung der Qualität und Abdeckung der Faktenbasis

Für eine spätere Analyse von Benutzerinteraktion mit IR-Systemen definieren wir Metriken zur Bestimmung der Qualität und Abdeckung der Faktenbasis. Diese Metriken helfen uns, verschiedene Teilaspekte des IR-Systems und des Interaktionsprozesses zu quantifizieren und zu evaluieren. Zwar sind einige der verwendeten Kriterien nicht numerisch ausdrückbar, doch für unseren Zweck genügt es oft, relative Werte oder Tendenzen festzustellen. Absolute Zahlenwerte sind also meist nicht notwendig. Wir werden später Beziehungen zwischen Benutzerinteraktionen und den hier vorgestellten Metriken aufbauen. Da wir die Metriken verwenden wollen, um Benutzerinteraktion als positive Auswirkung auf die Faktenbasis zu erkennen, liegt unser Hauptaugenmerk dabei auf der Entwicklung von qualitätsbasierten Metriken [18]. Wir betrachten als Ausgangswerte die messbare Verteilung von Häufigkeiten in der Faktenbasis und definieren Verteilungen pro Fakt und pro Fakt-Typ. Weitere Gruppierungen (z.B. pro Objekt, pro Objekt-Typ, pro Zeitabschnitt) können wir analog zu den hier angegebenen Ausdrücken definieren. Dabei seien:

t	ein Fakt-Typ, repräsentiert durch eine festgelegte Menge von Attributen, deren Wert leer (`null`) oder gesetzt (`not null`) sein kann
$Typ(t)$	die Menge der Attribute des Fakt-Typs t
$F = \bigcup_{\forall t} F_t$	die Menge aller Fakt-Instanzen in der Faktenbasis vom Typ t, mit $count(f, F) = \lvert\{i \in F \lvert f(i) = f\}\rvert$
$i \in F$	die Instanz eines Faktes, die an einer Stelle in einem Dokument gefunden wurde. Der gleiche Fakt kann ein- oder mehrmals als Instanz in F vorkommen
$f \in distinct(F)$	ein distinkter Fakt aus der Menge von Fakt-Instanzen ohne Duplikate, somit repräsentiert durch eine konkrete Belegung der Attribute, mit $distinct(F) = \{f(i) \lvert i \in F\}$
$f(i)$	eine surjektive Abbildung von einer Fakt-Instanz i auf ihre Repräsentation als Fakt f

Relevance (Relevanz) pro Fakt beschreibt die relative Häufigkeit einer Fakt-Instanz in der Gesamtmenge von Fakt-Instanzen in einem Korpus. Z.B. wird der Fakt „Airbus competes Boeing“ von vielen verschiedenen Autoren genannt, er hat also eine hohe Relevance. Ähnlich zu TF-IDF [23] beschreibt dieses Maß die *Interestingness* [15], eines Faktes in Bezug auf die

Gesamtmenge von Fakt-Instanzen des selben Typs. Wir definieren die Relevance eines Fakts als:

$$relevance(f,t) = \frac{count(f, F_t)}{|distinct(F_t)|}$$

Correctness (Korrektheit) pro Fakt beschreibt, ob die in einem Fakt enthaltene semantische Information richtig oder falsch ist. Z.B. ist der Fakt „A330 is product of company Airbus“ korrekt und der Fakt „A330 is product of company Boeing“ nicht korrekt. Da wir die Korrektheit nicht direkt aus den Daten ablesen können, müssen wir als Approximation andere Quellen als Indikator für Korrektheit verwenden, z.B. die Metrik Relevance oder Benutzer-Feedback.

Completeness (Vollständigkeit) pro Fakt beschreibt, wie hoch der Anteil der gesetzten Attribute in einem Fakt ist. Z.B. fehlen dem Fakt „Boeing employs 10.000 people in unit `null` located in `null`“ Informationen über Abteilung und Ort der beschäftigten Mitarbeiter. Die Completeness eines Fakts $f \in T$ wird in [16] wie folgt definiert:

Für ein Attribut $a \in Typ(t)$ gibt die Funktion $fullness(a)$ 0 zurück, wenn das Attribut unbelegt (`null`) ist, ansonsten 1. Dann definieren wir die Completeness eines Fakts als:

$$completeness(f,t) = \frac{\sum_{a \in Typ(t)} fullness(a)}{|Typ(t)|}$$

Interest (Benutzerinteresse) pro Fakt, Objekt oder Fakt-Typ beschreibt die relative Häufigkeit, mit der ein Benutzer mit einem bestimmten Fakt, Objekt oder Fakt-Typ interagiert. Z.B. wird die Ergebnisseite eines Objektes mit hohem Benutzerinteresse überdurchschnittlich häufig angezeigt.

Sei E eine repräsentative Menge von Interaktionen (Events) auf dem IR-System und p der Parameter, dessen Benutzerinteresse wir beschreiben wollen und dessen Vorkommen wir in E überprüfen können. Dann definieren wir den Interest eines Parameters:

$$interest(p,E) = \frac{count(p,E)}{|E|}$$

Fact Frequency (Frequenz) pro Fakt-Typ beschreibt die relative Häufigkeit von Fakten eines Typs in der gesamten Faktenbasis. Wir nutzen diese Verteilung als A-priori Fakten-Verteilung. Z.B. enthält der Reuters News Corpus [22] überdurchschnittlich viele Fakten vom Typ *personcareer*. Wir definieren die Frequency eines Fakt-Typs:

$$frequency(t,F) = \frac{|F_t|}{|F|}$$

Estimate (Schätzwert) pro Fakt-Typ beschreibt die relative Häufigkeit von Fakten eines Typs in allen verfügbaren Dokumenten. Da wir in einem IR-System davon ausgehen müssen, dass uns nur ein Teil aller Fakten und Dokumente im Web zur Verfügung steht, können wir diese A-posteriori Verteilung nur schätzen. Wir können diesen Wert beispielsweise vermuten, indem wir das Benutzer-Interesse an einem Fakt-Typ t als Approximation benutzen:

$$estimate(t) \approx interest(t,E)$$

Coverage (Abdeckung) pro Fakt-Typ beschreibt, wie hoch der Anteil von Fakten in einem Korpus aus der Menge aller möglichen Fakten ist. Anschaulich zeigt das Maß der Coverage, wie gut die A-priori-Verteilung der A-posteriori-Verteilung entspricht. Z.B. können wir die Coverage für den Fakt-Typ „companyproduct“ von Airbus ermitteln, indem wir die in der Faktenbasis enthaltene Liste von Flugzeugen mit dem offiziellen Produktkatalog von Airbus

als Schätzwert vergleichen. Wir definieren die Coverage eines Fakt-Typs als:

$$coverage(t, F) = \begin{cases} \frac{frequency(t,F)}{estimate(t)} & estimate(t) > 0 \\ \infty & estimate(t) = 0 \end{cases}$$

Deficiency (Mangel) pro Fakt-Typ beschreibt den Faktor zwischen der A-priori Verteilung und der vermuteten A-posteriori Verteilung der Faktenbasis. Anschaulich bedeutet deficiency$>$ 1, dass in der Faktenbasis verhältnismäßig weniger Fakten enthalten sind als wir sie in der Realität vermuten. Wir definieren die Deficiency eines Fakt-Typs als:

$$deficiency(t, F) = \begin{cases} \frac{estimate(t)}{frequency(t,F)} & frequency(t, F) > 0 \\ 0 & frequency(t, F) = 0 \end{cases}$$

Mit den vorgestellten Metriken ist es möglich, die beiden Prozesse Benutzerinteraktion und Retrieval quantitativ zu untersuchen und damit Datenqualität messbar zu machen. Wir werden sie in Kapitel 5 dazu nutzen, die für den Benutzer relevanten Suchergebnisse zu ermitteln, die Datenqualität der Faktenbasis zu bewerten und neue, ungesehene Fakten zu entdecken.

3.4. Darstellung der Interaktionen als Events

Um die Interaktionen und Operationen innerhalb eines IR-Systems messbar zu machen, formalisieren wir alle Aktionen des Nutzers und des Systems in möglichst minimaler Form als Events. Wir analysieren die Navigationsstruktur der GUI des IR-Systems und definieren für jede Interaktionsmöglichkeit ein Event mit spezifischen Parametern und einem sprechenden Identifier, z.B. `ShowStartPageEvent`. Minimal bedeutet, dass jedes Event einer Benutzerintention und einer festgelegten darauf folgenden Systemoperation entspricht. Einige komplexere Interaktionen müssen wir also als zusammenhängende Folge von mehreren Events definieren. Z.B. formalisieren wir die Interaktion „ein Benutzer erhält für seine Suchanfrage kein Ergebnis“ wie folgt: `SearchStringEvent` → `ShowNotFoundPageEvent`.

Wir analysieren unser prototypisches IR-System GoOLAP, dessen Web-Frontend in Abbildung 3.2 dargestellt ist. Dazu ermitteln wir alle Interaktionsmöglichkeiten auf diesem Frontend in möglichst minimaler Form und benennen diese als Events. Jedes Event besitzt eine unterschiedliche Anzahl an Parametern, die Informationen über die angezeigte Seite, die Objekte und Fakten im Fokus der Interaktion und die Benutzersession. Wir finden in GoOLAP sechs Klassen von Events und definieren als Basis für unsere Analyse 29 verschiedene Events. Wir stellen zuerst die Event-Klassen vor und zeigen dann einen wichtigen Teil der ermittelten Events.

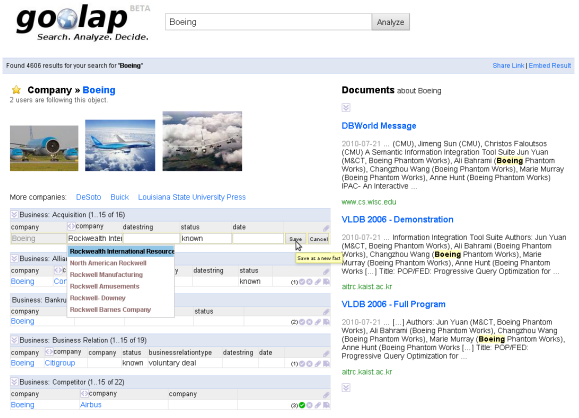


Abbildung 3.2.: Das GoOLAP Web Frontend

Page View Events werden immer dann aufgerufen, wenn das System eine neue Ergebnisseite zurückgibt (z.B. `ShowAnalyzePageEvent`). Viele Interaktionen auf dem Frontend sind über JavaScript/AJAX realisiert und geben deshalb keine kompletten Seiten als Ergebnis zurück. GoOLAP bietet die Views PAGE_START, PAGE_ANALYZE, PAGE_COMPARE und PAGE_NOTFOUND.

Session Interaction Events beschreiben Interaktionen des Benutzers, mit denen er in eine Session einsteigt, z.B. den Wechsel der Suchintention über die Eingabe einer neuen Suche in das Suchfeld (`SearchStringEvent`).

Result Navigation Events beschreiben die Navigation zwischen verschiedenen Ergebnisseiten innerhalb einer Session, z.B. die Clicks von einem Ergebnisobjekt zu einem anderen (`ClickObjectEvent`).

Compare Result Navigation Events beschreiben die Navigation auf der Ergebnisseite PAGE_COMPARE. Diese Seite erlaubt dem Benutzer, sein Suchergebnis in einer interaktiven Tabelle explorativ mit anderen Objekten zu vergleichen (z.B. `CompareAddObjectEvent`).

Fact Interaction Events beschreiben die Interaktion des Benutzers mit einzelnen Repräsentationen der angezeigten Fakten in einem Ergebnis, z.B. die manuelle Bewertung eines Fakts (`FactAgreeEvent`).

User Profile Events beschreiben die Verwendung der Benutzerprofile. Jeder Benutzer kann sich bei GoOLAP registrieren (z.B. `UserSignupEvent`) und mit seinem Profil Objekte abonnieren oder Tabellen abspeichern.

Tabelle 3.1 zeigt einen Ausschnitt aus allen Events, die wir auf dem GoOLAP-Frontend ermittelt haben. Sie beschreibt pro Zeile eine einzelne Interaktion und erklärt die Vorgänge innerhalb der beiden vorgestellten Prozesse beim Auslösen des Events: Die Intention, die der Benutzer mit seiner Interaktion verfolgt, die Rückgabe des Systems und die Aktivität, die der Retrieval-Prozess daraufhin im Hintergrund auslöst. Wir geben jedem Event einen Bezeichner (*Event ID*) und ordnen es einer der vorgestellten Aktivitätsklassen zu. Die vollständige Tabelle mit allen Events listen wir in Anhang A.1. Sie enthält außerdem komplette Parameterbeschreibungen zu jedem Event und wir verknüpfen dort unsere Definitionen für Benutzerziele (Abschnitt 3.2) und Qualitätsmaße (Abschnitt 3.3) mit allen Events, indem wir jedem Event zuordnen, ob die folgenden Eigenschaften zutreffen:

Fulfils a User Goal gibt an, ob der Benutzer mit Hilfe dieser Interaktion eines der Benutzerziele erreicht und sie ihm somit positiven Nutzen bietet. Ziel ist es, die Nutzung von Interaktionen, bei denen das nicht der Fall ist, vermeidbar zu machen.

Rates Interest/Coverage/Completeness/Correctness gibt an, ob die Interaktion daraufhin analysiert werden kann, das entsprechende Qualitätsmaß zu bestimmen und somit positiven Nutzen für das System bietet. Ziel ist es, die Nutzung dieser Interaktionen für den Benutzer möglichst erstrebenswert zu machen.

Creates Content gibt an, ob die Interaktion neue Inhalte generiert. Neben der Eigenschaft, dass dies zu neuen Daten in der Faktenbasis führt, ist diese Eigenschaft ein wichtiger Indikator dafür, dass eines der Qualitätsmaße im Kontext der Interaktion nicht ausreichend hoch ist.

Wir werden diese Events im Verlauf dieser Arbeit dazu nutzen, die Benutzerinteraktionen maschinell zu protokollieren und auszuwerten. Sämtliche Fragestellungen dieser Arbeit müssen durch die Analyse dieser Events beantwortbar sein, denn sie sind die einzigen Eingabedaten in unserem Auswertungsprozess. Deshalb ist es wichtig, dass wir die Navigationsstruktur des Systems vollständig durch Events abdecken und alle Facetten der Interaktion in den Events erfassen. Wir beschreiben im nächsten Kapitel, wie wir die automatische Protokollierung der vorgestellten Events implementieren können und wie wir die dabei entstehenden Logfiles auswerten können. Dabei beobachten wir, dass wir mit einer großen Menge an Events rechnen müssen, deren skalierbare maschinelle Verarbeitung eine Hürde für unser System darstellt.

Event ID	*Symbol*	*User interaction*	*Intention: The user...*	*System activity: The system...*	*Retrieval activity: The retrieval process..*	*Activity class*
PAGE VIEW EVENTS						
ShowStartPageEvent	**S**	Access the GoOLAP start page	wants to start a new search	proposes interesting objects to the user		navigate
ShowAnalyzePage-Event	**A**	Show AUGMENT result	requests an AUGMENT result for the object	displays the top rated facts and three arbitrary documents for the object	retrieves more facts for the object	learn
ShowComparePage-Event	**C**	EXPAND list of objects	wants to compare multiple objects of the same type and clicks on the EXPAND icon in the column header	expands the list of objects and displays an interactive table view for comparison	retrieves more facts for the objects	learn
ShowDocuments-Event	**D**	TRACE BACK the documents	wants to trace back the textual origin of a fact and clicks on the document symbol	displays a snippet from the source document and explains the origin of a fact	retrieves more evidences of this fact	lookup
ShowNotFoundPage-Event	**N**	Search missing object	wants to augment facts for an object that is not yet contained in the GoOLAP fact base	displays a message to inform the user about the empty result	retrieves facts for the object	lookup
SESSION INTERACTION EVENTS						
RefererEvent	**r**	Click a link that points to GoOLAP	discovers a GoOLAP result page in another search engine and clicks on the link	displays the AUGMENT result page for the object	retrieves more facts for the object	navigate
SearchObjectEvent	**s**	INTERPRET keyword search via autocomplete	wants to augment facts for an object displayed in the autocomplete and clicks on it	augments facts for the object and displays the AUGMENT page	retrieves more facts for the object	lookup
SearchStringEvent	**s**	Search with keyword	wants to augment facts for an object and types its name into the search field	interprets the input and displays an autocomplete dropdown with multiple interpretations	ranks the order of interpretations	lookup
RESULT / COMPARE RESULT NAVIGATION EVENTS						
ClickObjectEvent		Click on an object link in a fact table	clicks on an object that appears in a fact	displays the AUGMENT result page for the object	retrieves more facts for the object	lookup
CompareAddObject-Event		Add a row to the table	wants to compare another object and adds it to the table	displays all fact columns for the object	retrieves more facts for the object	investigate
FACT INTERACTION EVENTS						
FactAgreeEvent		Agree to a fact	thinks that a fact is correct	displays a green icon on the fact	retrieves more evidences of the fact	investigate
FactDisagreeEvent		Disagree to a fact	thinks that a fact is wrong	displays a red X on the fact	retrieves different facts for the object and fact type	investigate
USER PROFILE EVENTS						
UserSignupEvent		Sign up a GoOLAP user account	wants to use advanced features on the GoOLAP frontend and registers	creates a new user account and sends a welcome message		socialize
...						

Tabelle 3.1.: Ausschnitt der Liste aller Events im System GoOLAP

3.5. Zusammenfassung

Wir erkennen bei der explorativen Suche in einem IR-System zwei Prozesse: den *Suchprozess*, in dem der Benutzer seine Intentionen durch Interaktionen mit dem System ausdrückt, und den *Retrieval-Prozess*, welcher im Hintergrund die gewünschten Daten aus dem Web in eine Faktenbasis extrahiert. Abhängig von der Aktivitätsklasse *Lookup*, *Learn* und *Investigate* können wir die Anfragen des Benutzers bereits mit Abfragen an die Faktenbasis beantworten. Zur Bewertung der Qualität dieser Antworten definieren wir die Metriken *Correctness*, *Completeness*, *Interest*, *Frequency*, *Estimate*, *Coverage* und *Deficiency*. Zur Bestimmung einiger dieser Qualitätsmaße müssen wir die Interaktionen des Benutzers mit dem System beobachten. Wir definieren *Events* für alle im System möglichen Interaktionen und ordnen ihnen Aktivitätsklassen und ihren möglichen Einfluss auf die vorgestellten Qualitätsmaße zu. Die Beobachtung der Nutzung erfolgt, indem wir alle Events protokollieren und analysieren. Im folgenden Kapitel werden wir die Implementierung der Protokollierung in Logfiles erläutern, Techniken zur Analyse der Logfiles einführen und dem Problem der Skalierbarkeit dieses Auswertungsprozesses begegnen.

4. Analyse von Benutzerinteraktion

Wir haben konzeptuell vorgestellt, wie Benutzer ihre Intentionen durch Interaktionen innerhalb eines IR-Systems ausdrücken und die beiden Prozesse der Suchaktivität und Retrieval-Aktivität beschrieben. In diesem Kapitel stellen wir einen Analyseprozess vor und zeigen, wie wir die große Menge an Interaktionen erfassen und skalierbar auswerten können. Dazu protokollieren wir das Auftreten der in Abschnitt 3.4 vorgestellten Events in Logfiles und analysieren die Daten im Batch-Verfahren auf einem parallel verteilen System. Die Ergebnisse der Auswertung lassen Rückschlüsse auf das Interesse der Nutzer, ihr Navigationsverhalten innerhalb der Seite und die Qualität und Abdeckung der im System enthaltenen Fakten zu.

4.1. Was heißt Web-Scale?

Wir erwarten bei der Analyse von Benutzerinteraktion eine große Menge an zu verarbeitenden Daten. Alle vorgestellten Konzepte sollen von Beginn an hoch skalierbar sein, also z.B. auch auf einem großen Websystem einsetzbar sein, in dem mehrere zehntausend Benutzer mit einem rapide wachsenden Datenbestand von mehreren hundert Millionen Objekten und untereinander interagieren können. Der Begriff *Web-Scale* beschreibt dabei nicht nur den hohen Datendurchsatz dieses Systems, sondern vor allem den hohen Grad der Vernetzung zwischen den Daten, Benutzern und Interaktionen. Metcalfe's Law beschreibt als Auswirkung dieses Netzwerkeffekts auf den Wert einer solchen Datenbasis ein quadratisches Wachstum, verschiedene andere Interpretationen gehen zumindest von einem Wachstum stark überhalb eines linearen Faktors aus [11].

Zu diesen Daten gehören in einem IR-System z.B. alle Quell-Dokumente, extrahierte Fakten, ihre Typisierungen und ihr Ranking, Logfiles von Benutzerinteraktionen, Metadaten über die Benutzer des Systems und ihre Vernetzung untereinander. Typische Berechnungen auf diesen Daten sind invertierte Indexes, Repräsentationen der Daten in Graphen, Aggregation, Zusammenfassung und Gruppierung über Anzahl, Verteilung und Dichte der Daten. Um die stark wachsenden Datenmengen verarbeiten zu können, müssen wir Algorithmen verwenden, die sich leicht parallelisieren und auf vielen Maschinen verteilt ausführen lassen. Wir folgen dabei dem weit verbreiteten Programmiermodell MapReduce [9], das auch weitere Anforderungen von Web-Scale Anwendungen adressiert: Fehlertoleranz, effiziente Verteilung der Daten und Verteilung der Last.

4.2. Protokollierung der Benutzerinteraktion in Logfiles

Da wir umfangreiche und detaillierte Auswertungen über die Nutzung unseres Systems vornehmen möchten, protokollieren wir alle im System auftretenden Ereignisse in möglichst granularer Form. Wir bilden die in Abschnitt 3.4 beschriebene Formalisierung aller Nutzerinteraktionen als *Events* 1:1 in ein Logfile ab. Da wir in Web-Scale arbeiten, erwarten wir über einen längeren Zeitraum große Datenmengen im Terabyte-Bereich. Aus Gründen der Skalierbarkeit ist es deshalb nicht möglich, die Events in einer relationalen Datenbank zu speichern und abzufragen. Wir schreiben stattdessen sequentielle Textdateien und nutzen das einfache semi-strukturierte Datenaustauschformat *JavaScript Object Notation* (JSON) [13]. Jedes auftretende Event wird in der Reihenfolge des Auftretens als neue Zeile in die

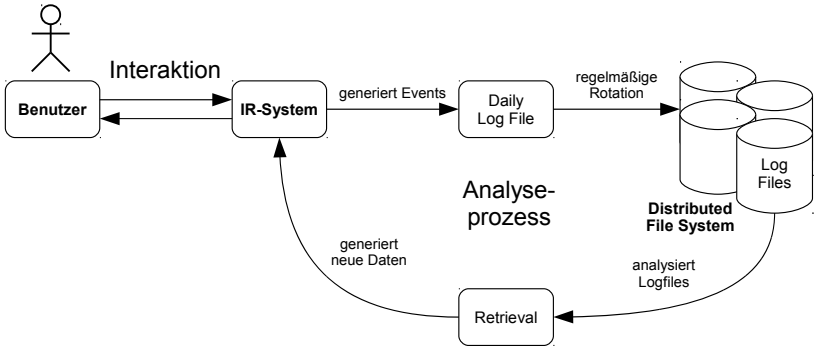


Abbildung 4.1.: Prozess der Protokollierung und Analyse von Benutzerinteraktion

Datei geschrieben. Dazu nutzen wir ein geeignetes Framework (z.B. log4j[1]), um direkt aus der Anwendung Logfiles zu generieren, diese zu rotieren und archivieren. Wir beobachten die Interaktionen des Benutzers anhand der folgenden Merkmale: (1) Jeder Funktionsaufruf im System (z.B. `showStartPage()`) wird einem Event zugeordnet. (2) Interaktionen, die nur auf dem Frontend ablaufen (z.B. der Klick auf einen externen Link) erkennen wir über JavaScript/AJAX. (3) Die Zuordnung von Events zu Benutzer-Sessions entnehmen wir dem unterliegenden Framework der Webapplikation (z.B. Java/JSP mit Stripes Framework[2]). (4) Die Abhängigkeiten zwischen Events (z.B. aufeinanderfolgende Interaktionen zwischen zwei Objekten im Frontend) berechnen wir bereits zur Laufzeit und speichern sie zu den Events als Parameter. Die systematische Auswertung erfolgt dann parallelisiert auf einem verteilen Dateisystem (siehe Abschnitt 4.5) in regelmäßigen Abständen im Batch-Verfahren. Abbildung 4.1 zeigt den Prozess der Protokollierung und Auswertung.

Format des Logfiles. Alle Events liegen in einem Logfile zeilenweise im JSON-Format vor. Wir betrachten das gesamte File als Array von Events, die jeweils als JSON-Objekt abgelegt sind. Listing 4.1 zeigt das JSON-Objekt für ein Event vom Typ `ShowAnalyzePageEvent` in unserem Prototypen GoOLAP. Die Event-Typen beschreiben die Interaktionsmöglichkeiten des IR-Systems und unterscheiden sich in ihren Parametern. Eine Übersicht über alle im System GoOLAP möglichen Events bietet die Tabelle in Anhang A.1.

```
{
  "type"            : "EVENT_SHOW_ANALYZE_PAGE" ,
  "timestamp"       : "Fri Jun 10 23:57:33 CEST 2011" ,
  "object"          : {
                        "ID"   : "c2fb25742db7ffbde4a5db1b3929fb72c7541728",
                        "label": "Boeing" ,
                        "type" : 9
                      } ,
  "pageType"        : "PAGE_ANALYZE" ,
  "sourcePage"      : "PAGE_ANALYZE" ,
  "sourceObjectId"  : "2607307523889b227eeb60b01d5a504f82fb8e49" ,
  "user"            : {
                        "id"   : null
                      } ,
  "sessionId"       : "140537F12B64C455F94F31C7AB391C01" ,
  "userAgent"       : "Mozilla/5.0" ,
  "referer"         : "http://www.goolap.info/analyze/product/Airbus+A330/",
  "botRequest"      : false
}
```

Listing 4.1: JSON-Repräsentation eines Events im Logfile

[1] http://logging.apache.org/log4j/
[2] http://www.stripesframework.org

Im Folgenden erklären wir die wichtigsten Parameter eines Events:

`type`	Bezeichner (ID) des Event-Typs
`timestamp`	Zeitpunkt, zu dem das Event aufgetreten ist
`pageType`	ID des Seitentyps, der angezeigt werden soll (PAGE_START, PAGE_ANALYZE, PAGE_COMPARE, PAGE_NOTFOUND)
`object`	Objekt (mit ID, label und type), auf das sich das Event bezieht
`sourcePage`	ID des Seitentyps, auf der das Event ausgelöst wurde
`sourceObjectID`	ID des Objektes, welches beim Auslösen des Events im Fokus stand
`sessionID`	eindeutige ID der Benutzer-Session
`botRequest`	gibt an, ob die Anfrage als automatisierter Aufruf durch einen Bot, Spider oder Crawler erkannt wurde

Die Parameter sind so gewählt, dass wir zur Auswertung direkten Zugriff auf die wichtigsten Kriterien zur Selektion, Gruppierung und Kombination haben. Z.B. können wir mit dem Ausdruck `filter botRequest==false` einfach menschliche Sessions selektieren, Sessions mit `group by sessionID` zusammenfassen und mittels der Parameter `sourcePage` und `sourceObjectID` Klickpfade und Abhängigkeiten nachvollziehen.

4.3. Ermittlung und Analyse von Clickstream-Graphen

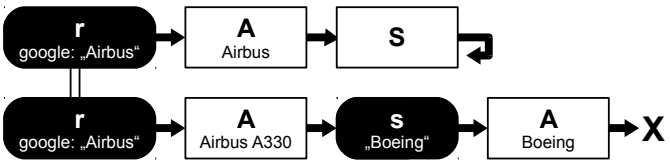


Abbildung 4.2.: Web Behaviour Graph einer Folge von Interaktionen

Wenn wir alle Events im Logfile bei der Auswertung über das Feld `sessionID` gruppieren, können wir die Benutzerinteraktionen auf Session-Ebene analysieren. Im Folgenden betrachten wir die Bewegungen der Benutzer in dem System und verfolgen den sequentiellen *Clickstream* von Events einer einzelnen Benutzersession. Wir stellen die Bewegungen des Benutzers grafisch dar und verwenden zu diesem Zweck die in [8] beschriebene Notation als *Web Behaviour Graph*. Abbildung 4.2 illustriert die Interaktion eines Benutzers mit dem Suchsystem GoOLAP im zeitlichen Ablauf von links nach rechts und von oben nach unten. Die Knoten repräsentieren die Events im Logfile, die gerichteten Kanten stellen den zeitlichen Ablauf dar. Ausgangspunkt ist in diesem Beispiel ein `RefererEvent` (`r`, Interaktionen sind als schwarzes abgerundetes Rechteck dargestellt) für eine Google-Suche nach „Airbus“, auf der der Benutzer zwei Ergebnisse angeklickt hat, die auf goolap.info zeigen. Die angezeigten Ergebnisse sind Page Views (als weißes Rechteck dargestellt) vom Typ `PAGE_ANALYZE` (`A`) für die Objekte „Airbus“ und „Airbus A330“. Der Benutzer navigiert im ersten Ergebnis zur Startseite (`S`), im zweiten Ergebnis über eine Sucheingabe (`s`) bis zum Ende der Session (x). Vertikale Linien und der „zurück“-Pfeil zeigen einen *Trackback* (`b`), also das Zurückkehren auf eine vorherige Seite (z.B. über den „back“-Button oder die Browser Tabs) an.

Aufbau des Graphs aus Events im Logfile. Für eine gegebene Session können wir den Graph mit einem iterativen Algorithmus aufbauen, der als Pseudocode in Listing 4.2 aufgeführt ist.

Dazu selektieren wir aus dem Logfile alle Events einer Session (FILTER) und betrachten sie in der zeitlichen Reihenfolge ihres Auftretens (SORT). Wir iterieren über alle Events (FOREACH) und fügen sie dem Graph in folgender Weise hinzu: (1) Wir ermitteln den Knoten der Ursprungsseite (`parent`) mit den Parametern `sourcePage` und `sourceObjectId` des Events. (2) Wenn der letzte Page View ungleich `parent` ist, fügen wir einen Trackback zur Ursprungsseite ein. (3) Danach fügen wir den Event als Knoten vom passenden Typ ans Ende des Graphs ein. Am Ende der Session schließen wir den Graph ab.

```
INPUT: eventLogs, sessionID
FILTER $.sessionID == sessionID
SORT BY $.timestamp
CREATE graph
FOREACH event
    parent = graph.getLastNode($.sourcePage,$.sourceObjectId)
    IF (graph.getLastPageViewNode() != parent) graph.addTrackback(parent)
    IF ($.type IN (PageViewEvents)) graph.addPageViewNode($)
    ELSE IF ($.type IN (InteractionEvents)) graph.addInteractionNode($)
NEXT
graph.addSessionEndNode()
OUTPUT: graph
```

Listing 4.2: Algorithmus zum Aufbau eines Web Behaviour Graphs aus einem Logfile

Repräsentation einer Folge von Interaktionen als Zeichenkette. Den resultierenden Graph können wir auch in vereinfachter Form als Zeichenkette (*Click Trail*) darstellen, welche von links nach rechts den zeitlichen Ablauf der Interaktionen als jeweils ein Zeichen darstellt. Dabei schreiben wir Page Views in Großbuchstaben und Interaktionen in Kleinbuchstaben und schließen bei kompletten Sessions die Zeichenkette mit einem `x` ab. Trackbacks können wir mit einem `b` darstellen, verlieren hierbei die Information zur Position der Ursprungsseite. Das Beispiel aus Abbildung 4.2 wird demnach durch die Zeichenkette `"rASbrAsAx"` dargestellt. Wir benutzen diese Click Trails im Folgenden, um größere Mengen von Benutzersessions miteinander zu vergleichen.

4.4. Ermittlung typischer Navigationspfade

Der in Abschnitt 4.3 vorgestellte Algorithmus visualisiert den Pfad einer einzelnen Benutzersession. Im Folgenden erweitern wir diese Analyse und versuchen, unter allen Benutzersessions gleichartige Sessions zu finden. Als Kriterium für Gleichartigkeit können wir verschiedene Parameter heranziehen. Es ist z.B. möglich, die Bewegungen auf der Navigationsstruktur einer Website (Navigationsbasiert) oder die Bewegungen auf ihrer Ontologie (Query-Basiert) zu betrachten. Im ersten Fall betrachten wir gleichartige Web Behaviour Graphen auf Basis der Knotentypen (z.B. Page View `C` oder Sucheingabe `s`), im zweiten Fall auf Basis der Inhalte der Queries (z.B. Objekt *Airbus* oder Typ *Company*).

Typische Pfade auf der Seitennavigation. Wir ermitteln gleichartige Pfade in allen Benutzersessions auf der Navigationsstruktur einer Webseite. Das heißt, wir vergleichen die Abfolge von besuchten Seitentypen aller Benutzer untereinander. Dazu stellen wir die Pfade als Click Trails dar. Da wir den Navigationsbasierten Ansatz wählen und die Inhalte der Queries außer Acht lassen, genügt diese Darstellung für unsere Analyse. Die Backtracking-Schritte könnten in diese Darstellung übernommen werden, wir lassen sie aber aus, um die Granularität der Pfadbeschreibungen nicht zu fein werden zu lassen. Ist der auszuwertende Datensatz sehr groß, könnten wir Trackbacks und weitere Interaktionen einbeziehen, um eine detailliertere Auswertung erreichen.

Mit Hilfe des in Listing 4.3 vorgestellten Algorithmus generieren wir den Click Trail für jede Session im Logfile und erhalten durch Gruppierung eine Verteilung über die absolute Häufigkeit gleicher Pfade. Wir betrachten keine automatisch generierten Seitenbesuche von Bots (FILTER), selektieren nur die Events, die wir für Navigationspfade betrachten wollen (FILTER) und stellen die zeitliche Abfolge sicher (SORT). Dann gruppieren wir die Events mit gleicher sessionID (GROUP), zählen die Länge des Pfades (`trailLength`) jeder Session (COUNT) und erzeugen den Click Trail, indem wir alle Event-Typen (`type`) jeder Session zu einer Zeichenkette konkatenieren. Das Ergebnis gruppieren wir nach gleichen Click Trails (GROUP) und zählen die Häufigkeit jedes Navigationspfades (COUNT). Wir sortieren das Ergebnis nach der Häufigkeit der Pfade (SORT), um die meist genutzten Pfade im Ergebnis vorne zu finden.

```
INPUT: eventLogs
FILTER $.botRequest==false
FILTER $.type IN (PageViewEvents, SessionInteractionEvents)
SORT BY $.timestamp
GROUP BY $.sessionID
    COUNT $ AS trailLength
    CONCAT $.type AS clickTrail
GROUP BY $.clickTrail
    COUNT $ AS trailCount
SORT BY $.trailCount DESC
OUTPUT: (clickTrail, trailLength, trailCount)
```

Listing 4.3: Generierung der Verteilung von Click Trails aus Logfiles

Durchsuchen des Click Trails nach häufigen Mustern. Aus dem Ergebnis der Verteilung gleicher Navigationspfade können wir möglicherweise häufige Muster in den Click Trails oder in Teilen der Click Trails erkennen, wie z.B. die direkte Transition von einer Ergebnisseite zu einer anderen `"AA"`, oder das Beenden der Session nach einem Suchergebnis `"sAx"`. Wir selektieren einige dieser Muster und zählen das ihr Auftreten in allen Session Click Trails, um ihre absoluten Häufigkeiten zu bestimmen. Der in Listing 4.4 beschriebene Algorithmus verwendet die oben berechneten Click Trails und nutzt Regular Expressions, um das Muster in den Zeichenketten zu zählen. Für unsere beiden Beispiel nutzen wir dafür die Ausdrücke `"?=AA"` (mit einer *zero-width positive look-ahead assertion*, um auch Mehrfachtransitionen wie z.B. `AAAA` korrekt zu zählen) `"sA$"` (mit dem Zeichen `$`, um das Ende der Zeichenkette zu erkennen). Wir übergeben die Liste von Mustern (`patternList`) und bilden das Kartesische Produkt von Click Trails und Patterns (CROSS). Für jedes dieser Paare berechnen mit `regexAll` jedes Auftreten des Patterns im Click Trail (`matches`), zählen ihre Anzahl und multiplizieren mit der Häufigkeit des gesamten Click Trails im Logfile (`countMatches`). Wir selektieren alle Paare mit Übereinstimmungen (FILTER) und gruppieren die Muster (GROUP) mit der Summe ihrer Matches (SUM). Als Ausgabe erhalten wir die Muster mit der Häufigkeit ihres Auftretens im Logfile.

```
INPUT: sessionTrails, patternList
CROSS sessionTrails, patternList
FOREACH trailAndPattern
    $.matches = regexAll($.pattern,$.trail)
    $.countMatches = count($.matches) * $.trailCount
NEXT
FILTER $.countMatches > 0
GROUP BY $.pattern
    SUM $.countMatches AS patternCount
OUTPUT: (pattern, patternCount)
```

Listing 4.4: Zählen aller Muster in Click Trails

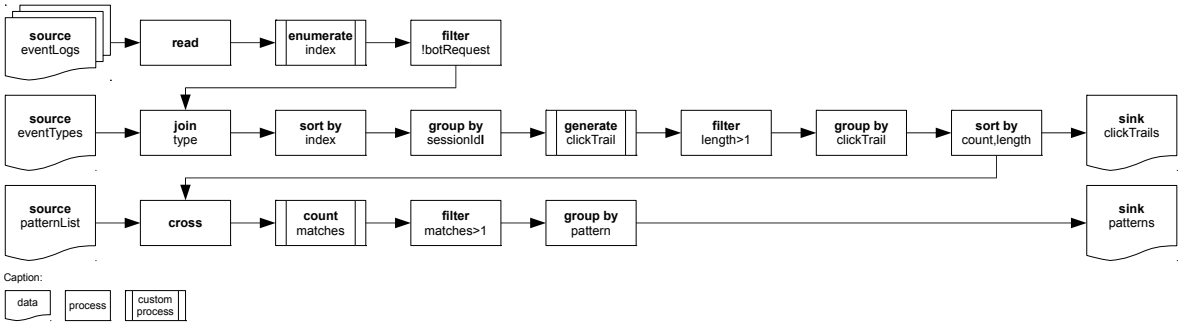


Abbildung 4.3.: Datenflussdiagramm der Analyse von Interaktionsmustern

4.5. Massiv parallele Ausführung der Logfile-Analyse

Da bei der Analyse von Benutzerinteraktion in Web-Scale große Datenmengen anfallen, benötigen wir einen Weg, um die Berechnungen auf den Daten effizient und mit niedrigen Wartezeiten durchzuführen. Wir haben es nicht mit einem Echtzeitsystem zu tun, in dem schnelle Antwortzeiten notwendig sind. Trotzdem ist es für einen kontinuierlichen Retrieval-Prozess nötig, die Analysen auf Logfiles mehrerer Monate oder Jahre regelmäßig innerhalb weniger Stunden durchführen zu können.

Wir speichern und berechnen die Daten deshalb parallel verteilt in einem *Apache Hadoop* Cluster [1]. Hadoop basiert auf dem MapReduce-Programmiermodell [9] und ermöglicht damit aufwändige Berechnungen auf großen Datenmengen. Dabei werden die zu verarbeitenden Daten auf mehrere Nodes verteilt und parallel berechnet, die Ergebnisse am Ende in Form eines verteilten Merge Sorts effizient wieder zusammengefügt. Für die Speicherung der Daten kommt das *Hadoop Distributed File System* (HDFS) zum Einsatz.

Viele iterative Algorithmen lassen sich nur schwer parallelisieren, deshalb verwenden wir für unsere Auswertung nur Operatoren, die effiziente Verteilung von Datenflüssen ermöglichen, z.B. FILTER oder GROUP. Die so aufgebauten Algorithmen drücken wir in der Hochsprache *JAQL* (JavaScript Object Notation Query Language) [12] aus. Die Sprache bietet eine funktionale Programmierschnittstelle zu Hadoop, indem sie die gängigen Operatoren zur Datenverarbeitung als MapReduce-Jobs abstrahiert und selbstständig für eine verteilte Ausführung sorgt. JAQL verwendet das JSON-Format für alle Datenstrukturen und bietet direkte Unterstützung für Lese- und Schreibzugriff in HDFS. Somit können wir über JAQL direkt auf die Objekte in den Logfiles zugreifen.

Abbildung 4.3 fasst die in den Abschnitten 4.3 und 4.4 vorgestellten Algorithmen zur Analyse der Navigationspfade in einem Datenflussdiagramm zusammen. Wir implementieren diesen Datenfluss zur parallelisierten Ausführung in JAQL und listen das gesamte Skript in Anhang A.2.

4.6. Zusammenfassung

Wir arbeiten bei einem explorativen Suchsystem in *Web-Scale* und erwarten deshalb bei der Analyse von Benutzerinteraktion eine große Menge an zu verarbeitenden Daten. Um diesem Problem zu begegnen, protokollieren wir alle Benutzerinteraktionen als Events in Logfiles, die wir in einem verteilten Dateisystem abspeichern und hochgradig parallelisiert auswerten können. Zum Einsatz kommen dabei einfache sequentielle Textdateien im JSON-Format, die wir in einem verteilten Hadoop-Cluster mit Hilfe der Datenflusssprache JAQL auswerten. Diese Lösung skaliert sehr gut, da Hadoop auf dem *MapReduce*-Programmiermodell aufbaut

und die vorgestellten Algorithmen auf die damit mögliche Parallelisierung ausgerichtet sind. Der Prozess der Auswertung hat zum Ziel, den strukturellen Verlauf der Benutzersessions auf *Click Trails* abzubilden und typische *Navigationspfade* innerhalb dieser Click Trails zu identifizieren. Wir durchsuchen dazu automatisiert die Click Trails aller Benutzersessions nach häufigen Mustern und zählen ihr Vorkommen. Aus den Ergebniswerten können wir Schlüsse über die Verwendung des Systems ziehen und die Qualitätsmaße Interest und Estimate bestimmen. Im folgenden Kapitel wenden wir dieses Prinzip an, um die Benutzerinteraktionen auf dem Prototypen GoOLAP zu analysieren und unsere Konzepte zu evaluieren.

5. Evaluierung am Prototypen GoOLAP

In diesem Kapitel nutzen wir die Erkenntnisse über das Konzept von Interaktion auf Webseiten und die Techniken für deren Analyse, um sie auf einem realen System in Web-Scale anzuwenden. Dazu stellen wir das prototypische IR-System GoOLAP vor und werten einen Datensatz mit Interaktionen aus, den wir über einen Zeitraum von 2 Monaten in diesem System erhoben haben. Wir nehmen eine quantitative Auswertung der Nutzung vor und untersuchen in einer qualitativen Analyse häufige Navigationspfade. Abschließend bewerten wir unsere Erkenntnisse und weisen auf Probleme hin, die in verschiedenen Stadien der Analyse auftauchen können.

5.1. Was ist GoOLAP?

Das System GoOLAP [2] ist ein interaktives Information Retrieval System, das Fakten aus ihrer textuellen Repräsentation in Web-Dokumenten aufspürt, extrahiert und für strukturierte Anfragen zur Verfügung stellt. Dazu bietet es ein Web-Frontend[1] mit einer Reihe von Operatoren, die es dem Benutzer ermöglichen, diese Fakten anzureichern, zu analysieren und miteinander zu vergleichen. Die Faktenbasis enthält derzeit über 6 Millionen Objekte und Fakten aus einer Ontologie von über 70 Objekt- und Fakt-Typen wie z.B. *Person*, *Company*, *personattributes* und *companycompetitor*.

Diese Arbeit dient dem Ziel, die Faktenbasis ständig zu erweitern und zu verdichten, indem wir die Nutzung der Operatoren beobachten, auswerten und dabei gezielt fehlende Fakten aufspüren. Die folgenden Operatoren stehen dem Benutzer dabei zur Verfügung (eine detaillierte Beschreibung bietet [2], die Repräsentation aller Operatoren als Events findet sich im Anhang A.1):

INTERPRET Das System interpretiert eine Benutzereingabe (z.B. „Boeing“) und liefert mögliche Interpretationen (z.B. *Company Boeing*, *Product Boeing 747*) in einem Auto-Complete zur Auswahl.

AUGMENT GoOLAP sammelt Fakten zu einem bestimmten Objekt und stellt sie in aggregierter Form auf der ANALYZE View dar. Dabei werden die Fakten nach ihrer Relevanz geordnet angezeigt.

EXPAND Mit Hilfe dieser Operation lassen sich Objekte in einer interaktiven COMPARE View vergleichen und analysieren.

TRACE BACK Der Benutzer kann zu jedem Zeitpunkt den Ursprung eines Faktes im Original-Dokument anzeigen lassen.

SUBSCRIBE Ein registrierter Benutzer kann ein Objekt abonnieren und somit GoOLAP beauftragen, neue Fakten für dieses Objekt zu finden.

CONTRIBUTE Ein registrierter Benutzer kann mit diesem neuen Operator zur Datenqualität beitragen, indem er Fakten bewertet oder manuell editiert.

Das System GoOLAP entsteht seit 2008 als regelmäßig durchgeführte Lehrveranstaltung mit Bachelor- und Masterstudenten an der Technischen Universität Berlin.

[1] zu erreichen unter der Adresse http://www.goolap.info

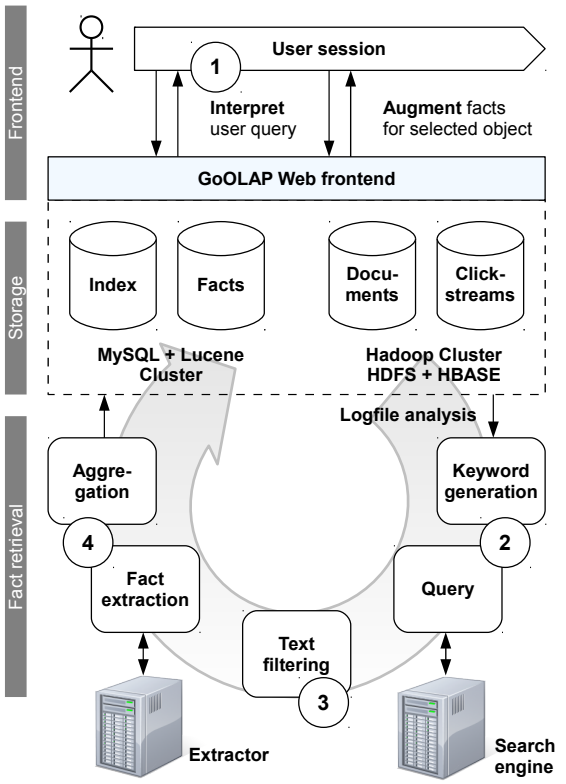


Abbildung 5.1.: Fact Retrieval über Web-Suchmaschinen mit GoOLAP

Abbildung 5.1 zeigt den Information Retrieval Prozess innerhalb des GoOLAP-Systems: (1) Der Benutzer interagiert mit dem Web Frontend (hier mit den Operatoren INTERPRET und AUGMENT). Das System protokolliert die Interaktionen, ermittelt die Suchintention des Nutzers und liefert ein Ergebnis zurück, indem es die lokale Faktenbasis anfragt. Gleichzeitig bewertet das System die Completeness und *Rarity* [16] der Ergebnisse, um potentiell fehlende Fakten zu identifizierten. Falls nötig, wird eine Fact Retrieval Aktivität gestartet. (2) GoOLAP leitet die textuelle Repräsentation des entsprechenden Objektes und weitere semantische Information (z.B. den Typ) an FactCrawl [6] weiter. Dieses Framework imitiert das Verhalten des Benutzers einer Suchmaschine, um ein inverses Fact Retrieval Problem zu lösen. Es generiert automatisch Keyword Queries und fragt damit eine Web-Suchmaschine an. Die Ergebnisse werden mit einem neuartigen Fact Score neu bewertet. (3) Die zurückgelieferten Dokumente werden analysiert und die potentielle Dichte an Fakten ermittelt [5]. Da die Faktenextraktion sehr viel Zeit und Ressourcen in Anspruch nimmt, ist es wichtig, nur relevante Dokumente an den Extraktor zu übermitteln. (4) Die relevanten Dokumente werden an einen externen Extraktor geschickt, welcher strukturierte Daten zurückliefert. Diese Daten werden aggregiert und in der lokalen Faktenbasis abgelegt.

5.2. Datenerhebung im Live-Betrieb

Zur Evaluierung unserer Konzepte beobachten wir die Interaktionen aller Nutzer auf der öffentlich zugänglichen Webseite goolap.info im Zeitraum 01.05.2011–30.06.2011. Dabei entstand ein Datenset aus 59 Tages-Logfiles (die zwei fehlenden Tage sind auf entwicklungsbedingte Downtime zurückzuführen) mit einer Gesamtgröße von 253MB. In dem Datenset befinden sich 650.412 Events, von denen aber eine Vielzahl von Bots, Robots und Crawlern generiert wurden. In einem ersten Schritt filtern wir deshalb das Datenset, um möglichst nur relevante Events zu betrachten. Wir erkennen die meisten Bots bereits beim Schreiben der Logfiles anhand des HTTP-Headers `UserAgent` und weiteren Indikatoren, wie z.B. die Ablehnung von Cook-

Date Range	**01.05.2011–30.06.2011**
Daily Log Files	59
Events	650.412
Events (without Bots)	112.304
Events per Day	1.903
Sessions	72.489
Sessions (length > 1)	14.756
Sessions per Day	1.228

Tabelle 5.1.: Übersicht über das Datenset von Events auf GoOLAP

ies und damit einer konsistenten SessionID über mehrere Anfragen hinweg. Wir filtern diese *BotEvents* heraus und erhalten 112.304 Events (1.903 pro Tag) bzw. 72.489 Sessions, auf denen wir im Folgenden die Abfragen ausführen. Tabelle 5.1 zeigt eine Übersicht über das Datenset.

5.3. Quantitative Analyse

Wir geben einen Überblick über die quantitative Nutzung des GoOLAP-Systems und stellen die Ergebnisse verschiedener Anfragen an das Datenset vor. Als Ausgangsmenge dient dazu jeweils die Menge an Events im vorliegenden Datenset. Unsere Analyse teilt sich in zwei Ebenen: (1) Wir treffen Aussagen über die Nutzung des Systems in Hinblick auf die Struktur der Nutzung. Dafür berechnen wir den Interest an den Operatoren des Systems und ermitteln Eigenschaften der gesamten Sessions aller Benutzer. (2) Danach analysieren wir die Nutzung des Systems im Hinblick auf die Struktur ihrer Inhalte. Wir ermitteln die am häufigsten angefragten Objekte, die häufigsten fehlgeschlagenen Sucheingaben (*Zero Hit Queries*) und berechnen Verteilungen über Interest und Coverage von Fakt-Typen.

Nutzung der Systemoperatoren. Wir untersuchen die Häufigkeit der Nutzung der zu Beginn des Kapitels vorgestellten Operatoren. Dazu gruppieren wir im vorliegenden Datenset aller Events nach ihrem Typ und summieren für jeden Operator die Häufigkeiten der Aufrufe der entsprechenden Events.

Wir erkennen in den Ergebnissen in Tabelle 5.2 einen hohen Interest für die Aufrufe der Ergebnisseiten (gesamt 80,35% aller Events). Außerdem gibt es eine hohe Anzahl an Aufrufen der Seite durch Referers (gesamt 12,81%), auf die wir später zurückkommen. Es fällt auf, dass relativ wenige Interaktionen auf den Seiten selbst auftreten, darunter vor allem die Bewertungsfunktionen (gesamt 3,42%) und die Suchfunktionen (gesamt 2,48%). Bei einigen Events ist das Verhältnis nicht nachvollziehbar. So gibt es weitaus mehr `ShowNotFoundPageEvents` als `SearchStringEvents` und überhaupt kein Auftreten von `ClickURLEvent`.

Wir interpretieren diese Beobachtungen wie folgt: (1) Die meisten Seitenaufrufe erfolgen direkt, z.B. über Links in Suchmaschinen. (2) Die GoOLAP-Suchfunktion wird wenig verwendet, das Autocomplete (`SearchObjectEvent`) nicht einmal in einem Drittel aller Fälle. (3) Die explorativen Interaktionsmöglichkeiten auf den Ergebnisseiten werden wenig genutzt. (4) Dies könnte auch dadurch zu erklären sein, dass viele Events

System Operator	*Events*	*Interest*
SESSION START	**27.696**	**24,66%**
ShowStartPageEvent	13.312	11,85%
RefererEvent	4.081	3,63%
RefererGoogleEvent	10.303	9,17%
INTERPRET	**9.661**	**8,60%**
SearchObjectEvent	760	0,68%
SearchStringEvent	2.028	1,81%
ShowNotFoundPageEvent	6.873	6,12%
AUGMENT	**58.529**	**52,12%**
Click*ObjectEvent	715	0,64%
ShowAnalyzePageEvent	57.814	51,48%
EXPAND	**5.609**	**4,99%**
ClickExpandEvent	99	0,09%
Compare*Event	5.510	4,91%
TRACE BACK	**6.724**	**5,99%**
ShowDocumentEvent	6.724	5,99%
ClickURLEvent	0	0,00%
CONTRIBUTE	**3.880**	**3,45%**
FactAgreeEvent	1.998	1,78%
FactDisagreeEvent	1.757	1,56%
FactClearAgreeEvent	81	0,07%
FactEdit*Event	44	0,04%
Others	**205**	**0,18%**
Total (subtotals in bold)	**112.304**	100,00%

Tabelle 5.2.: Nutzung der GoOLAP Systemoperatoren, aufgeschlüsselt nach Events

gar nicht von Benutzern, sondern automatisch erzeugt werden, z.B. von Spam-Bots. Damit lassen sich auch die unstimmigen Verhältnisse bei den `ShowNotFoundPageEvents` erklären. Wir nehmen an, dass die `Click*Events` davon nicht betroffen sind, da sie ausschließlich bei aktiviertem JavaScript erzeugt werden. Wir diskutieren diese Erkenntnisse in Abschnitt 5.5.

Eigenschaften der Benutzersessions. Wir gruppieren alle Benutzersessions nach ihrer Länge in Events und in Sekunden und zählen ihre Häufigkeiten im Logfile. Tabelle 5.3 zeigt das Ergebnis für Sessions der Länge $n = 1 \ldots 10$. Die Linke Seite beschreibt die Anzahl der Sessions im Logfile mit n Events, die rechte Seite die Anzahl der Sessions mit der Länge n (vom ersten zum letzten Event) in Sekunden.

Length	*Sessions w/*			*Sessions w/*		
***n*=**	***n** Events*	*%*	*cumm.*	***n** seconds*	*%*	*cumm.*
1	57.633	79,47%	79,47%	60.150	82,94%	82,94%
2	7.064	9,74%	89,21%	1.974	2,72%	85,66%
3	4.177	5,76%	94,96%	1.323	1,82%	87,48%
4	1.398	1,93%	96,89%	1.340	1,85%	89,33%
5	619	0,85%	97,75%	1.107	1,53%	90,86%
6	276	0,38%	98,13%	708	0,98%	91,83%
7	192	0,26%	98,39%	489	0,67%	92,51%
8	130	0,18%	98,57%	310	0,43%	92,93%
9	147	0,20%	98,77%	277	0,38%	93,32%
10	185	0,26%	99,03%	264	0,36%	93,68%
...	705	0,97%		4584	6,32%	
Total	72.526	100%	100%	72.526	100%	100%

Tabelle 5.3.: Summe aller Sessions auf GoOLAP gruppiert nach Länge

Wir beobachten, dass die meisten Sessions nur aus einem einzigen Event bestehen. Da für beide Verteilungen der Mittelwert und Median jeweils 1 sind, berechnen wir im Ergebnis die kumulierten Häufigkeiten und beobachten die 90%-Schranke (gestrichelte Linie). Dabei können wir beobachten, dass fast 90% aller Sessions aus maximal 2 Events und maximal 4 Sekunden Länge bestehen.

Häufig angefragte Objekte. Wir interessieren uns für die Inhalte der Interaktionen und ermitteln, welche Objekte in der Faktenbasis häufig angefragt werden. Dazu gruppieren wir die Events im Logfile nach dem Parameter `ObjektId` und summieren alle Aufrufe. Tabelle 5.4 zeigt die 25 häufigsten auf dem GoOLAP Frontend angefragten Objekte aus einer Auswertung von 70.711 Objekt-Aufrufen. Die Spalte *Spans* gibt an, an wie vielen Textstellen das Objekt gefunden wurde (Autor-Interest). Wir können keine Korrelation zwischen diesem Wert und dem Interest auszumachen, so hat z.B. das Top-Objekt „Rider“[2] eine Ergebnisseite mit niedriger Coverage und Completeness.

Rank	*Object Label*	*Type*	*Events*	*Interest*	*Spans*
1	Rider	Position	453	0,641%	402
2	Chicago Bears	Organization	415	0,587%	849
3	Staff Writer	Position	405	0,573%	1.036
4	FeedBurner	Company	353	0,499%	1.186
5	Fiona Loudon	Person	338	0,478%	4
6	MetLife	Company	310	0,438%	187
7	Chicago Sun-Times	Company	269	0,380%	748
8	President	Position	212	0,300%	220.933
9	Angela Merkel	Person	171	0,242%	8.103
10	IBM	Company	164	0,232%	9.764
11	Jay Leno	Person	143	0,202%	1.186
12	Microsoft	Company	143	0,202%	10.591
13	Maggie	Person	137	0,194%	297
14	designer	Position	136	0,192%	4.240
15	Chairman of the Board of Directors	Position	133	0,188%	209
16	Board Member	Position	133	0,188%	3.048
17	EBay	Company	132	0,187%	5.403
18	Google	Company	124	0,175%	29.488
19	Facebook	Company	122	0,173%	22.502
20	Irene	Person	118	0,167%	265
21	Plain Dealer	Company	114	0,161%	507
22	deputy head	Position	114	0,161%	207
23	Wal Mart Stores Inc.	Company	113	0,160%	331
24	City Manager	Position	113	0,160%	508
25	MP3	Technology	106	0,150%	15.011
...			...	...	
Total			**70.711**	100%	6.293.536

Tabelle 5.4.: Die 25 populärsten Objekte auf GoOLAP

Wir Interpretieren das Ergebnis wie folgt: Die Ergebnisseiten mit hohem Interest sind weit oben in einer Web-Suchmaschine platziert oder es handelt sich bei einigen vorderen Platzierungen um Sessions einzelner Power-User oder um automatisierte Anfragen. Dieser Verdacht erhärtet sich bei der Betrachtung der Zero Hit Queries.

[2] http://www.goolap.info/analyze/position/Rider/

Häufige Zero Hit Queries. Wir haben anfangs beobachtet, dass 6,12% aller Interaktionen auf Objekte verweisen, die nicht in der Faktenbasis vorhanden sind. Die Auswertung der `PageNotFoundEvents` zeigt uns, welche Objekte mit Null-Coverage einen hohen Interest haben und damit wichtige Kandidaten für den Retrieval-Prozess sind. Dazu gruppieren wir alle Events dieses Typs nach dem Parameter `searchString` und zählen die Häufigkeit.

Rank	*Count*	*Search Query*
1	338	`chairman of the board of directors`
2	89	`Thomas Lewis`
3	88	`George Lewis`
4	59	`George W. Bush`
5	52	`William+Henry`
...		
Total	**2.187**	

Tabelle 5.5.: Die 5 häufigsten Zero Hit Queries auf GoOLAP

Wir bemerken, dass die meisten Anfragen mit einer leeren Zeichenkette als Eingabe gestellt werden und filtern diese nachträglich. Die 5 häufigsten nicht-leeren Zero Hit Queries zeigt Tabelle 5.5. Wir interpretieren dieses Ergebnis wie folgt: (1) Wir erkennen, wie anfällig die Analyse der Interaktion für automatisch generierte Anfragen ist. Z.B. ist die Top-Query ohne Ergebnis `"chairman of the board of directors"` ein Tippfehler mit zwei Spaces. Sie ist der Anfrage an das Objekt „Chairman of the Board of Directors“[3] ähnlich (Rank 15 der populären Objekte) und hat einen auffällig großen Abstand zur zweiten Platzierung. Ähnlich sind die beiden folgenden Anfragen an ähnliche Objekte mit dem gleichen Tippfehler. (2) Die erfolglose populäre Anfrage an `"George W. Bush"` hingegen ist tatsächlich ein wichtiger Hinweis auf eine Null-Coverage mit hohem Interest, denn ein Objekt mit exakt diesem Namen ist in der Faktenbasis nicht enthalten. Der Grund dafür liegt allerdings nicht im Retrieval-Prozess selbst: es scheint, als könne der verwendete Extraktor diesen Namen nicht auflösen, wahrscheinlich aufgrund des Punktes.

Verteilung von Fakt-Typen. Wir interessieren uns für den Interest einzelner Fakten abhängig von ihrem Typ. Die Distribution von faktueller Information in Web-Dokumenten über Fakt-Typen ist nicht gleichverteilt. In [6] wird diese Verteilung im Reuters News Corpus (NIST) [22] evaluiert. Dieser Standardisierte Evaluationskorpus enthält 731.752 News-Dokument aus den Jahren 1996–1997 und wurde mithilfe von OpenCalais [20] auf enthaltene Fakten untersucht. Das Ergebnis zeigt eine Long Tail Verteilung über die Fakt-Typen. Das bedeutet, sehr wenige Fakt-Typen kommen sehr häufig in Dokumenten vor, sehr viele Fakt-Typen aber nur selten. Wir berechnen die gleiche Verteilung für die Frequency von Fakt-Typen in der GoOLAP-Faktenbasis. Das Ergebnis zeigen die schwarzen Balken im Diagramm in Abbildung 5.2. Die vertikale Achse listet die häufigsten 25 Fakt-Typen, die horizontalen schwarzen Balken geben die Fact Frequency relativ zum gesamten Korpus an. Diese Verteilung korrespondiert mit der Verteilung im NIST-Korpus, wir nutzen sie hier als A-priori-Verteilung.

Zu unseren Zielen gehört die Erhöhung der Coverage in der Faktenbasis. Da wir nicht wissen, welche Fakttypen häufig im Web vorkommen, untersuchen den Interest der Benutzer und approximieren damit den Estimate, um Hinweise auf fehlende Fakten eines Typs zu finden. Wir messen Interest, indem wir alle Events betrachten, in denen mit einem Fakt bestimmten Typs interagiert wird (z.B. `ShowDocumentsEvent` oder `FactAgreeEvent`) und gruppieren nach Fakt-Typ.

Die weißen Balken im Diagramm in Abbildung 5.2 zeigen das Ergebnis dieser Anfrage. Es fällt auf, dass die Verteilung des Interests grundsätzlich mit der Frequency korrespondiert. Wir berechnen aus dem Verhältnis von Interest zu Frequency die Deficiency, im Diagramm rechts dargestellt. Werte > 1 bedeuten, dass der Interest höher ist als die A-priori-Verteilung.

Wir interpretieren unsere Beobachtungen wie folgt: Das ausgeglichene Verhältnis der bei-

[3] http://www.goolap.info/analyze/position/Chairman+of+the+Board+of+Directors/

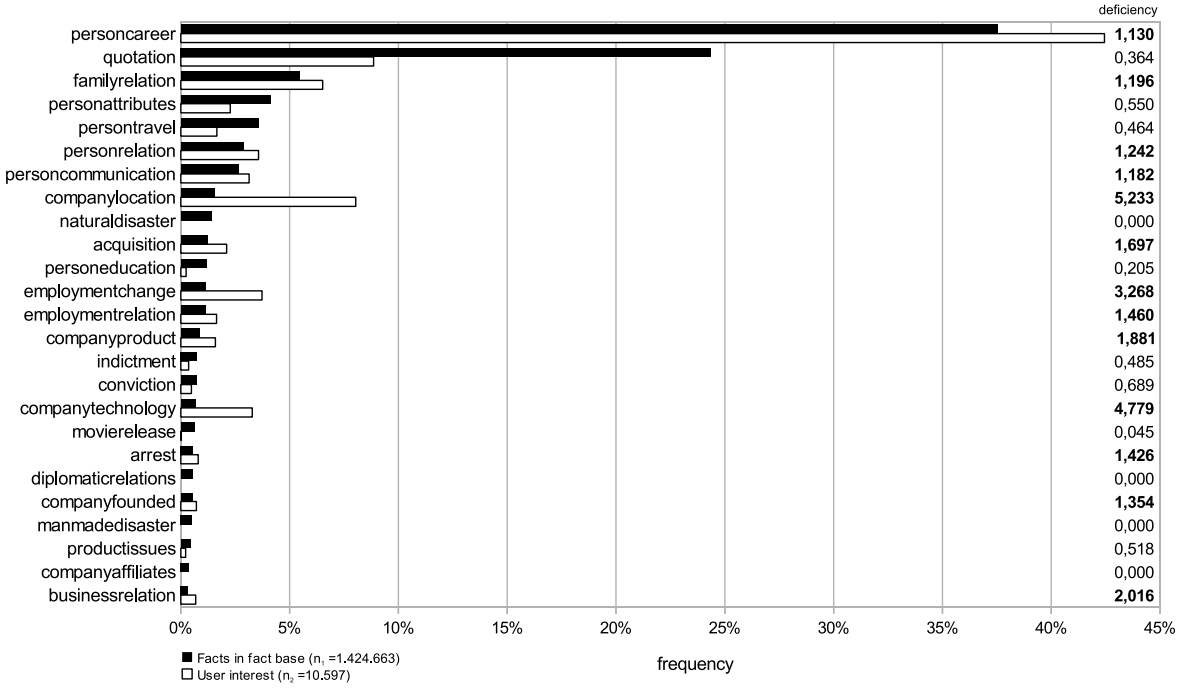


Abbildung 5.2.: Vergleich der Verteilungen von vorhandenen Fakt Typen in der GoOLAP Faktenbasis und ihrer Nutzung im Frontend

den Verteilungen folgt unter anderem daraus, dass Fakt-Typen, die mit großer Häufigkeit in der Faktenbasis enthalten sind, auch im Frontend öfter angezeigt und somit angeklickt werden. Interessant sind allerdings die Ausreißer mit signifikant höherer Deficiency. Z.B. wird `companylocation` in 8,0% aller Interaktionen mit Fakten verwendet, hat unter allen Fakten aber nur einen Anteil von 1,5%. Diese Ergebnisse können wir verwenden, um mit den in [5] vorgestellten Techniken gezielte Retrieval-Aktivitäten nach spezifischen Fakttypen zu starten.

5.4. Qualitative Analyse

Wir interessieren uns für typische Verhaltensmuster bei der Interaktion auf dem GoOLAP Web-Frontend und berechnen dafür die in Abschnitt 4.4 als Click Trails eingeführten Zeichenketten als Repräsentation für die Navigation eines Benutzers. Zur Illustration der Ergebnisse benutzen wir die Web Behaviour Graphen aus Abschnitt 4.3. Unsere Vorgehensweise in der qualitativen Analyse hat explorativen Charakter und erstreckt sich über zwei Ebenen: (1) Um typische Pfade auf der Navigationsstruktur zu finden, ermitteln wir alle vollständigen Session Click Trails und betrachten einige interessante häufig genutzte Einzelfälle näher. In diesen Einzelfällen erkennen wir Muster, nach deren Vorkommen wir in allen Teilen aller Click Trails suchen. (2) Auf Basis dieser Erkenntnisse erstellen wir weitere Anfragen, die die Verwendung dieser Muster auf Inhaltlicher Ebene untersuchen. Wir konzentrieren uns dabei auf Muster, die für den Benutzer und das System positiven Nutzen haben.

Übersicht häufiger Session Click Trails. Mit dem in Abschnitt 4.4 vorgestellten Algorithmus gruppieren wir alle vollständigen Click Trails für Sessions der Länge > 1 und zählen ihre Häufigkeiten. Tabelle 5.6 zeigt die Verteilung für die 15 häufigsten Click Trails und ihre Repräsentation als Zeichenketten. Wir beobachten viele kurze Click Trails der Länge 2–3, aber auch drei längere typische Sessions. Dies ist damit zu begründen, dass im Datenset fast

Rank	*Click Trail*	*Length*	*Activity class*	*Count*	*Interest*
1	**rAx**	2	lookup	8.347	56,046%
2	**rASx**	3	lookup	1.839	12,348%
3	SSx	2	navigate	445	2,988%
4	**rDx**	2	lookup	251	1,685%
5	**AAx**	2	learn	238	1,598%
6	**rSx**	2	navigate	222	1,491%
7	rASSx	4	lookup	106	0,712%
8	**ACCCx**	4	investigate	96	0,645%
9	ASx	2	navigate	85	0,571%
10	**ASAAAx**	5	learn	83	0,557%
11	**sNx**	2	lookup	80	0,537%
12	rNx	2	lookup	78	0,524%
13	**SAx**	2	lookup	74	0,497%
14	**rDSx**	3	lookup	72	0,483%
15	**rAAx**	3	learn	66	0,443%
...				...	...
Total				**14.893**	100%

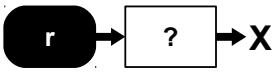
Tabelle 5.6.: Die 15 häufigsten Session Click Trails auf GoOLAP

Pattern	*Description*	*Activity class*	*Matches*
rS	Link to GoOLAP home page	navigate	574
rA	Link to GoOLAP result page	lookup	10.933
rD	Link to GoOLAP document	learn	393
rC	Link to GoOLAP compare view	lookup	24
r?x	Bounce after first page view	lookup	8.785
sNx	Bounce after no result	lookup	375
AA	Transition between two objects	learn	2.623
CC	Interaction in table view	investigate	1.535
AC	Expand operator	investigate	651
AsA	Explore with new search	learn	308
SsA	Start a new search	lookup	297
sNsA	Search again after no result	learn	285

Tabelle 5.7.: Die Häufigkeiten typischer Click Trails in allen Sessions

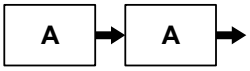
90% aller Sessions die Länge 1 oder 2 haben, und dass die Wahrscheinlichkeit des Auftretens gleicher kurzer Click Trails höher ist als für längere Click Trails.

Diesen Faktor relativieren wir, indem wir gezielt nach typischen Interaktionsmustern suchen. Wir wählen aus der Tabelle häufig verwendete Muster (z.B. **AA**), die unserer Ansicht nach typische Suchintentionen abbilden und positiven Nutzen für den Benutzer und das System haben (fett markiert). Diese Muster suchen wir dann in allen Interaktionen im Datenset. Die Muster sind nicht notwendigerweise disjunkt und können in einer Session mehrfach auftreten. Zum Einsatz kommt dabei der zweite Algorithmus aus Abschnitt 4.4. Die von uns gewählten Muster und die ermittelten Häufigkeiten zeigt Tabelle 5.7. Im Folgenden gehen wir auf einige dieser Muster näher ein.



Referer–PageView–Bounce (r?x, Klasse: Lookup). Wir beobachten in den Ergebnissen, dass die häufigsten Anfragen an GoOLAP von Referers, also von Web-Suchmaschinen oder Links stammen, welche meist direkt auf Ergebnisseiten vom Typ ANALYZE zeigen. Das kann bedeuten, dass die GoOLAP-Ergebnisseiten für viele Suchintentionen passende Ergebnisse liefern. So sind z.B. derzeit über 80.000 GoOLAP-Seiten bei Google indiziert und mögliche Kandidaten zur Erfüllung eines Ziels der Klasse *Informational* für eine Web-Suche.

Wir beobachten dabei allerdings auch, dass die Bounce Rate auf der Seite sehr hoch ist. Fast 59% aller Sessions stammen von Referers und enden sofort nach dem ersten Page View. Das bedeutet, dass von der Mehrzahl der Benutzer die explorativen Interaktionsmöglichkeiten auf der Seite nicht wahrgenommen werden. Durch die Analyse von Interaktionsmustern können wir aber gezielt die interessanten Fälle herausgreifen und sie auf inhaltlicher Ebene analysieren.



Analyze–Analyze (AA, Klasse: Learn). Die häufigste Interaktion zwischen zwei Ergebnisseiten ist die Transition zwischen zwei Seiten vom Typ ANALYZE. Dabei klickt der Benutzer auf der Ergebnisseite eines Objekts auf einen Objektnamen eines anderen Objekts. Bei der Analyse

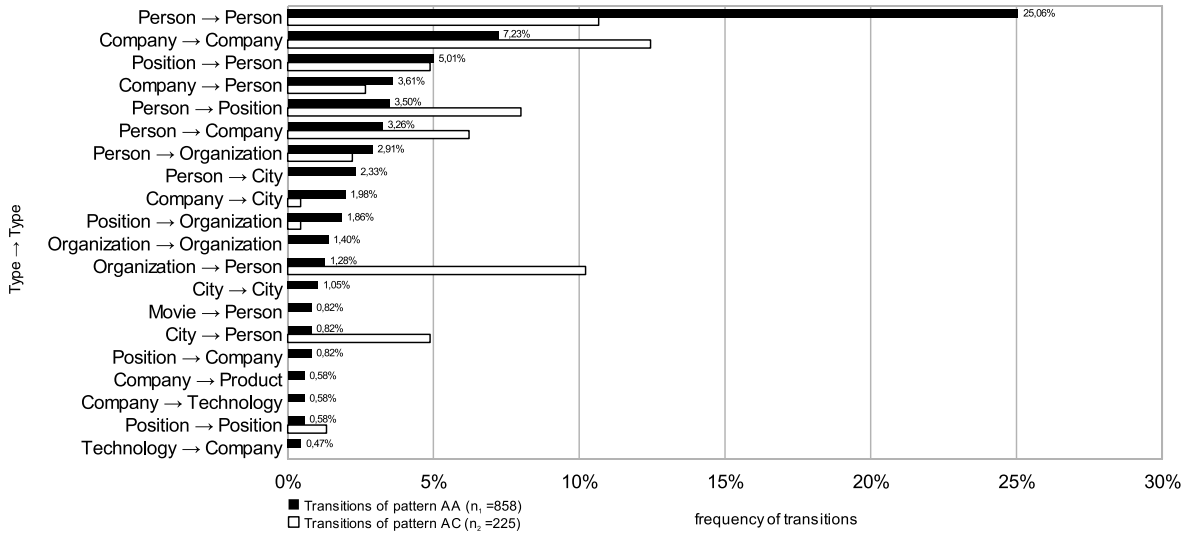
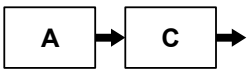


Abbildung 5.3.: Die häufigsten Transitionen vom Typ `AA` und `AC` gruppiert nach Objekt-Typen

dieser Interaktionen fällt auf, dass von den 2.623 Transitionen die meisten auf das selbe Objekt verweisen. Nur 858 Transitionen finden zwischen unterschiedlichen Objekten statt. Die Erklärung hierfür ist der Seitennavigation geschuldet: ein Klick auf „View Document“ lädt die Seite neu und generiert somit einen neuen `ShowAnalyzePageEvent` mit dem selben Objekt. Wir betrachten deshalb im Ergebnis nur die nicht-trivialen Transitionen und gruppieren über die Objekt-Typen von Quellseite und Zielseite. Damit erhalten wir die in Abbildung 5.3 als schwarze Balken gezeigte Verteilung von Typ-zu-Typ Transitionen.

Wir beobachten im Ergebnis, dass die häufigsten Transitionen zwischen verschiedenen Objekten vom Typ Person stattfinden und außerdem eine starke Häufung der Interaktionen zwischen den Typen Person, Company und Position auftritt. Wir können aus der Verteilung auch schließen, für welche Beziehungen zwischen verschiedenen Objekt-Typen sich die Benutzer am meisten interessieren. Z.B. wird Position→ Person relativ häufig angeklickt. Das bedeutet, der Benutzer interessiert sich für Personen, die eine bestimmte Position innehaben, z.B. „President“[4]. Anstatt aber den Operator EXPAND zu benutzen, mit dem sich auf einen Blick viele Präsidenten vergleichen lassen, klickt er auf einzelne Personen, die in den Fakten verknüpft sind. Wir können diese Erkenntnisse dazu nutzen, den EXPAND-Operator mit den richtigen Parametern im Frontend gezielt anzubieten.



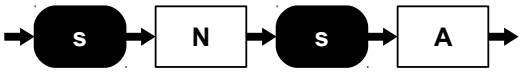
Analyze-Compare (`AC`, Klasse: Investigate). Die Transition `AC` stellt die Nutzung des Operators EXPAND dar. Dabei klickt der Benutzer auf einer Ergebnisseite vom Typ ANALYZE auf das entsprechende Icon, um mehrere Objekte eines Typs zu vergleichen. Dabei stehen alle Objekte über Fakten in Beziehung mit dem Ursprungsobjekt. Z.B. ist es (wie oben angedeutet) möglich, vom Ursprungsobjekt „President“ (Position) aus alle Personen zu vergleichen, die über den Fakt-Typ personcareer mit der Position „President“ verknüpft sind. Wir berechnen Analog zur Vorgehensweise bei Muster `AA` die Verteilung von Typ-zu-Typ Transitionen (im Beispiel Position→ Person). Das Ergebnis dieser Anfrage illustrieren die weißen Balken im Diagramm in Abbildung 5.3.

Wir beobachten: (1) Die Nutzung des EXPAND-Operators erfolgt vergleichsweise selten.

[4]http://www.goolap.info/analyze/position/President/

Da wir die relativen Häufigkeiten im Diagramm normiert auf die jeweilige Stichprobe vergleichend darstellen, können wir nur Tendenzen feststellen, nicht absolute Nutzungshäufigkeiten vergleichen. (2) Anstelle von einer typischen Power-Law Verteilung entdecken wir im vorderen Segment mehrere Transitionen, die ähnlich häufig genutzt werden.

Wir schließen daraus, dass der EXPAND-Operator von den Benutzern sehr zielgerichtet in der Aktivität *Investigate* benutzt wird, und weniger stark abhängig von der Coverage einzelner Fakt-Typen ist wie bei einfachem explorativem Browsing. Wichtig ist deshalb gerade für die häufig verwendeten Objekt-Typen, die Einstiegspunkte im Frontend an prominenter Stelle anzubieten.



Search–NotFound–Search–Analyze (`sNsA`, Klasse: Learn). Dieses Interaktionsmuster beschreibt das folgende Szenario: Der Benutzer gibt einen Suchbegriff in das Suchfeld ein, bekommt kein Ergebnis (*Zero Hit Query*) und sucht erneut, diesmal mit Erfolg. Das Muster `sNsA` können wir im vorliegenden Datenset 285 mal finden. Interessant ist für uns die Beziehung zwischen dem ursprünglichen, nicht gefundenen Suchbegriff und dem darauf folgenden erfolgreichen Suchbegriff, wir untersuchen deshalb den Abstand zwischen den beiden eingegebenen Zeichenketten. Dazu verwenden wir das Distanzmaß von Levenshtein [14], welches die minimale Anzahl an Editieroperationen berechnet, die wir für die Umwandlung einer Zeichenkette in eine andere Zeichenkette benötigen. Wir normalisieren den Wert, indem wir ihn durch die Summe der Länge beider Zeichenketten teilen. Das hat die Auswirkung, dass Einfügungen und Auslassungen in Suchbegriffen nicht zu sehr ins Gewicht fallen, und dass der Wert für die vollständige Veränderung eines Suchbegriffes weniger von dessen Länge abhängt. Z.B. ist der Wert für die Änderung der Zeichenkette `"ABC"` in `"XYZ"` gleich 1. Wir gruppieren für alle errechneten Distanzen nahe beieinander liegende Werte und betrachten die Verteilung der Werte. Abbildung 5.4 zeigt die Häufigkeiten der Distanzwerte für alle Suchbegriffe im Muster `sNsA` im Datenset. Die horizontale Achse beschreibt die Distanz zwischen den beiden Suchbegriffen, die vertikale Achse die Häufigkeit des Auftretens.

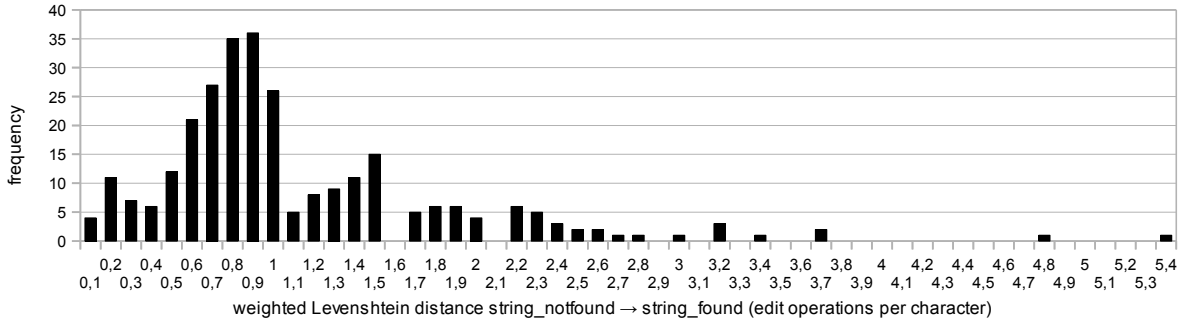


Abbildung 5.4.: Levenshtein-Distanz zwischen den Zeichenketten einer Suche ohne Ergebnis und der darauffolgenden erfolgreichen Suche

Wir beobachten, dass es einige ähnliche Paare von Suchbegriffen geben muss und betrachten die eingegebenen Suchworte mit hoher Ähnlichkeit qualitativ. Dabei fällt auf, dass im Bereich der Distanzen 0,1–0,6 fast vollständig keine Veränderung der Suchintention auftritt, sich aber verschiedene Typen von Änderungen zwischen den Suchbegriffen ergeben: Tippfehler, Formen der Identifikation, Sprache, Spezialisierung / Generalisierung, Auswahl ähnlicher Objekte und Umformulierungen. Tabelle 5.8 zeigt zu jeder dieser Klassen Beispiel-Queries, wie wir sie im Datenset finden konnten.

Est. Distance	*Type*	*Example NotFound*	*Found*	*Distance*
0.0–0.2	**Typing error**	`Paul Bloxam`	`Paul Bloxham`	0,08
0.1–0.6	**Identification**	`Mary B Cahill`	`Mary Cahill`	0,17
0.2–0.5	**Language**	`Schokolade`	`Chocolate`	0,30
0.3–0.8	**Specialization / Generalization**	`XING & Facebook`	`Facebook`	0,78
0.4–0.8	**Similar Object**	`Jürgen Klopp`	`Jürgen Klinsmann`	0,41
0.5–1.0	**Formulation**	`Who is miss universe?`	`Miss Universe Inc.`	0,60
0.5–5.0	**Intention Change**	`Katze`	`Laboratory of Photosynthesis`	0,83

Tabelle 5.8.: Beispiele für Klassifizierung gleichbleibender Suchintentionen bei erneuter Suche nach einer Zero Hit Query

5.5. Auftretende Probleme

Unser langfristiges Ziel ist es, die Auswertung der Benutzerinteraktion in Teilen zu automatisieren und die hier vorgestellten Erkenntnisse zu nutzen, um Qualität und Abdeckung der Faktenbasis zu erhöhen. Bei der Evaluation zeigen sich verschiedene Probleme, die für die zukünftige automatisierte Auswertung gelöst werden müssen, um verfälschte Ergebnisse zu vermeiden. Ein anderer Faktor, der unsere vorgestellten Ergebnisse mit Sicherheit stark beeinflusst ist die kleine Stichprobe dieser Evaluation, bedingt durch wenige Benutzer auf der Seite, eine hohe Bounce Rate und einen relativ kurzen Evaluationszeitraum. Wir gehen kurz auf die zwei wichtigsten Probleme ein:

Beibehaltung der Reihenfolge der Events. Da wir bei der Auswertung der Benutzerinteraktion mit zeitlich positionierten sequenziellen Daten arbeiten, ist die Einhaltung der Reihenfolge durch den gesamten Analyseprozess hindurch sehr wichtig. Wir haben diesen Faktor an einigen Stellen unterschätzt und müssen die Umsetzung für den zukünftigen Einsatz in Web-Scale sorgfältig anpassen:

(1) Da das Web-Interface mit vielen aufeinanderfolgenden HTTP-Redirects arbeitet, kann es passieren, dass einige Events (z.B. `RefererEvent`) doppelt protokolliert werden. Da der Zeitstempel nur sekundengenau arbeitet, können wir diese Events aber nachträglich zusammenfassen. Wir stellen für die zukünftige Protokollierung sicher, dass alle Events eindeutig ausgelöst werden.

(2) Durch die sekundengenauen Zeitstempel ergibt sich aber ein anderes Problem: die Reihenfolge der Eingabedaten bleibt bei vielen Operationen in JAQL nicht erhalten. Zusammengesetzte Events, welche im System zwar in der richtigen Reihenfolge ausgelöst werden, durch ihre quasi-gleichzeitige Ausführung aber den gleichen Zeitstempel erhalten (z.B. `RefererEvent` → `ShowAnalyzePageEvent`) könnten so vertauscht werden. Wir benutzen deshalb beim Einlesen der Logfiles die Operation *enumerate*, die jedes Event mit einem aufsteigenden Index versieht und an den entsprechenden Stellen im Datenfluss nach dem Index sortiert. In Zukunft werden wir auch die Zeitstempel auf eine feinere Skalierung umstellen.

(3) Es hat sich als aufwändig herausgestellt, die Abhängigkeiten sequentieller Daten untereinander mit verteilten Algorithmen zu behandeln. Um die volle Skalierbarkeit zu gewährleisten ist es in Zukunft nötig, weitere Daten vorauszuberechnen und in die Logfiles aufzunehmen. Z.B. protokollieren wir den Ursprung (Seitentyp, ObjektID) jedes Seitenaufrufes, benötigen aber viele Operationen um weitere Informationen über das Ursprungsevent und -objekt abrufen zu können. Da das GoOLAP-Logging auf Hadoop basiert und über ausreichende Hardwarekapazitäten verfügt können wir den Trade-off von redundanter Datenspeicherung zugunsten höherer Ausführungsgeschwindigkeit in Kauf nehmen.

Störende Eingabedaten. Durch die Vielzahl von verschiedenen Browsern, JavaScript- und Cookie-Einstellungen und automatisierte Anfragen von Bots, Crawlern, Spidern und Spam-Generatoren sind die Ergebnisse verfälscht. Z.B. erreichen GoOLAP 10–60 Anfragen pro Minute vom Google-Crawler *GoogleBot*. Diese Anfragen können wir problemlos filtern, aber in anderen Fällen funktioniert die Erkennung nicht. Auf Anfragen mit fehlerhaften Parametern (z.B. indizierte Links auf Ergebnisseiten alter GoOLAP-Versionen) ist das System zwar ausgelegt, trotzdem tauchen immer wieder Artefakte auf, die sich damit erklären lassen.

Bei der Erkennung der Interaktionen nutzen wir eine zweigeteilte Lösung: einige Events erkennen wir durch Funktionsaufrufe im System, andere lassen sich nur Client-seitig über JavaScript/AJAX auslösen. Das hat den Vorteil, dass die Protokollierung bei den meisten Events unabhängig von Benutzerseitigen Einstellungen funktioniert, was wir als wichtig erachten. Der Nachteil ist dabei, dass wir bei der Analyse auch den Fall ohne JavaScript-generierte Events beachten müssen. Eine Verbesserungsmöglichkeit ist dafür, das Vorhandensein von JavaScript im Browser für jedes Event mitzuspeichern.

Zusammenfassend wollen wir für die weitere Entwicklung drei Stufen des Prozesses verbessern: (1) Das System (GUI und Logging-Prozess) benötigt stärkere Validierung gegen fehlerhafte Eingabedaten, (2) die Erkennung von automatisierten Anfragen zur Laufzeit muss effizienter werden und (3) die Events müssen bei der Analyse intensiver gefiltert werden, um nicht nachvollziehbare Daten zu vermeiden.

5.6. Zusammenfassung

Wir wenden die Konzepte von Benutzerinteraktion bei explorativer Suche und die Techniken zur Analyse von Benutzerinteraktion auf dem System GoOLAP in Web-Scale an. Wir werten eine Erhebung von Interaktionen über einen Zeitraum von 2 Monaten aus und richten unser Augenmerk dabei auf die Verbesserung der Qualität und Abdeckung der GoOLAP-Faktenbasis. Wir betrachten zuerst die strukturelle Nutzung des Systems. Eine Beobachtung dabei ist, dass ein Großteil der protokollierten Interaktionen nicht durch menschliche Benutzung des Systems, sondern durch automatisierte Anfragen aus dem Web entsteht. Wir filtern diese Interaktionen aus. Wir beobachten in der quantitativen Analyse eine hohe Zahl von sehr kurzen Sessions der Länge 1 bis 2, wovon viele über Links von anderen Webseiten und Suchmaschinen entstehen und direkt auf eine Ergebnisseite zeigen. Es zeigt sich bei der Analyse der Benutzersessions, dass nur wenige Benutzer die explorativen Interaktionsmöglichkeiten des Systems ausschöpfen. Die qualitative Analyse zeigt aber, dass komplexere Operatoren wie EXPLAIN in gleichmäßiger Verteilung über die Faktenbasis genutzt werden und somit repräsentative Daten generieren können. Genauso können wir aussagekräftige Daten über das Benutzerinteresse an Fakten und den Mangel von Fakten bestimmter Typen ermitteln, indem wir gezielt Sessions mit typischen Navigationspfaden auswählen. Diese analysieren wir auf ihre Inhalte und beobachten, dass wir zukünftig Daten über das Interesse an Objekten und die Korrektheit von Fakten nutzen können, um diese mit einem Ranking zu versehen. Wir erkennen gezielt fehlende *Coverage* an häufigen *Zero Hit Queries* und der *Deficiency*, dem Verhältnis von *Fact Interest* zu *Fact Frequency*. Eine Analyse der eingegebenen Suchwörter zeigt auf, dass wir mit unserer Implementierung in der Lage sind, komplexere Zusammenhänge zwischen Interaktionen zu erkennen und diese z.B. dazu nutzen können, Sucheingaben besser zu interpretieren. Wir fassen im letzten Kapitel unsere Erkenntnisse zusammen und stellen weitere Ansätze für zukünftige Arbeiten vor.

6. Zusammenfassung und Ausblick

Im letzten Kapitel fassen wir unsere Erkenntnisse zusammen und geben einen Ausblick auf zukünftige Entwicklungen.

6.1. Zusammenfassung

Die komplexe Suchstrategie des Benutzers einer Web-Suchmaschine zeigt, wie kompliziert die Aufgabe ist, die Suchintentionen des Benutzers zu erkennen und seine Ziele zu erfüllen. Wir zeigen, wie wir das explorative Suchsystem GoOLAP mit interaktiven Information Retrieval-Methoden kombinieren können. Dazu analysieren wir die Navigationsstruktur des Systems, beobachten den Prozess der Benutzerinteraktion mit dem System und werten die Protokolldaten strukturiert aus, um Hinweise auf mangelnde Qualität und Abdeckung in der Faktenbasis zu erhalten. Dabei zeigt sich, dass es typische Verhaltensmuster gibt, die ein Benutzer auf seiner Navigation durch die Seitenstruktur aufgreift. Durch die Beobachtung dieser Muster erkennen wir Interesse von Benutzern an Objekten, Fakten und ihren Abhängigkeiten und nutzen das Interesse, um relevante Strukturen zu identifizieren. Da wir über geeignete Werkzeuge zum gezielten Retrieval von Fakten in Dokumenten im Web verfügen, können wir das Benutzerinteresse auch dahingehend nutzen, fehlende oder unvollständige Fakten aufzuspüren und über einem parallel verteilten Prozess in die Faktenbasis zu integrieren.

6.2. Ausblick

Um die Erkenntnisse der vorliegenden Arbeit in einem Produktivsystem einzusetzen, sind noch einige Schritte zur Integration nötig. Einige notwendige Weiterentwicklungen haben sich erst bei der Evaluierung der Konzepte gezeigt, andere bewegen sich außerhalb des Rahmens dieser Arbeit.

Einbindung der Interaktionsanalyse in den Retrieval-Prozess. Die Ergebnisse des Analyse-Prozesses sind nun in die GoOLAP Crawling-Pipeline zu integrieren. Die in dieser Arbeit manuell ausgeführten Anfragen sind 1:1 auf eine automatisierte Pipeline übertragbar, in der sie regelmäßig ausgeführt werden und Veränderungen in den Qualitätsmaßen registrieren. Wir werden für GoOLAP einen neuen Retrieval-Prozess implementieren und dabei die Analyse der Benutzerinteraktion zum gezielten Anstoßen der Retrieval-Aktivitäten nutzen.

Beheben der erkannten Probleme. Wir haben strukturelle Probleme beim Schreiben der Logfiles erkannt, die unsere Daten zwar nicht verfälschen, aber den Aufwand der Auswertung erhöhen. Wir werden das Logfile-Format geringfügig anpassen und das Schreiben von Events in das Logfile optimieren, indem wir weitere Kontextinformationen speichern. Das Problem der störenden Eingabedaten können wir nicht lösen, aber alle Stufen des Prozesses dahingehend verbessern, nicht nachvollziehbare Daten entsprechend zu markieren.

Evaluation und Weiterentwicklung der Interaktionsanalyse. Die in dieser Arbeit vorgestellten Konzepte haben sich in einer ersten Evaluation als hilfreich herausgestellt, müssen sich aber erst noch im Live-Einsatz beweisen. Dort wird sich zeigen, bis zu welcher Komplexität die Analyse von Benutzerinteraktion Sinn macht, oder ob einfache Heuristiken manchmal

zum selben Ziel führen. Im Verlauf der Arbeit entstanden neue Ideen für weitere Analysen, die neue Entwicklungen oder eine größere Zahl an Benutzern benötigen. Darunter sind z.B. eine detailliertere zeitbasierte Analyse der Benutzersessions, die Evaluation der neuen Operatoren zur Bewertung und manuellen Editierung von Fakten sowie die Projektion der hier vorgestellten Analysen auf eine detailliertere Ebene, um Qualitätsmaße nicht nur für Fakt-Typen, sondern auch auf der Objekt- oder Beziehungsebene repräsentativ zu ermitteln.

Evaluation und Weiterentwicklung des Benutzerinterfaces. Die Analyse der Nutzung hat gezeigt, dass das Web-Interface für einige Anwendungen nicht intuitiv genug ist. Zudem zeigt sich an der hohen Bounce Rate, dass das Interesse an komplexen explorativen Operatoren niedrig ist und die anzeigten Daten möglicherweise nicht relevant sind. Können wir die Kernfunktionen der Seite besser platzieren und verstärkt ihre Funktionalität kommunizieren, um das Interesse entsprechend zu lenken? Können wir die Ergebnisse der Interaktionsanalyse dazu nutzen, Fakten neu zu sortieren, ein- und auszublenden und gezielt interessante Operatoren mit den richtigen Parametern anzubieten?

Literaturverzeichnis

[1] Apache Hadoop. http://hadoop.apache.org (last visited 06/14/11).

[2] S. Arnold, T. Fiehn, and A. Löser. GoOLAP: interactive fact retrieval from search engines. 2011. Submitted to *20th ACM Conference on Information and Knowledge Management (CIKM) Demonstrations*, 2011.

[3] A. Aula and D. M Russell. Complex and exploratory web search. In *Information Seeking Support Systems Workshop (ISSS)*, 2008.

[4] M. J Bates. Where should the person stop and the information search interface start? *Information Processing & Management*, 26(5):575–591, 1990.

[5] C. Boden, T. Häfele, and A. Löser. Classification algorithms for relation prediction. In *DaLi Workshop at IEEE 27th International Conference on Data Engineering (ICDE)*, pages 46–52, 2011.

[6] C. Boden, A. Löser, C. Nagel, and S. Pieper. FactCrawl: a fact retrieval framework for Full-Text indices. In *14th International Workshop on the Web and Databases with ACM SIGMOD*, 2011.

[7] A. Broder. A taxonomy of web search. In *ACM SIGIR Forum*, volume 36, page 3–10, 2002.

[8] S. K Card, P. Pirolli, M. Van Der Wege, J. B Morrison, R. W Reeder, P. K Schraedley, and J. Boshart. Information scent as a driver of web behavior graphs: Results of a protocol analysis method for web usability. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, volume 3, page 498–505, 2001.

[9] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[10] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K. P Yee. Finding the flow in web site search. *Communications of the ACM*, 45(9):42–49, 2002.

[11] J. Hendler and J. Golbeck. Metcalfe's law, web 2.0, and the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):14–20, 2008.

[12] JAQL, Hadoop Query Processing Language. http://www.almaden.ibm.com/cs/projects/jaql (last visited 07/19/11).

[13] JSON (JavaScript Object Notation). http://www.json.org (last visited 07/19/11).

[14] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[15] T. Lin, O. Etzioni, and J. Fogarty. Identifying interesting assertions from the web. In *Proceeding of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 1787–1790, 2009.

[16] A. Löser, C. Nagel, S. Pieper, and C. Boden. Self-Supervised web search for any-k complete tuples. In *Second International Workshop on Business IntelligencE and the WEB with EDBT*, page 4–11, 2011.

[17] G. Marchionini. Exploratory search: From finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[18] F. Naumann, U. Leser, and J. C Freytag. Quality-Driven integration of heterogeneous information systems. In *Proceedings of the 25th International Conference on Very Large Data Bases*, page 447–458, 1999.

[19] O. Nov. What motivates wikipedians? *Communications of the ACM*, 50(11):60–64, 2007.

[20] OpenCalais. http://www.opencalais.com/documentation/calais-web-service-api/api-metadata/entity-index-and-definitions (last visited 06/14/11).

[21] D. E Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web*, page 13–19, 2004.

[22] T. G Rose, M. Stevenson, and M. Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31, 2002.

[23] G. Salton and C. Buckley. Term-Weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[24] R. W White and S. M Drucker. Investigating behavioral variability in web search. In *Proceedings of the 16th international conference on World Wide Web*, page 21–30, 2007.

[25] R. W White, B. Kules, S. M Drucker, and M. C. Schraefel. Supporting exploratory search. *Communications of the ACM*, 49(4):36–39, 2006.

[26] R. W White, S. M Drucker, G. Marchionini, and M. Hearst. Exploratory search and HCI: designing and evaluating interfaces to support exploratory search interaction. In *CHI'07 Extended Abstracts on Human Factors in Computing Systems*, page 2877–2880, 2007.

[27] M. L Wilson and M. C. Schraefel. Bridging the gap: Using IR models for evaluating exploratory search interfaces. In *SIGCHI 2007 Workshop on Exploratory Search and HCI*, 2007.

A. Anhang

A.1. Liste der Events im System GoOLAP

Event ID	*Symbol*	*Parameter*	*User interaction*	*Intention: The user...*	*System activity: The system...*	*Retrieval activity: The retrieval process..*	*Activity class*	*user goal*	*interest*	*coverage*	*completeness*	*correctness*	*cr. content*
PAGE VIEW EVENTS		**timestamp, sessionID, sourcePage, sourceObjectID, userID, referer, userAgent**			**displays a web page as result to a user request**								
ShowStartPageEvent	**S**		Access the GoOLAP start page	wants to start a new search	proposes interesting objects to the user		navigate						
ShowAnalyzePageEvent	**A**	objectID	Show AUGMENT result	requests an AUGMENT result for the object	displays the top rated facts and arbitrary documents for the object	retrieves more facts for the object	learn	X	X				
ShowComparePageEvent	**C**	objectType, factType	EXPAND list of objects	wants to compare multiple objects of the same type and clicks on the EXPAND icon in the column header	expands the list of objects and displays an interactive table view for comparison	retrieves more facts for the objects	learn	X	X		X		
ShowDocumentsEvent	**D**	factID	TRACE BACK the documents	wants to trace back the textual origin of a fact and clicks on the document symbol	displays a snippet from the source document and explains the origin of a fact by highlighting the sentence where the fact was extracted from	retrieves more evidences of this fact	lookup	X	X				
ShowNotFoundPageEvent	**N**	inputString	Search missing object	wants to augment facts for an object that is not yet contained in the GoOLAP fact base	displays a message to inform the user about the empty result	focuses on facts for the object	lookup		X	X			
SESSION INTERACTION EVENTS		**timestamp, sessionID, sourcePage, sourceObjectID, userID**			**interacts with the web frontend**								
RefererEvent	**r**	url, objectID	Click a link that points to GoOLAP	discovers a GoOLAP result page in another search engine and clicks on the link	displays the AUGMENT result page for the object	retrieves more facts for the object	navigate	X	X	X			
RefererGoogleEvent	**r**	url, queryString, objectID	Click a Google result that points to GoOLAP	discovers a GoOLAP result in Google and clicks on the link	displays the AUGMENT result page for the object	retrieves more facts for the object	navigate	X	X	X			
SearchObjectEvent	**s**	objectID	INTERPRET keyword search via autocomplete	wants to augment facts for an object displayed in the autocomplete and clicks on it	augments facts for the object and displays the AUGMENT page	retrieves more facts for the object	lookup	X	X	X			
SearchStringEvent	**s**	inputString, objectID	Search with keyword	wants to augment facts for an object and types its name into the search field	interprets the input and displays an autocomplete dropdown with multiple interpretations	ranks the order of interpretations when the	lookup	X	X	X			

Event ID	*Symbol*	*Parameter*	*User interaction*	*Intention: The user...*	*System activity: The system...*	*Retrieval activity: The retrieval process..*	*Activity class*	*user goal*	*interest*	*coverage*	*completeness*	*correctness*	*cr. content*
RESULT NAVIGATION EVENTS		**timestamp, sessionID, sourcePage, sourceObjectID, userID**		**interacts with a result page**									
ClickObjectEvent		objectID	Click on an object link in a fact table	clicks on an object that appears in a fact	displays the AUGMENT result page for the object	retrieves more facts for the object	lookup	X	X				
ClickProposedObjectEvent		objectID	Click on a proposed object link	clicks on an object that was proposed by the system	displays the AUGMENT result page for the object	retrieves more facts for the object	lookup	X	X				
ClickExpandEvent		objectID, factType	Click on the EXPAND icon in the fact table header	wants to compare multiple objects that share a relation	displays the EXPAND result page for the objectType and factType	retrieves more facts for the objects and factTypes	learn	X	X				
ClickURLEvent		factID, url	Click on the URL of a document	wants to trace back the original document of a fact	displays the source page (if it still exists)	retrieves more evidences of this fact	lookup	X	X				
COMPARE RESULT NAVIGATION EVENTS		**timestamp, sessionID, userID**		**interacts with the table view**									
CompareAddObjectEvent		objectID	Add a row to the table	wants to compare another object and adds it to the table	displays all fact columns for the object	retrieves more facts for the object	investigate	X	X	X		X	
CompareRemoveObjectEvent		objectID	Remove a row from the table	thinks the object is not related to the other objects in the table	removes the row from the table view	ranks interest on the object	learn	X				X	
CompareAddFactTypeEvent		factType	Add a column to the table	wants to compare the objects on a different facet	displays more facts for the given fact types	retrieves more facts of the given type	investigate	X	X		X		
CompareRemoveFactTypeEvent		factType	Remove a column from the table	wants to have a smaller result list	removes the column from the table view	ranks interest on the fact type	learn	X			X	X	
CompareEmptyCellEvent		objectID, factType	A cell with no data appears in the result	wants to compare facts that are not yet contained in the fact base	cannot find any facts for the given object and fact type	focuses on facts for the object of the given type	learn		X	X	X		

Event ID	*Symbol*	*Parameter*	*User interaction*	*Intention: The user...*	*System activity: The system...*	*Retrieval activity: The retrieval process..*	*Activity class*	*user goal*	*interest*	*coverage*	*completeness*	*correctness*	*cr. content*
FACT INTERACTION EVENTS		**timestamp, sessionID, userID**		**interacts with objects or facts on a result page**									
FactAgreeEvent		factID, factType	Agree to a fact	thinks that a fact is correct	displays a green icon on the fact	retrieves more evidences of the fact	investigate	X	X	X	X	X	
FactDisagreeEvent		factID, factType	Disagree to a fact	thinks that a fact is wrong	displays a red X on the fact	retrieves different facts for the object and fact type	investigate	X	X	X	X	X	
FactClearAgreeEvent		factID, factType	Reset agree state	reverts his opinion on the correctness	reverts the agree state of the fact	retrieves more evidences of the fact	investigate	X	X				
FactEditInsertEvent		factData	Insert a new fact	discovers a missing information in the result	displays the newly created fact	retrieves evidences of the new fact	investigate	X	X	X		X	X
FactEditOverwriteEvent		factID, factData	Overwrite an existing fact	discovers a wrong information in the result and overwrites it	displays the correct result additionally	retrieves evidences of the new fact	investigate	X	X		X	X	X
FactEditDeleteEvent		factID	Delete an existing fact	discovers that he inserted a wrong fact and deletes it	removes the fact from the view	ranks interest on the fact type	investigate	X	X			X	
SubscribeObjectEvent		objectID	Subscribe for updates on an object	is interested in a particular object and wants to receive updates	displays the objects in the user profile under "subscribed objects" and sends a notification on updates	focuses on facts for the subscribed objects	learn	X	X	X			
UnsubscribeObjectEvent		objectID	Cancel subscription	is not interested in updates for an object any more	removes the object from subscribed objects	ranks interest on the object	learn		X				
USER PROFILE EVENTS		**timestamp, sessionID, userID**		**uses profile features of the web frontend**									
UserSignupEvent		socialLogin	Sign up a GoOLAP user account	wants to use advanced features on the GoOLAP frontend and registers using an e-mail adress or social login	creates a new user account, sends a welcome message and logs in the new user		socialize	X					
UserLoginEvent			Log in to GoOLAP	logs in with an existing account	loads the user settings, displays notifications and updates the view		navigate	X					
UserLogoutEvent			Log out	logs out	ends the session and resets all saved states		navigate	X					

A.2. JAQL-Skript zur Analyse von Click Trails

```
/* Find most common behaviour patterns in log files */

// INPUT get all event logs in path and read them
eventLogs = ls("file:///data/goolap/events/events.log--2011-05*");
data = eventLogs -> transform read(file($.path));

// concatenate all event logs and distribute them over hdfs
data -> expand -> write(hdfs("/goolap/events.data"));
events = read(hdfs("/goolap/events.data"));

// FILTER botrequests and create incremental index to keep ordering
events = enumerate(events)  -> transform { index:$[0], $[1].* };
                            -> filter not $.botRequest==true;

// Array used to generate short symbols for the click trail
navigation =  [ { type:"EVENT_SHOW_START_PAGE",     symbol:"S" } ,
                { type:"EVENT_SHOW_ANALYZE_PAGE",   symbol:"A" } ,
                { type:"EVENT_SHOW_COMPARE_PAGE",   symbol:"C" } ,
                { type:"EVENT_SHOW_DOCUMENTS",      symbol:"D" } ,
                { type:"EVENT_SHOW_NOTFOUND_PAGE",  symbol:"N" } ,
                { type:"EVENT_REFERER",             symbol:"r" } ,
                { type:"EVENT_REFERER_GOOGLE",      symbol:"r" } ,
                { type:"EVENT_SEARCH_OBJECT",       symbol:"s" } ,
                { type:"EVENT_SEARCH_STRING",       symbol:"s" } ];

// FILTER only navigational events via equi-join and augment symbols and
// GROUP BY session and generate click trail
sessions =  join events, navigation
            where events.type == navigation.type
            into { events.sessionId , events.index , navigation.symbol }
         -> sort by [ $.sessionId , $.index ]
         -> group by sessionId = $.sessionId
            into { sessionId , trailLength:count($) ,
                   clickTrail:strJoin($[*].symbol,"") };

// GROUP BY click trail and find most frequent ones
trails =    sessions
         -> filter $.trailLength > 1
         -> group by clickTrail = $.clickTrail
            into { clickTrail , trailLength:$[0].trailLength , trailCount:count($) }
         -> sort by [ $.trailCount desc , $.trailLength desc ];

// Array with regular expressions used to search for frequent patterns
patternList = [ "(^r+[ADS]r?$)", "(r+S)", "(r+A)", "(r+D)", "(r+C)", "(?=AA)",
                "(AC)", "(?=CC)", "(s+N$)", "(Ss+A)", "(As+A)", "(s+Ns+A)"];

// CROSS trails and patterns to trailAndPattern (tp) and find each pattern
// in each trail and GROUP BY patterns with count
patterns =  trails
         -> expand each tp ( patternList -> transform each pattern { pattern , tp } )
         -> transform { $.pattern , $.tp.clickTrail , $.tp.trailCount ,
               matches:count( regex_extract_all(regex($.pattern),$.tp.clickTrail) ) }
         -> filter $.matches > 0
         -> transform { $.pattern , $.clickTrail ,
                        countMatches:($.matches * $.trailCount) }
         -> group by pattern = $.pattern
            into { pattern , patternCount:sum($[*].countMatches) };

// OUTPUT to a file
        [ { runDate         :   now() ,
            countLogfiles   :   count(eventLogs) ,
            countEvents     :   count(events) ,
            countSessions   :   count(sessions) ,
            sessionTrails   :   trails ,
            trailPatterns   :   patterns } ]
         -> write(file("/tmp/trailAnalytics.json"));
```