Beyond Search: Business Analytics on Text Data

vorgelegt von
Dr.-Ing. Alexander Löser
geboren 1976 in Meißen

Habilitationsschrift (Venia Legendi)

an der Fakultät IV- Elektrotechnik und Informatik
der Technischen Universität Berlin

Lehrgebiet: Angewandte Informatik

Gutachter:
Prof. Dr. Volker Markl
Prof. Dr. Gottfried Vossen
Prof. Dr. York Sure-Vetter

Eröffnung des Habilitationsverfahrens: 7.3.2012
Habilitationsvortrag: 30.10.2013
Datum des Fachbereichsratsbeschlusses
über die Zuerkennung der Lehrbefähigung:
30.10.2013

Berlin 2013
D 83

To my family

Abstract

The management of text data has a longstanding history in the human mankind. This book is intended for anyone who wishes to find out how text data will continue to affect enterprise applications or who wishes to create new business processes from analyzing text data on the Web and within the enterprise.

The book is divided into two parts:

- *Part I* gives an overview of techniques for searching, extracting and managing text data in an enterprise context, such as the intranet or corporate forums.
- *Part II* provides a more in-depth description of how we can execute structured queries over the sheer endless mass of textual data available on the Web in a cost effective and ad-hoc fashion.

Having structured information from text data available, with low costs for extracting and organizing, provides knowledge workers and decision makers in an organization with insights that have, until now, not existed.

The presented key scientific achievements base on real life observations from enterprise search systems within IBM, SAP, the exploratory search portal www.GoOLAP.info and from www.mia-marktplatz.de, a data marketplace for the '.DE Web'. A major part of the scientific results are being applied to improve technology in these systems and organizations.

I encourage you to visit the official Web site of this book. The website contains updates, papers, talks about ad-hoc Web research, and shows our running demo. You can access the book's Web site via: GoOLAP.info

Zusammenfassung

Das Management von Textdaten hat eine lange Geschichte in der Menschheit. Dieses Buch ist unterstützt Entscheider, Analysten und Anwender im Design von Enterprise-Anwendungen die Textdatenverarbeiten oder bei der Erstellung und Schärfung von Geschäftsprozessen die auf der Analyse der Textdaten im Internet basieren.

Das Buch ist in zwei Teile gegliedert:

- *Teil I* liefert einen Überblick über die wichtigsten Techniken für die Suche, Gewinnung und Verwaltung von Text-Daten im unternehmensinternen Kontext, wie im Intranet oder in einem Corporate Forum.
- Teil II beschreibt das Design von Technologien für die Ausführung strukturierter Anfragen auf der schier endlosen Masse von Textdaten im Web.

Die im Buch vorgestellten wichtigsten wissenschaftlichen Leistungen basieren auf Beobachtungen real existierdner Enterprise-Such-Systeme. Dazu gehören das Intranet der IBM, das Intranet der SAP AG, das Portal www.GoOLAP.info und die Analyseplattform www.mia-marktplatz.de. Die wissenschaftlichen Ergebnisse des Buches wurden in diesen Systemen angewendet.

Bitte besuchen Sie auch die offizielle Website www.GoOLAP.info des Buches mit Updates, wissenschaftliche Arbeiten und einer Demo.

Acknowledgements

Writing a book always involves more people than just the author. I would like to thank the members of our Database and Information Systems Group at the Technische Universität Berlin in Germany, my former colleagues at SAP AG and my former colleagues at the IBM Almaden Research Center. Sebastian Arnold, Christoph Boden, Thomas Häfele, Tillmann Fiehn, Steffen Lutter, Christoph Nagel and Stephan Pieper co-authored many papers. I had the pleasure to work with these bachelor, master and PhD students at Technische Universität Berlin over the last two years in the field of ad-hoc Web research. My former colleagues Wojciech Barczynski, Dr. Henrike Berthold, Dr. Falk Brauer, Dr. Hong-Hai Do, Dr. Gregor Hackenbroich, Klemens Muthmann and Marcus Schramm from SAP AG also made significant contributions to managing enterprise text data. I'm grateful to Dr. Shivakumar Vaithyanathan, my former manager, and to my former colleagues Sriram Raghavan, Huaiyu Zhu and Rajasekar Krishnamurthy at the IBM Almaden Research Center for sharing both, insight and pleasure while working on information extraction, email and intranet search.

Additionally, writing this book would not have been possible without the help of many colleagues at European projects OKKAM, ROBUST and SCAPE, in particular Prof. Paolo Bouquet, Prof. Steffen Staab and Dr. Ross King. I would like to thank Dr. Paul Hofmann and Prof. Wolfgang Lehner for their valuable support when I was working with SAP AG.

Finally, I would like to thank my supervisors Prof. Volker Markl from Technische Universität Berlin, Prof. Gottfried Vossen from University of Münster and Prof. York Sure-Vetter from University of Koblenz-Landau who have been coaching me over the past years, accompanying this work generously both intellectually and in terms of organizational backin. I also thank Prof. Odej Kao, Prof. Anja Feldmann and Prof. Uwe Nestmann for being part of the habilitation committee and for sharing valuable insights.

Contents

1 Introduction

From the earliest days of computers, search has been a fundamental application that has driven research and development. Today's data retrieval applications range from database systems that manage the bulk of the world's structured data to Web search engines that provide access to petabytes of text data. As the Internet has become a mass medium, searching has become a daily activity for everyone from children to research scientists. On the other hand, hundreds of millions of active users generate searchable content in online-forums, blogs, tags and wikis. Example analysis questions of these individums cover areas, such as competitive market analysis, business decision making or the analysis of medical treatments. The following example illustrates a typical question of potential search session:

Competitor analysis: Show competitors of Boeing. *Identify on cnn.com and ft.com potential customers of these competitors for "Boeing.com". List and aggregate the value of planed deals for each competitor, from today on.*

Neither 'classic search technology portals', such as Google or Bing, as well as commercial BI analysis systems, such as "SAP BW" or 'Crystal Reports', support such analysis queries over text data, which lies at the intersection of their capabilities. Therefore, most content producers, mostly the ordinary end-consumers but also other kinds of commercial knowledge workers, are not able to drill down into that "knowledge" yet.

Computer scientists have recognized the theoretical importance and practical relevance of assoziated problems when answering such queries (see also [38][43][59][66][102] and the vision of the author of this book in [133][136]). The methods and technologies in this book contribute to the grand challenge of answering analytical queries on text data.

1.1 Trichotomy of Web Search Intentions

Currently, the most popular method to answer the example queries from the former section is the manual implementation of an extraction and analysis system, e.g., with UIMA [52] and a relational database system. Often, 'ordinary people', smaller companies or departments in large companies

are not able to execute such projects, because people that can manage the technical complexity of such systems. Another core problem is the high cost for the implementation and the extraction of data as well as high set up times. As a result such projects are often not started, although the analysis of the text data would provide valuable additional information for business critical decisions.

Vendors of search engines are well aware of the demands of users, which is often expressed through search query sessions. In the 1990'ies and early 2000 these vendors have started analyzing systemantically query logs and identified *lookup searches* as a first common query intention class among all search queries**.** Following the study of the authors of [104], lookup search is the most basic kind of search tasks and has been the focus of development for database management systems and much of what Web search engines support. Lookup tasks return discrete and well-structured objects such as numbers, names, short statements, or specific files of text or other media. Database management systems support fast and accurate data lookups in business and industry; in journalism, lookups are related to questions of who, when, and where as opposed to what, how, and why questions.

Throughout this book we utilize the following trichotomy of lookup search types (see also work in [1] [9][10] [16] [30][104]):

Navigational searches *are intended to find a specific web site that the user has in mind.*

Authors of [16] define the navigational goal as to go to a specific known website that I already have in mind, for example the query *[aloha airlines]* or *[duke university hospital]*. The only reason for searching is that it's more convenient than typing the URL or perhaps the searcher does not know the URL. The same authors aruge, that most queries consisting of names of companies, universities, or well-known organizations are considered navigational because the intended target is a single URL, while most queries for people – including celebrities – are not. A search for celebrities such as *[Cameron Diaz]* or *[Ben Affleck]* typically results in a variety of

fan sites, media sites, and so on; it's unlikely that a user entering the celebrity name as a query had the goal of visiting a specific site.

Informational searches *are intended to find information about the query topic.*

Following authors of [16], this rather broad category includes goals for ***answering questions (both open- and closed)*** that the user has in mind, ***requests for advice, such as*** *[help quitting smoking]*, and "***undirected" requests, such as*** *[duke university hospital]* to simply learn more about a topic. These undirected queries may be viewed as requests to "find out about" or "tell me about" a topic; most queries consisting of topics in science, medicine, history, or news qualify as undirected, as do the celebrity queries mentioned above. This class also includes the desire to locate something in the real world *[phone card]*, or simply get a list of suggestions for further research, such as *[baby names]*. Most shopping queries have the "*locate*" goal – I'm searching the web for X because I want to know where I can buy X.

Finally, this class also contains ***complex aggregation queries***. The query *[average emergency room nurse wages in Virginia]* may origin from a nurse that seeks comparing her salary to other nurses in a location called 'Virginia' (depending on different features such as browser language or IP address, this location might be most likely in the US). The searcher expects that the search engine can parse the query or at least may be able to understand the intention. However, the current state of the art of search engines will likely seek only for documents that may contain discriminative keywords from the query in documents. If the searcher is very lucky, the search engine might return a single, published document that already contains an aggregated view answering the query. Another analytical query of this kind is *[best ipods and mp3 players]*, which intends to retrieve an ordered list (for the vague critera 'best') of music a union of playing decives, grouped by products from the Apple called IPOD and other products.

Transactional searches *are intended to perform a web-mediated activity.*

Authors of [16] describe that resource (or transactional) queries represent a goal of obtaining something (other than information). They distingish between four types of transactional queries, obtain, download, entertain, and interact: The goal is '*obtain*' if the resource is something the user plans to use in the offline world, such as song lyrics *[django minor swing guitar sheets]*. If the resource is something that needs to be installed on my computer or other electronic device to be useful, the goal is *'downloading'*, for example *[kazaa lite]*. If the users goal is simply to experience (typically view or read) the resource for my enjoyment, the goal is '*entertain*', such as the query *[free movies]*. Finally, the '*interac*t' goal occurs when the intended result of the search is a dynamic web service (such as a stock quote server or a map service) that requires further interaction to achieve the user's task, an example query is *[measure converter]*.

1.2 The Public Web vs. the Corporate Web

Throughout the last years search engine vendors started to investigate technologies for answering navigational, informational and transactional queries on both, the general public Web and the corporate Web. However, transferring a high performing search algorithm, developed in the laboratory of a Web search vendor, to a corporate search engine is far from trivial. As authors in [5] state:

The corporate intranet is an organism that is at once very similar to and very unlike the Internet at large. A well-designed intranet, powered by a high-quality enterprise search portal, is perhaps the most significant step that corporations can make to improve productivity and communication between individuals in an organization.

The work in [5] is one of the chapters first which systematically investigated the differences of Web search and search within a corporate environment. Basically, the work stated the following important axioms:

Axiom 1: Intranet documents are often created for simple dissemination of information, rather than to attract and hold the attention of any specific group of users.

A fantastic aspect of a web search engine is that any query will return a reasonable number of documents. The sheer endless number of available documents for 'every' topic orignes from hundreds of millions of commercial and non-commercial content producers (and content replicators) on the Web. Often, these human efforts are supported with addditinal mechancis which replicate existing records from data bases into a textual representation on a Web page. Contrary, content on the corporate Webs is often published by a relatively small percentage of people that have access to this resource. Since these people are paid by the company, often these people focus their content production on the most critical topics only and do not pay attention for adding mechanics to a page that may help to attract a continuous visitor flow to the page. As a result, typical 'mechanics' for identifying pages on the public Web will not work for the corporate Web. One example for solving navigational search querys are query-independent methods, like measuring page in-degree, methods based on the URL-type, and variants of PageRank [18]. Because of content production process on the corporate Web, most pages have, if at all, very few links. Worse, mostly these links are outgoing links to external pages on the public Web or link to islotaled pages from the core of the corporate Web. Therefore, the common technique of PageRank for identifying authoritative pages for answering a navigational query on the public Web, through ingoing and outging links, will not work well on the corporate Web.

Axiom 2: A large fraction of queries tend to have a small set of correct answers (often unique), and the unique answer pages do not usually have any special characteristics.

On the corporate Web even common queries, such *[travel reimbursements]* only return a single answering page, which is often good enough for the searcher. Therefore, solving critical search problem, such as matching a semantic vocuabulry mismatch, is somehow simpler on the corporate Web, since both, content producer and searcher somehow already agreed on a common language within the corporate setting. On the other hand, the single page that may answer a search page is not distinguished in any special way that makes it easily identifiable as relevant for answering a query.

Axiom 3: Intranets are essentially spam-free.

Not all is negative about intranets, though. Clearly, there is no incentive for an employee to advertise products on the corporate Web. As a result, even simple techniques that will fail in the public Web, such as anchor text analysis, will help to retrieve fairly reliable in the corporate Web.

Axiom 4: Large portions of intranets are not search-engine friendly.

Finally, often company uses automated content generation approaches for the corporate Web, such as generating pages from relational databases or ERP systems. These pages can often only accessed through specific interfaces, portals which faces an additional problem for a corporate search engine. Worse, a search request may not reach these additional pages, because the searcher may miss additional security credentials. As a result of this general problem, the set of accessible and searchable pages is reducued.

1.3 Challenges and Outline

The main contribution of this book is the integration of text data into analytical decision systems. In Part I we focus on navigational and informational queries on textual data that has been created in the corporate Web. In Part 2 we consider textual data from the general Web for answering informational queries. Here, we outline the book through major research challenges and summarize our contributions for solving these challenges.

In Part I 'Searching the corporate Web' we overview novel query processing techniques for answering navigational and information keyword queries on text data from the corporate Web. We divide this problem into several sub problems:

Challenge 1: What are typical search query intentions in the corporate Web? Do queries from an enterprise search engine follow the same intention as queries for ordinary Web search engines?

In Chapter 2 we investigate common search intentions for corporate Web forums. Our study is based on a query log analysis from the SAP software developer network that is utilized by more than one million SAP customers currently. The study confirms that users have mostly informational search intentions, such seeking advice for a problem or requesting information

about an object. Our analysis also unravel that users utilize corporate search engines to navigate to previously seen pages, such as product- or user home pages. To a significant lesser extend users utilize corporate search engines with a transactional intent, such as to download software. We also notice that the process of creating questions and answers in a corporate Web forum is often moderated. Therefore *spammers* are rather the rare case and most intentions aim to resolve business critical problems.

Challenge 2: Approaches for resolving navigational queries in the general Web, such as PageRank, are not directly applicable to search engines for the corporate Web. What is an appropriate strategy for resolving navigational queries in the corporate Web, in particular in the absence of dense link structures?

In Chapter 3 we focus on methods for resolving navigational queries in an intranet. We address the problem of providing high quality answers to navigational queries in the intranet (e.g., queries intended to find product or personal home pages, service pages, etc.). Our approach is based on offline identification of navigational pages, intelligent generation of term-variants to associate with each page, and the construction of separate indices exclusively devoted to answering navigational queries. Using a test bed of 5.5 million pages from the IBM intranet, we present evaluation results that demonstrate that for navigational queries, our approach of using custom indices produces results of significantly higher precision than those produced by a general purpose search algorithm.

Challenge 3: Current corporate forum search engines lack the ability to group near duplicate threads in search results. As a result, forum users (which are often potential customers) are overloaded with duplicated content in search results. As a result these users create additional duplicate postings. How can we identify near duplicate threads from human generated content in corporate forums?

In *Chapter 4* we identify common reasons leading to near-duplicates on a large developer forum with more than one million users. Based on our findings we design and test a new near-duplicate detection algorithm for forum threads. In our experiments we report that our algorithm outper-

forms existing algorithms drastically and that we are able to group forum threads with a precision of 74%

In Part II 'Ad-hoc Web research' we introduce, discuss and provide solutions for the novel problem of supporting *situational business intelligence* queries over web-based textual content. Often these queries have an informational query intention. In particular many of these queries aim to retrieve fact rich pages, to extract facts from these pages and to aggregate facts into a common view. Neither conventional search engines nor conventional Business Intelligence and ETL tools address this problem, which lies at the intersections of their capabilities. Chapter 5 introduces so called situational applications for issuing and solving such query intentions. These applications are usually short-lived programs that are created for a small group of users with a specific business need. The second part of this book gives an informative and practical look at the following underlying research challenges and our solutions for supporting these applications:

Challenge 4: How can we retrieve documents that likely contain factual information from a lexical index with limited access, such as from a Web search engine?

A common task of Web users is querying structured information from Web pages. Chapter 6 focuses on the important problem of leveraging existing index structures from Web search engines with structured queries which is an often requested, but still unsolved requirement of many Web users. A major challenge in this scenario is identifying keyword queries for retrieving relevant pages from a Web search engine. In Chapter 6 we solve this challenge by automatically generating keywords. Our approach is based on the observation that Web page authors have already evolved common words and grammatical structures for describing important relationship types. Each keyword query should return only pages that likely contain a missing relation. Therefore our keyword generator continually monitors lexical phrases from processed Web pages during query execution. Thereby, the keyword generator infers significant and non-ambiguous keywords for retrieving pages which likely match the mechanics of a particular relation extractor.

Chapter 7 presents the BlueFact system which extends techniques for keyword-based fact retrieval from Chapter 6. BlueFact observes keywords using structural, syntactic, lexical and semantic features in an iterative, cost effective training process, combines observed keywords as evidence for a factual information. Only based on observations from the index and without downloading the document, BlueFact infers a novel factual relevance score for each document identity. Finally, BlueFact forwards the documents in the order of their estimated factual relevance to an information extraction service. That way BlueFact can efficiently retrieve all the structured, factual information contained in an indexed document collection. We report results of a comprehensive experimental evaluation over 20 different fact types on the Reuters News Corpus Volume I (RCV1). Blue Fact's novel scoring model and its feature generation methods significant outperform existing approaches in terms of fact retrieval performance. BlueFact fires significant fewer queries against the index, requires significant less execution time and achieves very high fact recall across different domains.

Challenge 5: Often factual information on a single page misses certain attributes for answering structured query. How can we retrieve and extract these missing attributes from additional pages?

In Chapter 8 we present a novel query processor for systematically discovering any-k relations from Web search results with conjunctive S-P-J queries. In this scenario the *'any-k' phrase* denotes that retrieved tuples are not ranked by the system. For realizing this interesting scenario the query processor transfers a structured query into keyword queries that are submitted to a search engine, forwards search results to relation extractors, and then combines relations into result tuples. Unfortunately, relation (or fact) extractors may fail to return a relation for a result tuple. We propose a solid information theory-based approach for retrieving missing attribute values of partially retrieved relations. Moreover, user-defined data sources may not return at least k complete result tuples. To solve this problem, we extend the Eddy query processing mechanism for our 'querying the Web' scenario with a continuous, adaptive routing model. The model determines the most promising next incomplete row for returning any-k complete result tuples at any point during the query execution process. We report a

thorough experimental evaluation over multiple relation extractors. Our experiments demonstrate that our query processor returns complete result tuples while processing only very few Web pages.

Challenge 6: Filtering potentially irrelevant pages may further reduce the amount of forwarded pages to an extractor. What is the design of a fast and effective filter for the task of fact extraction?

In Chapter 9 we propose a relation predictor which filters out irrelevant pages and further can speed up the overall information extraction process. As a classifier, we train a support vector machine (SVM). We consider pages on a sentence level, where each sentence is transformed into a token representation of shallow text features. We evaluate our relation predictor on 18 different relation extractors. Extractors vary in their number of attributes and their extraction domain. Our evaluation corpus contains more than six million sentences from several hundred thousand pages. We report a prediction time of tens of milliseconds per page and observe high recall across domains. Our experimental study shows that the relation predictor effectively forwards only relevant pages to the relation extractor. We report a speedup of at least factor two while discarding only a minimal amount of relations. If only fixed amount, e.g. 10% of the pages in the corpus are processed, the predictor drastically increases the recall by a factor of five on average.

Challenge 7: What are common user activities beyond fact retrieval?

In general, "fact retrieval searches are suited to analytical search strategies that begin with carefully specified queries and yield precise results with minimal need for result set examination and item comparison. As the Web has become the information resource of first choice for information seekers, people expect it to serve other kinds of information needs and search engines must strive to provide services beyond lookup.We present GoOLAP, a system that aims to automate common user activities beyond fact llokup from Web search engines. The system provides powerful operators for the fact retrieval task on textual information, such as augmenting facts for an object, tracing back the textual origin of a fact or comparing factual information for a list of objects. As a natural alternative to

crawling a large proportion of the Web, GoOLAP interprets user interactions as input to identify missing facts. These user interactions trigger a fact retrieval process with the goal to populate the GoOLAP fact base from Web-scale indices of existing search engines in an ad-hoc fashion. This process is powered by the FactCrawl engine that leverages sophisticated keyword generation techniques (see Chapter 7) and our page classification techniques (see Chapter 9) to retrieve only pages that likely contain missing and rare factual information. (3) GoOLAP combines these approaches to drastically avoid crawling, indexing and extracting potentially billions of irrelevant pages.

We address and solve these challenges during the course of this book.

Part I

Searching the Corporate Web

2 Query Intentions in Corporate Web-Forums

This chapter provides a study about common query intentions in a corporate Web-forum. It is based on a query log analysis from the SAP software developer network that is utilized by more than one million SAP customers. Our study confirms that users have mostly informational search intentions, such seeking advice for a problem or requesting information about an object. Our study also unravels that users utilize forum search engines to navigate to previously seen pages, such as product- or user home pages. To a significant lesser extend users utilize forum search engines with a transactional intent, such as to download software.

The content and ideas of this chapter are published in [135].

2.1 Introduction

Corporate portals often are the dominant source of information for employees, customers and other company affiliates. One example is SAP's software developer network (SDN) at http://sdn.sap.com, SAP's premier community site for SAP developers, experts and software engineers. Users have access to documents about products either by browsing a pre-defined static taxonomy of software systems or use a full text search engine. The ultimate goal of any search system is to answer the intention behind the

Sample characteristics	#
total number of queries	470.973
number of empty queries	62.603
unique queries before normalization	14.408
unique queries after normalization	12.608

Figure 1: Queries before and after normalization

query. However, a current problem in forum search engines is the low result precision for keyword based search queries. For instance, Figure 17 shows the top 10 queries for March 2007 for SDN. Even though the correct result for all of the queries is available via the SDN portal, none of the right results was mentioned in the highest ranked 30 result links from the search engine.

Query	Frequency	% of Queries
ruby	40,098	2.36
firebug	12,546	0.74
solution manager	2,070	0.12
workflow	1,515	0.09
abap	1,455	0.09
wiki	1,282	0.08
visual composer	1,270	0.07
smartforms	1,217	0.07
idoc	939	0.06
alv	928	0.05
XI	816	0.05

Figure 2: Top-10 queries in March 2007 for SAP's forum

In this study we focus on identifying common intention classes for informational queries based on in a corporate forum. The following study bases on a detailed query log over a period of four weeks in May 2007. The study unravels quantitative characteristics and unravels the four most popular user intentions. This chapter is organized as follows: in Section 2.2 we give brief analysis on query logs and in Section 2.3 we unravel common user intentions. Finally, Section 2.5 presents related work.

2.2 Query Normalization

We use a sample from SDN query logs of May 2007, including 470.973

Term size	# total queries	[%] total	# unique queries	[%] unique
1	111669	83	136	68
2	20845	16	55	27
3	1754	0.7	7	4
4	0	–	0	0
>4	419	0.3	2	1
total	134687		200	

Figure 3: Query term size for top-200 queries

Term size	# total queries	[%]total	#unique queries	[%]unique
1	3982	45	80	40
2	3034	33	59	30
3	1359	15	36	18
4	419	5	18	9
>4	145	2	7	4
total	8939		200	

Figure 4: Query term size for random 200 infrequent queries

total numbers of queries and 12.609 unique entries. Figure 15 and Figure 16 show a sample of the top 10 queries. We normalize queries in a process that includes multiple operations, such as trimming white spaces at the beginning and the end, lowercasing query terms and excluding empty queries. After normalizing queries we received 12.608 unique queries that are submitted 408.370 times.

2.2.1 Query Distribution

We observe a long-tailed distribution of queries and query frequency [26]: A very few queries are very common, but we also observe that most of the workload is on queries that individually occur rarely. Therefore we created two subsets: Query subset Q1 includes the top 200 most frequent queries (see also Figure 17) and query subset Q2 includes 200 queries randomly chosen from the less frequent queries (see also Figure 18). The probability for choosing an infrequent query was counted from its frequency divided by the sum over all other infrequent queries.

2.2.2 Term Size Distribution

Figure 3 and Figure 4 show term size distributions: One keyword queries are highly common. 68% of the most common and 40 % of the less common unique queries are one keyword queries. Hence, if we can optimize the search engine to the top 137 one keyword queries, the search system could give an exact answer to nearly one quarter of the search requests.

2.3 Common Query Intentions

Recent research [16][14] has shown the importance of understanding the query intention. In this section we analyze how often users a request matches a particular intention class and map intention classes to common extraction technology.

To provide an accurate measure of user intention study, for each query, two investigators manually determined independently from each other the most relevant answer for the query and defined this information in a gold standard. For spotting the answer documents the investigators used the current SDN search engine as initial seed answers and conducted further browsing. For most queries only one answer was found. Some queries could be mapped to more than one intention classes. E.g., the query *[widget]* could refer to a *list query* -informational; or a *product site* (SAP widgets) - navigational.

Following [16] we structured our query set into navigational, informational and transactional queries. For the special case of a software developer portal we added further sub-classes:

2.3.1 Navigational Queries

These queries are directed towards navigating to previously visited pages.

Queries for product sites. Typical queries are *[solution manager],[visual composer]* with the intention to navigate towards the product web page or to visit a previously seen page again. Sometimes also abbreviated product names are used, such as *[XI]* for *[exchange infrastructure]*.

Community portal sub-sites. These queries intent to visit sites related to the structure of the portal. Example queries are *[wiki], [blogs]*. The intention of these queries is to navigate to site directly using a search query instead of browsing the link structure.

Query Intention	Top 200 queries	Less frequent 200 queries
Navigational		
Product home pages	41	9
Abbr. Product pages	34	1
Sub-sites	2	3
Developers Notes	0	1
Informational		
What is/How to	36	73
List technical concepts	33	27
Configuration Advice	1	12
(Error) Messages	2	8
Code Fragments	4	3
Locate documents	2	10
Configuration	0	3
Transactional		
Download requests	9	2
Unclear	36	48

Figure 5: Common search intentions in a developer forum

Queries for developer notes. 'Developer notes' describe solutions for problems. These specific documents are identified with a six to seven digit number. Such numbers are infrequently used as search request e.g., *[701654]*.

2.3.2 Informational Queries

The major goal of a developer portal is supporting software developers with advice and support for common and rare software problems. Another important goal is forming an expert community that provides qualified responds. Both tasks, listing products and resolving problems, are typical candidates for informational queries. In our sample we discovered several types of informational queries:

Closed queries using question words. Users submit these queries and intend to find web pages for solving a specific problem. Common question words, such as [how to] or [what is] are used. E.g., an example query is *[How to create an XSLT mapping file]*.

List queries. These queries intend to list common products given a technical concept without explicitly mentioning (or even knowing) the name of a particular product. E.g., the query *[data mining]* intents to list data mining products like *SAP Business Intelligence suite* or *SAP Accelerator*.

Advice queries. Typical advice queries request instructions for installing software products, such as *[solution manager configuration]*.

Locate queries. Often developers ”copy & paste” messages directly from an applications or from code files. Query examples are (error) messages, such as *[OBJECTS_OBJREF_ NOT_ ASSIGNED]*, code fragments, such as *[MESSAGE_ TYPE_X]* or request information on configuration parameters, such as *[login/create_sso2_ticket]*. The search goal of such queries is to locate technical documents or relevant forum threads, where the query request is mentioned.

2.3.3 Transactional Queries

The portal supports typical transactions for software products, e.g. users may download or upgrade software. In this study we could only count those download requests where potential software was available for download. We assume that more queries intent towards downloading software.

2.4 Detailed Observations

Figure 5 shows detailed information about common query intentions. Our study shows that informational queries are the most often requested class of infrequent queries. Transactional queries (at least for software products) appear quite infrequently. Finally, navigational and informational queries are nearly equally often issued. This observation is contrary to a typical query intention distribution in an intranet [16], where navigational queries are the most dominating class among the frequent queries. Our explanation for these findings is that a developer portal provides many documents on

solving problem with software, but the portals typically provide only a few software downloads. For the rest of our analysis we focus in the most often requested query classes:

Navigational queries. 91 out of 400 unique queries have one of the following navigational intentions:

- *Queries for product home pages and sub sites.* 85 queries belong to navigational queries for product home pages. Both, long and abbreviated forms are equally often used. After we inspected pages that are requested by the query we observe that page authors often mention on the page a discriminative term, e.g. (abbreviated) products, error messages, or names for sub sites of the portal. Also, often in *wikis*, *forum* entities appearing in a query also appear in the URL and the title of the page. Therefore such navigational queries could be resolved by spotting and indexing navigational pages *a priori* [141]. For each spotted navigational page n-grams titles, URLs and anchors, are extracted and stored in a separate index. A query is matched against the index and the top results of the most relevant types are returned. If the query does not match this special-case-index, results from the organic search engine are returned.

- *Developer note queries.* These queries aim to navigate towards a specific document type given a unique document identifier. Recognizing such queries is simple, since the query term contains a 6 digit number. Developer portals mark developer notes with the phrase *developer note* in the title which is followed by the six digit number used by the developer to identify the note.

Informational queries. 215 out of 400 unique queries have one of the following informational intentions:

- The purpose of a [*What is/How To]* query is to obtain a definition about a product or technology. We manually inspected pages that answer *What is* queries in our study: 101 **[***What is/HowTo]* queries are answered with a *wiki* document and 14 *[Whatis/How to]* queries are resolved with a *forum thread*. E.g., for query *[lo extraction step by*

step] the answer can be found at thread title *Lo extraction*. *Forum* and the *Wiki* are moderated by an expert and provide a consistent structure.

- *System messages queries.* Queries of this type include error and system messages. To create a query request, most users just copy a system message to the search engine to locate more information e.g., why this message has been thrown. We also observe that messages queries either have a length of more than five terms, such as *[A pop-up window was blocked in visual composer]* or a length of one keyword, such as *[DATASET _ WRITE _ ERROR]*. We observed that for most of these queries the relevant answer was discussed in a thread of the forum.

- *List further information for given technical concept.* 60 queries are related to list further information for given technical concept. These queries are slightly more issued among the frequent queries than among the infrequent queries.

Unclear query intention. 84 out of 400 queries have an unclear query intention. We marked a query as unclear if the term is vague or extremely ambiguous, such as *[performance]* or *[install]*.

2.5 Related Work

There are two broad research areas of work that are relevant to the work presented in this chapter:

Understanding search goals. The classification of search queries into navigational, transactional, and informational was originally proposed in [1]. Several examples and scenarios for each class of queries in the context of enterprise search are described in [7] and a more recent analysis of user goals in Web search is presented in [16]. Authors in [22] propose classification techniques for automatic user goal identification. Identifying transactional queries in the Intranet is also investigated in [15]. Authors of [141] resolve navigational queries in the intranet.

Text analytics. Text analytics is a mature area of research concerned with the problem of automatically analyzing text to extract structured information. Examples of common text analytic tasks include *entity identifica-*

tion (e.g., identifying persons, locations, organizations, etc.) [20], *relationship detection* (e.g., person X works in company Y) [25][27] and *co-reference resolution* (identifying different variants of the same entity either in the same document or different documents) [24]. Text analytic programs used for information extraction are called annotators and the objects extracted by them are called annotations. Traditionally, such annotations are directly absorbed into applications. A prominent example is the AVATAR Information Extraction System (IES) which tackles some of these challenges [28].

2.6 Conclusion

Current corporate search portals are not able to capture the user intention and thus often confront users with imprecise answers.

Based on a large query log for a corporate portal we identified search intentions for product home pages, sub sites as common navigational query intentions and *[What is]* and *[HowTo]*, system messages queries as common informational queries as most common.

Navigational and informational query intentions can be resolved with specific extraction and indexing techniques. In the following two chapters we introduce strategies for resolving these query intentions.

3 Navigating the Intranet with high Precision

Despite the success of web search engines, search over large enterprise intranets still suffers from poor result quality. Earlier work that compared intranets and the Internet from the view point of keyword search has pointed to several reasons why the search problem is quite different in these two domains.

In this chapter, we address the problem of providing high quality answers to navigational queries in the intranet (e.g., queries intended to find product or personal home pages, service pages, etc.). Our approach is based on offline identification of navigational pages, intelligent generation of term-variants to associate with each page, and the construction of separate indices exclusively devoted to answering navigational queries.

Using a test bed of 5.5M pages from the IBM intranet, we present evaluation results that demonstrate that for navigational queries, our approach of using custom indices produces results of significantly higher precision than those produced by a general purpose search algorithm.

The content and ideas of this chapter are published in [141].

3.1 Introduction

The ultimate goal of any search system is to answer the need behind the query. Analogous to Broder's taxonomy of Web search, queries on the intranet can also be classified as informational, navigational or transactional [1]. In this chapter, we present our approach for spotting and resolving "*navigational queries*" in the intranet. The purpose of such queries is to reach a particular page that the user has in mind, either because they visited it in the past or because they assume that such a page exists. We refer to such pages as "*navigational pages*". Typical examples of navigational pages in the IBM intranet include the homepage of a person, an authoritative page for a service such as Employee Assistance Program, the homepage of a department such as 'The Center for Business Optimization', or the internal home page for the 'DB2 UDB' product. Popular web-search engines routinely answer such navigational queries. For instance, if the user query is the name of a person, the top-ranked results from most search en-

gine are predominantly user homepages. Unfortunately, this does not imply that navigational search in the intranet is a solved problem.

Indeed, the differences between intranets and the Internet (from the view point of search) were aptly captured in four axioms presented in [5]. We draw particular attention to axioms 2 and 4 that we repeat below for convenience:

Axiom 2: A large fraction of queries tend to have a small set of correct answers (often unique) and the unique answer pages do not usually have any special characteristics.

Axiom 4: Large portions of intranets are not search-engine friendly.

The first part of Axiom 2 essentially describes navigational queries and highlights the difficulty in identifying such pages in the intranets. Axiom 4 refers to multiple challenges. For one, it refers to queries for company-specific information such as employees, projects, product names, acronyms, etc. It also reflects the difficulties encountered in a geographically disperse corporation with very local, organization-specific and site-specific terminologies and queries.

Consider, for example, a user seeking to find the "`Global Technology Services`" homepage. The most common query for this task is '`gts`'; this is one of the top 4000 unique queries on the IBM Intranet. Using standard web-IR techniques — essentially a combination of link-analysis and IR ranking heuristics — the top 50 hits on the IBM intranet do not contain the correct answer [6]. On the other hand, *a priori*, if the Global Technology Services homepage had been identified and associated with the term '`gts`', the task of the search engine at query time is extremely straightforward. Identifying these candidate pages before a search query is issued is precisely what we set out to accomplish.

Our proposed approach consists of an offline process in which we recognize all navigational pages and associate appropriate term variants with each page. It turns out that during the actual process of identifying a navigational page we get more information than simply whether a page is navigational. In particular, depending on the sequence of analysis steps that is

used to identify them, navigational pages are placed into one of several "*semantic buckets*" (e.g., there is a semantic bucket that holds all of the personal home pages). For each bucket, we build a standard inverted index using the terms and variants associated with the set of pages in that bucket — we refer to this index as a "*navigational index*". At runtime, a given search query is executed on all these navigational indices and the results are merged to produce an answer to the navigational query.

Focus on precision. Since we are concerned only with navigational queries where the goal is to identify the one or few right answers to each query, precision is of the utmost importance. Our overall approach, a number of design choices, as well as the evaluation results that we present in the subsequent sections are guided by this emphasis on precision.

Contributions. This Chapter gives four primary contributions:

1. We propose a solution to the problem of answering navigational queries on the intranet. Our solution is based on off line identification of navigational pages, generation of term-variants to associate with each page, and the construction of separate indices exclusively devoted to answering navigational queries.

2. We propose a generic templatized procedure for identifying navigational pages using a sequence of "*local*" (intra-page) and "*global*" (cross-page) analysis. We illustrate this template with several specific local and global analysis algorithms.

3. We consider the problem of filtering and ranking the results of navigational queries based on user profiles. In this context, we present a technique for answering geo-sensitive navigational queries, i.e., queries for which the correct result page depends on the geography of the user posing the query.

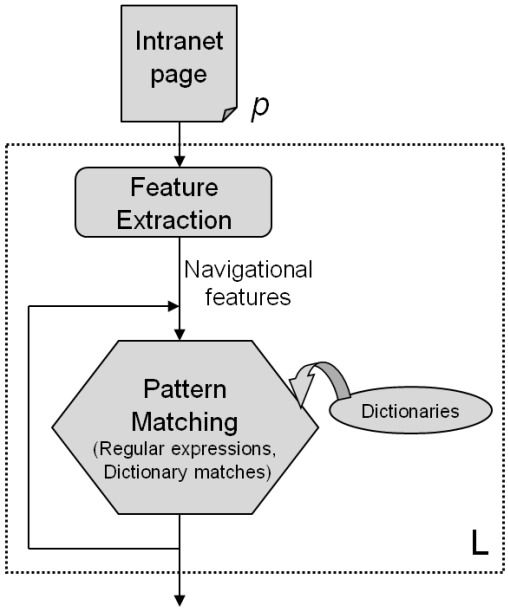


Figure 6: Local Analysis Template

4. Using a collection of 5.5 million pages from the IBM intranet and a set of real intranet user queries issued within IBM during the period June–August 2006 as our test bed, we report evaluation results that validate the efficiency of our approach.

3.2 Identifying Navigational Pages

As mentioned in Section 3.1, the first step in answering navigational queries is one of *identifying navigational pages*. Our strategy for identifying such pages broadly consists of two phases of analysis:

Local (page-level) analysis. In the first phase, each page is individually analyzed to extract clues that help decide whether that page is a "*candidate navigational page*".

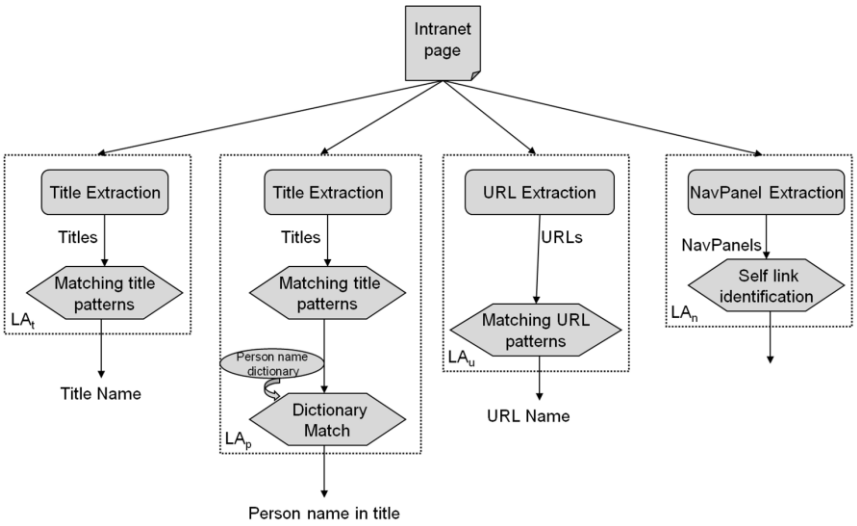


Figure 7: Local Analysis Algorithms

Global (cross-page) analysis. In the second phase, groups of candidate navigational pages are examined to weed out false positives and generate the final set of navigational pages.

We now present a generic template for local and global analysis. We instantiate this template and describe several local and global analysis algorithms. The following heuristics are motivated by observations about the naming and organization of intranet web pages. These observations, derived from extensive manual examination of the IBM intranet, appear to hold true across a range of web sites — from individual home pages to project, group, or departmental portals.

It is sufficient to restrict attention to specific attributes of a page. In general we observed that a small but specific set of attributes are sufficient indicators of a navigational page. We refer to such attributes as "*navigational features*". Examples of such features are title and URL. For instance, the presence of phrases such as "`home`", "`intranet`", or "`home page`" in the title, or an URL ending in "`index.html`" or "`home.html`", is a strong indication that the corresponding page is a candidate navigational page. (We note here that the importance of exploit-

ing URLs and titles for intranet search was also recognized by the authors of [5].)

Page hierarchies provide input groups for global analysis. While page-level cues yield candidate navigational pages, they also include a number of false positives. As an example, several pages in the IBM intranet related to "*Global Technology Services*" mention the phrase "`global technology services intranet`" in the title. However, not all of them are home pages. In fact, there is exactly one page — the "*root*" page of the global technology services web site — that is the preferred answer to any navigational query referring to global technology services. Later, we describe how candidate pages can be grouped into hierarchies based on URLs or titles and fed into the global analysis phase to identify navigational pages.

Domain dictionaries can yield significant benefits. Consistent with observation in [5] domain dictionaries such as acronyms and employee directories can dramatically improve precision. Acronyms, for example, proliferate throughout a modern enterprise as they are used to compactly name everything from job descriptions to company locations and business processes. Our experimental results (Section 3.6) show that the ability to match acronyms to their expansions significant improved the performance of navigational search. Based on query logs gathered over a period of 3 months starting June 2006 we further observed the importance of acronyms is the fact 32 of the 60 most frequent search queries on the IBM intranet were acronyms.

3.2.1 Local Analysis

Figure 6 shows the generic template of a local analysis algorithm. As shown in the figure, the first step of such an algorithm is feature extraction in which one or more navigational features are extracted from the input page. These navigational features are then fed into a sequence of pattern matching steps. Each pattern matching step either involves the use of regular expressions or an external dictionary (such as a dictionary of person names or product names). Depending on the output of the final pattern matching step, the local analysis algorithm will decide whether a given

page is a "*candidate navigational page*" and optionally associate a "*feature value*" with each output candidate.

We use the following notations in this section: Given an intranet page p and a local analysis algorithm L, let $nav_L(p)$ be a boolean value that is set to true iff p is a candidate navigational page according to L. Let $L(P) = \{p : nav_L(p)\}$ denote the set of candidate navigational pages identified by L from an input set of pages P. For each $p \in P$, let $fval_L(p)$ denote the "*feature value*" associated with p by L.

In the following, we describe four local analysis algorithms (Figure 7) that fit the template of Figure 6. Note that we use rounded rectangles to depict extraction steps and hexagons to depict pattern matching steps.

Candidate title home pages (LA_t). Algorithm LA_t is inspired by our earlier observation that page titles are excellent navigational features. LA_t extracts titles from web pages and applies a carefully crafted set of regular expression patterns to these titles. Examples of patterns that we used are given below (using the syntax of Java regular expressions [8]):

```
\A\W*(.*)'s? <Home>\b

\b<Home> of (.*)<junk>\Z

\A\W*(.*) <Home><junk>\Z

\A\W*(.*) Home<junk>\Z

\A\W*(.*) Info Page<junk>\Z
```

where

```
<Home>=(?:Home\s*Page|Intranet              (?:Site|Page))
<junk>=[\W\d]*
```

Essentially, these patterns match titles that contain phrases such as "`John Smith's home page`", "`Lenovo Intranet`", or "`Autonomic Computing Home`". When a match is detected, the corresponding page is returned as a candidate navigational page and a portion of the match is used as the feature value[3]. For our examples, the feature values are respectively "`John Smith`", "`Lenovo`", and "`Autonomic Computing`".

Note, the feature value is essentially the substring of the title that matches the portion of the regular expression within parentheses

Candidate personal home pages (LA_p). Algorithm LA_p is an extension to LA_t that focuses only on personal home pages. For each page p such that $nav_{LAt}(p)=true$, we apply an additional pattern matching step in which $fval_{LAt}(p)$ is is matched against a dictionary of person names. A successful match results in p being returned as a candidate personal home page with the name of the person as the feature value.

In our implementation, a dictionary of person names was produced by extracting names from *BluePages:* IBM's internal enterprise-wide employee directory. Note that whenever other dictionaries of entity names are avail-

Figure 8: Sample page with navigational panel

able (dictionary of product names, department names, names of company locations, etc.) a similar local analysis algorithm can be employed. The use of such dictionaries is a powerful mechanism for exploiting available organization-specific structured information. This helps to make local analysis algorithms more effective and customized to a particular intranet.

While the distinction between LA_p and LA_t (i.e., specialized processing of personal home pages) may seem artifical, it is important for reasons that will be clear when we describe term-variant generation in Section 3.4. Es-

sentially, the process of generating term-variants for person names is quite unique and does not apply to all feature values in general. By keeping personal home pages separate, the appropriate variant generation technique can be applied exclusively to those pages.

Candidate URL home pages (LA_u). Analogous to the previous algorithms that operate on the title, algorithm LA_u applies regular expression patterns on the URL of a page to identify navigational cues. Our patterns captured several heuristics, two of which we list below as examples:

1. When the URL ends in a "/" or contains "`index.html`", "`home.html`", "`logon.jsp`", etc., as the last segment, the corresponding page is returned as a candidate navigational page. The feature value is the last directory segment of the URL. For example, the URL `http://w3-03.ibm. com/services/iga/index.html` will result in the feature value "`iga`".

2. When the URL only contains a host name or a host name followed by "`index.html`", "`index.jsp`", etc., the corresponding page is considered a candidate navigational page. The associated feature value is a segment of the host name after excluding common segments such as "`www`", "`w3`", "`ibm.com`", etc. For example, the URL `http://w3.research.ibm. com/` will result in the feature value "`research`".

Candidate NavLink home pages (LA_n). Finally, algorithm LA_n exploits a particularly common characteristic of IBM intranet pages, namely, the presence of navigation panels. Such navigation aids of one form or another are quite common in most web sites, within and outside the enterprise. Typically such panels are part of a department/group-level page template and contain a list of links that point to specific pages within the web site of the corresponding department or group (see Figure 8**).** The key observations that allows us to exploit navigation panels is the following: whenever one of the links in a navigation panel on page p is associated with text such as "`home`" or "`main`" (e.g., the link labeled "`global technology services home`" in Figure 8) and if the link is a self-reference, (i.e., the link points to p), p is likely to be a navigational page.

3.2.2 Global Analysis

The local analysis algorithms presented in the previous section rely on patterns in the title or URL of a page. Given multiple pages with similar URLs/titles that match these patterns, our local analysis algorithms will recognize all of these pages as candidate navigational pages and assign

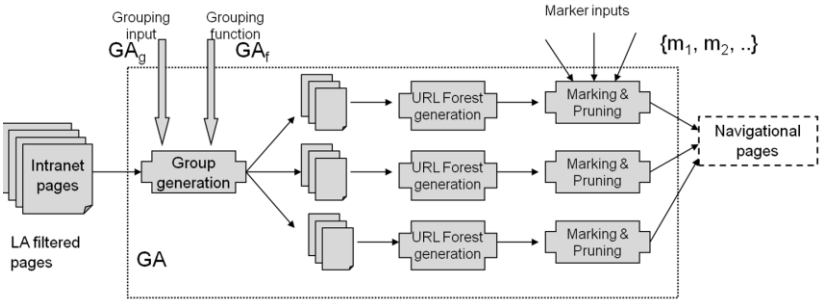


Figure 9: Site Root Analysis Algorithm

identical feature values. To illustrate, let us revisit our *Global Technology Services* example. Several pages within the Global Technology Services website contain the phrase "`Global Technology Services Intranet`" in the title. Therefore all of these pages will be identified by LA_t as candidate home pages with a feature value "`Global Technology Services`". Similarly, authors of personal home pages often use a title such as "`Howard Ho home page`" on all the pages in their website (CV page, publications page, projects page, etc.). As a result, LA_p identifies all of these pages as candidate personal home pages.

However, in both examples, the intended target of users issuing navigational keyword queries is the "*root*" or "*entry*" page. To filter out spurious pages from the output of local analysis, we propose a global analysis algorithm called "*site root analysis*".

Site root analysis. Site root analysis exploits the hierarchical structure inherent in groups of related pages to identify root pages. Figure 9 shows a generic template for this algorithm. Specific instances of this algorithm can be employed by supplying a "*grouping input*", a "*grouping function*", and

a set of one or more "*marker inputs*". In a typical instantiation, the grouping and marker inputs will be the set of candidate navigational pages identified by different local analysis algorithms. In the following, we first present the generic algorithm and then the describe specific instantiations shown in Figure 11.

For a particular instance GA, let GA_g, GA_f, and $\{GA_{m_1}, \ldots, GA_{m_n}\}$ denote the grouping input, grouping function, and marker inputs respectively. The first step in site root analysis, called "*group generation*", partitions the input pages in GA_g into groups of related pages. Precisely how this grouping is accomplished is dependent on GA_f and will be illustrated later in this section when we describe specific grouping functions.

For each group, the next step is to execute a process known as "*forest generation*". In this process, given a group S, we produce a graph G_s such that (i) there is a node in G_s for each page in S, and (ii) there is a directed edge from pages a to b iff the following condition holds: among all the URLs of pages in S, url(a) is the longest URL that is a strict prefix of url(b).

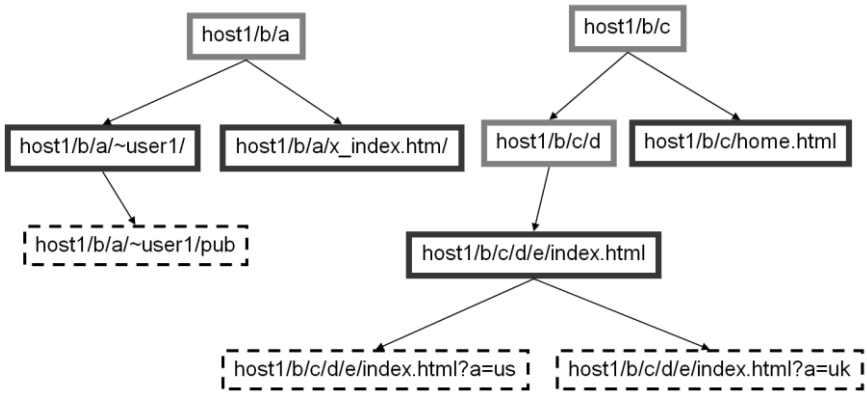


Figure 10: Site forest generation and site root marking and pruning example

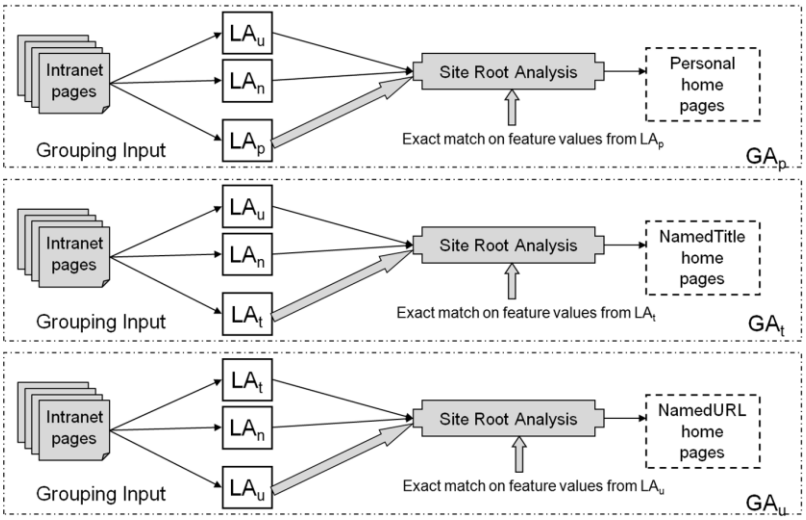


Figure 11: Specific instances of site root analysis

It is easy to see that the resulting graph G_s is a forest of rooted trees as illustrated by the example in Figure 10. When computing prefixes, we only consider complete URL segments. Thus "http://host/abc" is a prefix of "http://host/abc/def" but "http://host/abc/de" is not a prefix. In the next step, every node in the forest whose corresponding page is present in at least one of the marker inputs $G_{m_1},...,G_{m_n}$ is "*marked*". Using these marked nodes, the final output of this algorithm is generated as follows:

1. First, we add all marked nodes to the set of output navigational pages.
2. Next, from each tree in the forest, we remove all sub trees that are rooted at marked nodes (i.e., all marked nodes and their descendants are removed).
3. Finally, we add the root of each remaining tree in the forest to the output.

Figure 10 illustrates how marking and pruning works using a simple example of a forest with two trees. Let the nodes labeled 2 and 3 be marked as indicated by the circles around the node labels. The pruning step will remove the entire sub tree rooted at 3 as well as the nodes 2 and 3 resulting in the forest shown in the bottom part of the figure. Nodes 1 and 6 are the

roots of the remaining trees in the forest. Thus, the final output consists of nodes 2, 3, 1, and 6.

Instantiating site root analysis. Figure 11 shows three specific instantiations of site root analysis that we have implemented and deployed on the IBM intranet.

Algorithm GA_p is the global analysis used to identify Personal home pages. GA_p uses the output of LA_p as the grouping input and the output of LA_u (local analysis on URL) and LA_n (local analysis on navigation panel) as the marker inputs. Recall that LA_p is the local analysis that identifies candidate personal home pages and extracts person names from the title as feature values. Since GA_p uses an "*exact match*" grouping function, all candidate personal home pages with identical person names will be placed in a single group and subject to site root analysis.

Figure 11 also depicts analogous algorithms for identifying NamedTitle home pages (GA_t) and NamedURL home pages (GA_u) which differ from GA_p only in the choice of the marker and grouping inputs.

Extended site root analysis. In addition to the hierarchical structure inherent in the URLs, we noticed that a set of related pages in the IBM intranet often exhibited an implicit hierarchy in their titles. For example, the *Diversity Councils* home page is located within the website of the *Global Workforce Diversity* program which in turn is located under the main human resources website. The titles for these three pages are, respectively: "`You and IBM`", "`You and IBM | Global workforce diversity`", and "`You and IBM | Global workforce diversity | Initiatives | Diversity Councils`". Intuitively, web page authors use segments of the titles (the portions separated by "`|`") to indicate how the particular page fits within the overall organization of the web site.

To exploit this hierarchy, we developed another instance of the site root analysis algorithm as shown in Figure 12. In this case, the grouping input is the set of titles extracted from the entire corpus of intranet pages. The results of the three earlier global analysis algorithms (GA_p, GA_n, and GA_t) as well as the navigation panel local analysis algorithm (LA_n) are used as

marker inputs. The grouping function is such that any two pages whose titles are hierarchically related (e.g., all three titles listed above) are placed within the same group. We refer to the output of this global analysis algorithm as the set of rooted home pages.

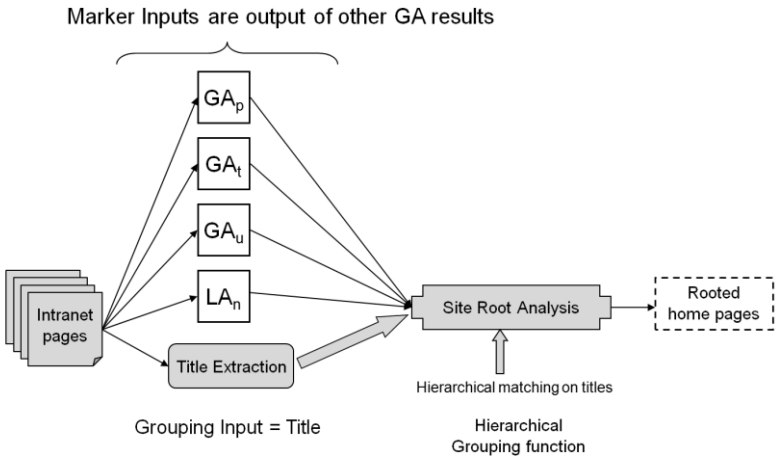


Figure 12: Site root analysis using hierarchical title grouping

3.2.3 Anchor Text Analysis

An important consideration in most implementations of Web search is the use of "*anchor text*". However, there is no single accepted approach and consequently there is a large body of work describing various implementations that exploit anchor text in different ways [5]. Our implementation to exploit anchor-text follows the two-phase strategy of local and global analysis. Following the template in Figure 6 anchor-text is treated as yet another "*navigational feature*". Note that unlike title and URL anchor-text is an external navigational feature. External because it is not obtained from the page itself; instead it is extracted from pages "hyperlinked" to the page for which it is a navigational feature. Further, we restrict the anchor-text string to the body of text within the "`<a href...>`" and "`</a>`" elements of a hyperlink. It is to this string that we apply the identical set of regular expression patterns used to identify candidate title home pages. These restrictions ensure that our exploitation of anchor-text meets the

same stringent requirements as the other local analysis algorithms. At the end of local analysis, "*hyperlinked*" pages with anchor-text strings containing phrases such as "`John Smith's home page`" and "`Lenovo Intranet`" will be retained as candidate navigational pages. Note that analogous to candidate personal home pages it is possible to restrict attention by applying an additional dictionary filter, such that, in the above examples only "`John Smith's home page`" will be identified as a candidate personal home page.

In contrast to site root analysis — where the goal was consolidation from candidate navigational pages to root pages — the purpose of global analysis in anchor-text evaluation is to pick a particular feature value to associate with the candidate navigational page. Consider for example a candidate navigational page with four anchor-text features and associated values "`John R. Smith`", "`John R. Smith`", "`John Smith`" and "`Manager`". Pointing out the obvious the first two feature values were extracted from anchor text "`John R. Smith's home page`", the third from "`John Smith's home page`" and the last from "`Manager's home page`".

Global analysis operates by first creating groups of related features using an appropriate grouping function. A canonical feature value for the group with the largest number of elements is then determined. This is the feature value to be associated with the candidate navigational page. In the event where there is no group with the largest number of elements the candidate navigational page is discarded. In the above example using an "*exact match*" grouping function "`John R. Smith`" will be the feature value associated with the navigational page.

3.3 Intranet Local Search

As we mentioned in the introduction, one of the challenges of search in a large geographically distributed and diverse intranet is that of dealing with organization and location-specific terminologies, pages, and queries. In their study of the IBM intranet, the authors of [5] report that the correct answer to a query is often specific to a site, geographic location, or an organ-

izational division, but the user often does not make this intent explicit in the query.

For example, a fairly popular query in the IBM intranet is "`bto`", an acronym that expands to Business Transformation Outsourcing. Using the analysis algorithms presented in the previous section, we were able to identify several home pages that were all associated with the feature value "`bto`" — including a BTO research home page, a BTO sales home page, and a BTO marketing home page. If a user who submits this query is primarily interested in sales figures and sales leads related to BTO customers, presenting him/her with the research home page is clearly not the best option. Ideally, if the profile of the user indicates that he/she belongs to the sales division (based on job description, job title, position in the organizational hierarchy, etc.), the sales home page for BTO must rank first in the search results.

In the same vein, there are several examples of navigational queries in the IBM intranet where the best result is a function of the geography of the user, i.e., the region or country where the user is located. For instance, there are about a dozen home pages that we identified as relevant for the query "`ip telephony`", one for each country where IBM has deployed the IP telephony program. More often than not, users who type in this query intend to reach the IP telephony home page for the country or region (Asia-Pacific, Latin America, etc.) in which they are located.

In general, given a search query with an associated user profile, our goal is to use certain attributes of the user profile, such as work location and job description, to further filter or rank the results from the navigational search index. Unlike on the Web, the task of associating a user profile with a search query is easily accomplished and poses fewer technical and privacy challenges, since this infrastructure is already available at the IBM intranet. However, two challenges remain to be addressed:

Task 1. Recognizing that a given navigational query is sensitive to a particular attribute of the user profile.

Task 2. Filtering or ranking the results from the navigational index given a particular attribute value.

In this section, we address a specific instance of the above problem, called "*intranet local search*", where the attribute of interest is the "*geographic location*" of the user. The term intranet local search is inspired by Web search services such as Google Local and Yahoo! Local. Our solution consists of the following steps:

Geo-tagging: a local analysis step in which each intranet page is individually analyzed and tagged with the names of one or more countries and regions.

Geo-sensitivity analysis: a global analysis step in which the geography tags for all the pages with a given navigational feature value are examined to conclude whether queries matching that value are geography-sensitive.

Geo-filtering: a runtime filtering step in which the results for queries that are judged to be geography-sensitive are filtered to include only the pages from the geography where the user is located.

3.3.1 Geo-Tagging

We have developed a local analysis algorithm for associating geographic tags with each page in the IBM intranet. Similar to the other analysis presented in Section 3.2, geography tagging consists of a feature extraction step followed by the application of regular expression and dictionary matches on the extracted features.

The set of features we extracted for each page consisted of the title, the URL, and the values associated with three specific HTML meta fields named "`Description`", "`Keywords`", and "`Country`". On each of these features, specially crafted regular expressions were applied to identify and extract names of countries, regions (e.g., Asia-Pacific, Americas, Eastern Europe, etc.), country codes, and region codes. To help with this process, we employed several generic as well as IBM-specific dictionaries such as a dictionary of country names, a dictionary of ISO country codes, a dictionary of geographic regions recognized within the IBM organization, and a dictionary of all known IBM sites along with their names and locations. The output of this step is a set of region and country names associated with each page in the corpus.

3.3.2 Geo-Sensitivity Analysis

Geo-sensitivity analysis is required to ensure that filtering of results only happens for queries where such filtering makes sense. For example, if we examine all the pages associated with the navigational feature value "`ip telephony`", we see that the pages are distributed across a wide range of geographies. It is therefore a reasonable assumption that "`ip telephony`" is geography-sensitive and that results for this query submitted from, say IBM Italy, and must include only the ip telephony home page for Italy or Europe. On the other hand, the query "`sam palmisano`" has nothing to do with geography and is always answered with the home page for Sam Palmisano, independent of the location from where the query is issued. For the results presented in this chapter, we employed the following simplistic metric: any query for which the results from the navigational index involved more than one country or region was presumed to be geo-sensitive.

3.3.3 Geo-Filtering

On the IBM intranet, each query is associated with a particular user whose employee profile includes the country where the user works. Therefore, given a query that is judged to be geo-sensitive as per the metric described above, the task of geo-filtering is merely one of removing all those result pages whose geography tag does not match with the geography of the corresponding user. Note that while the simple geo-sensitivity measure described above can be applied completely at query time by examining the result set from the navigational index, we expect more sophisticated geo-sensitivity analysis techniques to require offline pre-processing.

3.4 Indexing Navigational Pages

In this section, we address the task of building a "*navigational index*" that exploits the results of local and global analysis to answer navigational queries with significantly higher precision than a generic search index. There are two steps in this process: "*semantic term-variant generation*" and "*indexing*". We describe each of these steps below.

3.4.1 Semantic Term-Variation Generation

Recall that at the end of the analysis described in Section 3.2 , we are left with multiple collections of navigational pages (Personal, NamedTitle, NamedURL, etc). We refer to these collections as "*semantic buckets*".

Associated with each page in each bucket is a feature value — e.g., a person name, a phrase in the title, a segment of a URL, and so on. In the traditional indexing process, all of the terms in these features will be treated as regular text tokens and the resulting inverted index will only benefit from standard techniques such as stemming or stop word elimination. However, since each semantic bucket reflects the underlying analysis steps that was responsible for placing a particular page in that bucket, we can employ more sophisticated techniques when generating "*term-variants*" for these feature values.

Variant generator for person names. A rule based approach is ideal for navigational feature values that have very specific semantics that gets reflected in their structure. Among our examples, person names (i.e., the feature values associated with pages in the Personal bucket) fall into this category. Given a person name consisting of some subset of first name, middle name, last name, and nick name tokens, we have enumerated a set of rules for automatically generating all valid syntactic variants of that name. For instance, using our rules, we were able to generate 10 variants for the name "`Ching-Tien T. (Howard) Ho`" (e.g., "`howard ho`", "`ching-tien`", etc.).

Thus, an index that exploits these variants will be able to match a query "`ho, ching-tien`" against the feature "`Ching-Tien T. (Howard) Ho`" and return Howard Ho's home page from the navigational index.

Acronym-based variant generator. Recall our earlier observation about the prevalence of acronyms within the intranet and the fact that acronyms are a major fraction of the most common single word queries issued within IBM.

To take advantage of this observation, we developed an acronym-finder by extending the techniques described in [17]. We ran this acronym-finder over a corpus of 1.5M pages and were able to generate a dictionary containing approx. 27K pairs of unique acronyms and their expansions.

Using this dictionary, we implemented a variant generator that checks each navigational feature for the presence of an acronym or an expansion. Whenever an acronym (resp. expansion) is present in the feature, the corresponding expansion (resp. acronym) is used as a variant. Thus, the navigational feature "`Global Technology Services`" will result in a variant "`gts`" that will be indexed along with the original feature.

N-gram variant generator. Our default variant generator is a simple N-gram based approach that we use whenever the other two generators do not apply. Essentially, this generator treats all possible n-grams of a feature value as valid variants. Since typical keyword queries in the intranet are fairly short, we limit ourselves to n-grams for $n \leq 3$. Thus, the feature value "`reimbursement of travel expenses`" that we extracted from the title is associated with the following variants: "*reimbursement*", "*reimbursement travel*", "*travel expenses*", "*reimbursement travel expenses*".

3.4.2 Phrase Indexing

Once the appropriate variant generator has been applied to the feature values in each semantic bucket, the indexing process is straightforward. For each bucket, we build a corresponding inverted index in which the index terms associated with a page are derived exclusively from the navigational feature values and associated variants. None of the terms from the original text of the page are included. Thus the resulting inverted index is a pure "`navigational index`" that will provide answers only when user queries match navigational feature values or their variants.

3.5 Ranking Algorithm

After off line analysis (local, global and semantic variant generation) we are left with multiple indexes, each corresponding to one semantic bucket. During runtime, in response to a query, each index returns zero or more re-

sults and the task of the ranking algorithm is to merge these results into a single rank-ordered list. The focus of this chapter is less on rank aggregation and merging and more on the discovery of semantic buckets. However we do implement a simple ranking algorithm based on the following statistical model.

Ranking based on expected precision. Let us assume that we have a set of queries Q and their relevance judgments. For every q ∈ Q each of the indexes provides a ranked list. Thus to every index corresponds a dataset represented as a matrix where the rows are the ranks and the columns are the different queries q. The value of each cell is a pair of booleans indicating whether a result at this rank is provided for the given query and whether the answer is correct. Using this data we estimate the probability of success for every index and for every rank, as

$$P_{t,r} = \frac{C_{t,r}}{N_{t,r}}$$

where *t* subscripts the semantic bucket and *r* subscripts the rank, $C_{t,r}$ is the number of queries answered correctly at rank *r* by semantic bucket *t*, and $N_{t,r}$ is the number of queries answered at rank *r* by semantic bucket *t*.

With sufficient data, a natural ranking order for the merged document list would be this estimate $P_{r,t}$. However in practice, with limited data, the estimated probabilities are not very reliable and consequently need to be smoothed. Moreover, since the final output is a ranked list, any smoothing technique we use must account for the fact that probability of success for neighboring ranks should be highly correlated. This intuition is captured in the probability estimate calculated using the formula

$$P_{t,r} = \frac{\sum_{r-k}^{r+k} C_{t,r}}{\sum_{r-k}^{r+k} N_{t,r}}$$

where k is a parameter which defines the window around which the probability of the current rank is computed. For our experiments we used a value of k =5.

3.6 Experimental Results

In this section we describe our experiments comparing the performance of the approaches presented in this chapter (System X) against the existing system for IBM intranet search (W3). The goal of our experiments is two-fold:

Evaluation based on mean reciprocal rank. Since our goal is precision, our measurement bases on the statistical measure of the *mean reciprocal rank (MRR)*. Typically, the MRR evaluates the effectiveness of a search engine, ordered by the probability of the correctness of returned pages. The assumption of the MRR is that higher ranked pages are more likely to be correct. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q:

$$MRR(Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

Sensitivity to system parameters. Since we believe that intranets will be in a constant state of evolution, a major goal in our undertaking is to build a system that can continuously be updated with new domain knowledge for both local and global analysis. It is therefore necessary to understand the overall effect to the system as parameters change.

3.6.1 Data Set and Queries

Our data set consists of a sample of about 5.5 million web pages taken from the IBM intranet. After offline processing we were left with approximately 55,000 pages -i.e., around 5 million pages were pruned away after local and global analysis.

The queries used in evaluation were selected from the top 10000 most frequent queries in the period June through August 2006. After removing duplicates we were left with 4759 distinct queries. From these we selected a subset of 346 queries for which we were able to get expert-defined gold standard consisting of 446 pages (some queries have more than one page that is the correct answer). All 446 results are present in our subset of 5.5 million pages.

Semantic Bucket	M@50X	M@50W3	C@50X	AX
A-NamedTitle	0.3619	0.2179	15	35
A-NamedURL	0.3568	0.1800	18	37
A-Rooted	0.1264	0.1711	17	58

Figure 13: All keywords (Acronyms only)

System	M@50	C@50	S@1	S@5	S@50
System X	0.3479	179	0.2679	0.4642	0.5576
W3	0.1799	110	0.1402	0.2274	0.3427

Figure 14: Overall results using merged ranking

Semantic Bucket	M@50X	M@50W3	C@50X	AX
Personal	0.5948	0.2649	36	58
NamedTitle	0.2226	0.1459	42	153
NamedURL	0.2907	0.1626	56	125
Rooted	0.1463	0.1637	99	290

Figure 15: All keywords by semantic bucket

Semantic Bucket	M@50X	M@50W 3	C@50X	AX
A-NamedTitle	0.3841	0.1366	10	23
A-NamedURL	0.3480	0.1787	13	25
A-Rooted	0.1624	0.1645	14	40

Figure 16: One keyword queries (acronyms only)

Semantic Bucket	M@50X	M@50W 3	C@50X	AX
Personal	0.9194	0.3029	30	31
NamedTitle	0.1838	0.1431	8	34
NamedURL	0.1481	0.1111	2	9
Rooted	0.1734	0.2125	32	95

Figure 17: Two keyword queries

Semantic Bucket	M@50X	M@50W 3	C@50X	AX
Personal	1.0	0.5625	2	2
NamedTitle	0.6250	0.2040	8	12
NamedURL	-	-	-	-
Rooted	0.3493	0.1653	11	19

Figure 18: Three keyword queries

Country	Region	M@50	C@50X	AX
UK	EMEA	0.4858	99	137
DE	EMEA	0.4879	104	144
US	AMERICAS	0.3384	164	293
CA	AMERICAS	0.4433	106	161
JP	ASIA-PACIFIC	0.484	101	138
AU	ASIA-PACIFIC	0.493	100	137
GLOBAL	GLOBAL	0.489	99	136

Figure 19: Results for merged ranking and geography filtering

3.6.2 Evaluation Criteria

A query is said to be "*answered*" by a system if the result list from that system is not empty. A query is said to be "*covered*" by a system if at least one result from that system matches the gold standard. Of the 346 queries, 321 were answered by System X, i.e. System X provides at least one result for each query. Since we are evaluating the precision of System X, all our results are reported on this set of 321 queries. The following criteria are used in our evaluation of the systems.

Mean Reciprocal Rank at 50 (M@50). As stated beofore, the MRR is the average, over all queries, of the reciprocal rank (1/r) of the best rank r of the correct results. The reciprocal rank is zero for queries without correct results. For computational expediency we only calculate M@50, where correct results beyond rank 50 are not considered.

1. **Covered at 50 (C@50).** The number of queries for which one of the correct answers is ranked in the top 50 in the result list.
2. **Answered (A).** The number of queries for which at least one answer is given (i.e. the result list is nonempty).
3. **Success at 1 (S@1).** The proportion of queries for which one of the correct answers was ranked first in the result list.
4. **Success at 5 (S@5).** The proportion of queries for which one of the correct answers was ranked in the top 5 in the result list.

3.6.3 Evaluation

Overall Performance of System X. Figure 8 shows an overall comparison between System X and the existing system, W3. System X outperforms W3 on all the criteria. For example, out of the 179 queries System X answered with at least one correct result, 149 of which have at least one correct result ranked at 5 or better, giving S@5 = 149/321 = 0.4642. Note that System X provided a correct result in the top 50 hits in 56% of the time that it believed it contained the navigational page. Our data also show that rank 1 position of System X alone contains as many correct results as the first 10 positions of W3, while rank 1 and 2 positions of System X contain as many correct results as the first 48 positions of W3.

Effects of individual semantic buckets. Figure 9 shows the results for individual semantic buckets: Personal, NamedTitle, NamedURL and Rooted. The columns C@50X and AX are the number of queries covered and answered by system X, respectively. Although W3 results are not divided into the semantic buckets, we nevertheless calculate M@50W3 for comparison purposes in the following way. First we note that W3 answers every query. For each query, we take the best ranked results from System X and find the semantic bucket(s) it belongs to. The M@50W3 column is the MRR of a system that answers each query exactly in these buckets with the same result as that provided by W3.

Effects of acronym dictionaries. Figure 8 shows that bucket *Personal* outperforms the other semantic buckets. Given that bucket *Personal* was bootstrapped using an employee dictionary, we decided to isolate and measure the effect of using other dictionaries such as an acronym dictionary. Figure 10 shows the results of additional semantic buckets formed by matching acronyms against NamedTitle, Rooted and NamedURL. The MRR of A-NamedTitle and A-NamedURL for System X are 65% and 23% higher than their non-acronym counterparts. The MRR of A-Rooted is smaller than that of Rooted, but adding this semantic bucket does not decrease the overall MRR, because the global ranking algorithm (Section 3.5) ensures that results with lower expected precisions do not get ahead of results with higher expected precisions.

Effects of keyword length. The effects of semantic buckets are not uniform over queries of different length. Figure 11, 12 and 13 report the results on queries with 1, 2 and 3 keywords. It is interesting to note that NamedURL performs best on one keyword queries, Personal best for two keyword queries, while NamedTitle best for three keyword queries. The case for NamedURL is probably due to the fact that the URL names extracted contains a single path segment. The case for Personal is probably due to the fact that most personal name queries contain two keywords (first name and last name). The case for the NamedTitle is probably due to the fact that matching three keywords simultaneously against a string that is already identified as the name of a homepage is unlikely to be due to random accidents.

Figure 11 reports the results on queries with 1 keywords using acronyms. The MRR of all three semantic buckets improved over that of Figure 10, due to many one-keyword queries involving an acronym that is matched in the title.

Effects of geography. Although System X answered 321 queries, only179 queries contain correct answers in the top 50(see also Figure 8). A closer examination of those results that did not contain correct answers show that many in fact contain geography sensitive homepages those that should be considered homepages in a particular country but not globally. Performing geo-analysis on identified pages allows appropriate geography filtering based on the user geography.

Figure 19 shows the results for different user geographies. Out of the 321 queries that System X answered, 137 are pertinent to UK. Among these 137 queries, 99 are answered by System X with at least one answer pertinent to UK. The MRR of these answers is 0.4858. It is obvious that filtering the results by the user geography improves the MRR for all geography locations except US. The reason for this exception is likely due to the fact that, at least for IBM, the distinction between a GLOBAL page (i.e., a page with no particular geographical tag) and a US page is often difficult to ascertain.

Results of anchor text analysis. We also measured the performance of our implementation of the anchor text analysis algorithm described in Section 3.2.2. A comparison of the MRR values in Figure 9 and Figure 10 reveals that even our fairly stringent implementation of anchor text performs worse than our other global analyses algorithms. However, we did notice that a significant fraction of the results from AnchorHP were in geographies that our current gold standard does not cover. It is quite likely that AnchorHP in conjunction with suitable geo-filtering might result in a fairly effective semantic bucket.

3.6.4 Discussion

In general, our results appear to indicate the general benefit of "*stratification*" in constructing search engines. It allows improvements that

are valid for particular aspect or subset of the results. A general ranking algorithm allows such improvements to be reflected in the overall results. Several observations can be made on our experimental results:

1. *A priori* identification of "*navigational pages*" boosts the MRR of the system significantly, compared with traditional information retrieval techniques.
2. The separation of retrieved pages into multiple semantic buckets allowed us to produce a global ranking of the results that takes into account relative precision of various buckets.
3. Domain dictionaries have significant value in improving the precision of many semantic buckets, and in creating new semantic buckets with higher precision.
4. The relative strengths of semantic buckets are not uniform for different keyword lengths. This point to a way to further improve the overall MRR, by producing an overall rank of results based on keyword lengths in conjunction with semantic buckets.
5. The geography analysis that consists of identifying page geography, user geography, query geo-sensitivity and results geo-filtering improves the precision significantly for geo-sensitive queries.

3.7 Related Work

There are four broad areas of work that are relevant to the work presented in this chapter. This section discusses related work in each of these areas.

Understanding search goals.. The classification of search queries into navigational, transactional, and informational was originally proposed in [1]. Several examples and scenarios for each class of queries in the context of enterprise search are described in [7] and a more recent analysis of user goals in Web search is presented in [16]. There has also been prior work in the use of techniques based on classification and user behavior for automatic user goal identification [10][11][14].

Answering transactional queries in the intranet has been investigated in [15][9]. Analogous to our approach of pre-identifying and separately indexing navigational pages, the work presented in [15] describes a similar process for the class of transactional queries.

Web genre detection. Automatic web genre identification (AWGI) is being recognized as a key factor for improving the quality of search results [3][12]. For example, genre classification could allow users to sort search results according to their immediate interests [12] and could potentially improve navigational search as well.

Intranet search. Upstill et. al. [18] investigate the use of evidence such as in-degree, variants of PageRank, and URL-type, when identifying home pages on several test collections including an intranet data set. Their results indicate that of the three types of evidence investigated, re-ranking based on URL-type provided the maximum benefit. The study on "workplace web search" by [5] established that several conventional ranking approaches that find favor in Web search are not effective discriminators when applied to intranet pages. Several aspects of their study that are particularly relevant to this chapter have been mentioned in Section 3.1.

Web page search. There is a large body of work in the area of using structural information on a Web page (such as URL, anchor text, and title) to improve general Web search and link-based page classification [2][9][13]. Our work in this chapter has shown how such structural information can also be used to identify navigational pages and thereby improve navigational search.

Finally, several of our local and global analysis algorithms are inspired by techniques (such as regular expression matching and dictionary matching) that are regularly used in "*information extraction*".

3.8 Conclusion

We have addressed the problem of answering navigational queries in an intranet setting. Learning from the experiences of previous studies [5] our system is designed explicitly to address several of the conditions that make search over intranets both non-trivial and different from search on the Web. In particular, our approach pre-identifies navigational pages during off line processing, assigns pages to semantic buckets, and associates semantic term variants with each page prior to indexing. Our experiments over a corpus of 5.5 million pages from the IBM intranet demonstrated that this approach outperforms traditional WebIR ranking algorithms based on

link analysis and static ranking. The ability of our approach to leverage domain dictionaries (such as acronyms and person names) was an important factor in improving the precision of our system. A particularly important result of our investigation was the powerful effect of incorporating geo-filtering to customize the results from our navigation index to the geographical location from which search requests are received.

Despite this success there are several areas that still need to be addressed. In Section 3.5 we have presented a simple, yet natural, rank-merge algorithm. Closer examination, however, reveals several inadequacies. For instance, there is a significant improvement due to the addition of acronym dictionary (see Section 3.6). However, merging the results from all semantic buckets he MRR by a mere 3%. Clearly we can do better and we intend to investigate other rank-merge algorithms. Another area that has the potential to provide dramatic improvements, such as those provided by geo-filtering, is ranking based on organizational hierarchies and job roles. Finally, we believe there are considerable gains to be obtained by going beyond semantic-variant generation and determining variants based on statistical analysis of the feature values associated for some semantic buckets.

4 Near Duplicate Detection for the Corporate Web-Forum

Current forum search technologies lack the ability to group near duplicate threads in search results. As a result, forum users are overloaded with duplicated content in search results. Worse, they prefer to create new postings without trying to search for existing answers. To overcome these problems we identify common reasons leading to near-duplicates on a large developer forum with more than one million users. Based on our findings we design and test a new near-duplicate detection algorithm for forum threads. We report from an experimental study that our algorithm outperforms existing algorithms for general Web pages, such as home pages or product pages, significantly.

The content and ideas of this chapter are published in [132].

4.1 Introduction

Our study from Chapter 2 shows that informational query intentions are frequent in corporate setting. In this chapter we focus on resolving informational queries with the intention to receive advice or to explain**.** In corporate Web- forum users pose questions with an informational intent and other users provide answers. As the number of threads in a forum often grows drastically, a high precision search engine over forum content is necessary to grasp the mass of forum content and to spot relevant answers with high precision. However in practice, forum search engines miss the ability to identify threads with near-duplicate content and are not able to group similar threads within search results. As a result users are often frustrated by the low precision of existing forum search engines. Worse, in our test-snippet of 219,000 contributions from the forum SDN.SAP.COM we measured one day as average waiting time for a first reply (not necessarily

a helpful reply) while only 27% of posted questions receive a response by another community member.

To overcome this situation, we propose a new algorithm for grouping similar answers from existing threads in an offline process. When a user will formulate a query posting, the posting is matched against existing thread groups. As a result, users are able spot relevant answers from existing content immediately. Furthermore, users willing to post answers can immediately oversee, which new threads are already answered already and can focus on unanswered threads.

Common methods to detect duplicate Web-content are near-duplication detection approaches e.g., see [31][32]. These methods are developed for detecting machine replicated Web pages, such as Web pages that describe the same entity (e.g., a product), but present the entity through different agents (e.g., vendors). However, unlike replicated Web-pages, forum threads that represent a semantically similar question often differ in syntax, lexical content and language. In fact, in our sample we observe that existing approaches for machine-replicated Web content only recognize near-duplicate user generated content with a precision of ca. 50%. To our best knowledge, we are not aware of any other work for forum threads. Therefore we investigate a new near-duplicate detection algorithm tailored towards forum threads. Our major contributions are:

- We identify common reasons that lead to near-duplicate threads in web-forums.
- We propose a new hybrid-model for measuring the similarity among forum threads.
- We extend the hyper graph algorithm [37] for grouping near-duplicate threads.
- We report a 26% higher precision on forum content, compared to existing approaches for general Web content.

This chapter is structured as follows: In Section 4.2 we unravel common reasons for near-duplicate threads in Web-forums. In Section 4.3 give a brief overview on the process of detecting near duplicate content in a forum and in Section 4.5 we introduce a novel thread grouping algorithm.

Section 4.6 reports on our results, Section 4.7 reviews related work and Section 4.8 summarizes our work.

4.2 Near-Duplicates in Corporate Web Forums

Multiple scenarios may lead to duplicate postings in Web forums. In this section we identify common patterns during the process of thread creation and answering process that may result in near-duplicates.

4.2.1 What is a Near-Duplicate Thread?

Chowdhury et al. states in [33] *the definition of what constitutes a duplicate is unclear*. As result of our manual inspection of typical patterns leading to near-duplicate threads we will use the following intuition to capture a near-duplicate thread:

Two threads are near-duplicates: (1) If both threads share the same intention in the question post, (2) if their answer post provides a similar solution to the problem (and optionally is marked as correct by the requesting user) or (3) if both threads are linked to each other. A near-duplicate is incorrect if both threads discuss mainly different questions and the answer to one problem is not helpful for solving the other one.

Although this definition captures the human intention when detecting a duplicate it provides much room for interpreting 'topic', 'problem' and appropriate 'similarity' measure. In the following sections we will give details on how to formalize, capture and model relevant features from threads, how to compare feature representations and how to group threads.

4.2.2 Common Scenarios in Corporate Web Forums

The following analysis bases on interviews with experienced forum administrators and a user study from a data set from the SAP Developer Network (SDN). For our user study we chose 50 threads between 2007 and 2008 from the *ABAP, General channel* that contain the word *smartform*. This term is specific to the domain of this forum and results in several duplicate threads. For each thread we analyze the posting structure, thread content, and occurrences of duplicates in threads manually. Our initial research and

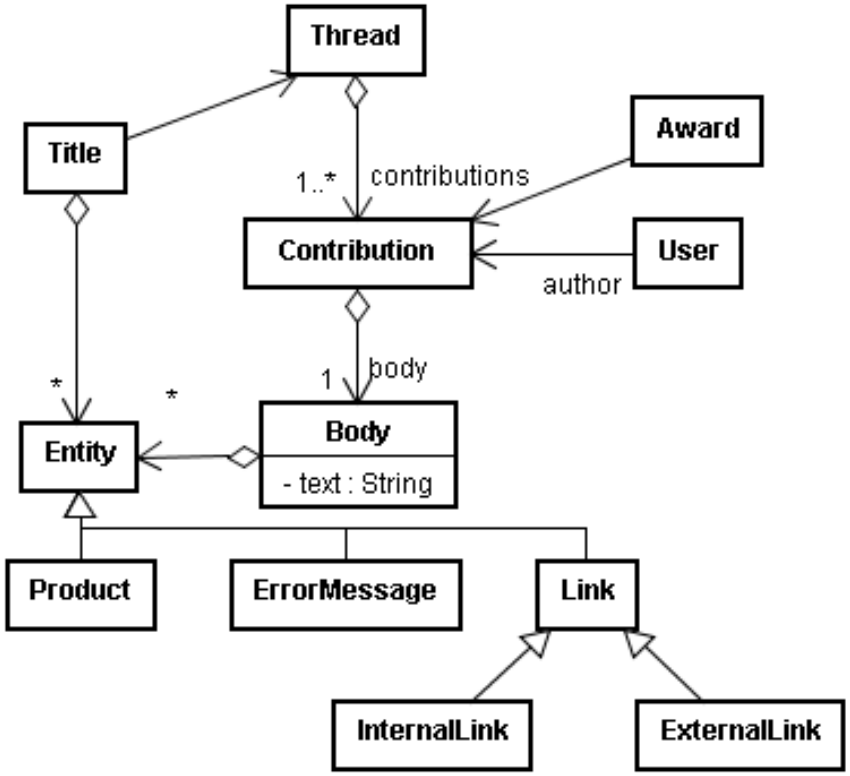


Figure 20: General thread model

discussion with a forum administrator unraveled eight common scenarios that generate duplicate content in threads:

S1: Aggressive submitter. Over time, older threads are shifted to the end of the forums overview page. To improve their questions visibility, some users copy their contributions to a new thread, which is then again presented at the top of the overview page.

S2: Inexperienced submitter. Users that are unable to decide which area (i.e. sub-forum, channel) is the right place for their topic post their contribution to multiple or even any possible channel in hope to catch the right one by chance.

S3: Too lazy to search. Often users require an immediate answer, but are unable to search existing answers. These users ‘forward‘ their request then to the community and create a new thread with their question, even if the chance is high, that their question is answered already before.

S4: Impatient submitter. Sometimes a forum engine reacts slowly. In that case some users just hit the submit button several times. Forum engines are not able to handle such an overload. As a result the engine becomes even slower and the thread is overloaded copies of the same request.

S5: Correcting mistakes too late. Some users might modify their posting, such as assigning a new title to their posting. Often these users create a corrected, new copy of their posting instead of modifying the existing thread.

S6: Thread closed too early. Some forums support the feature to mark threads as answered by the creator of the thread. Sometimes the threads creator realizes later that the existing answers in the closed thread are not helpful enough. Most forum engines do not support the feature of reopening a closed thread. Hence the thread creator copies the old request and creates a new thread.

S7: Copy/paste cheater. Most forums identify active users with a ranking system. For cheating the system, some users copy existing answers into new threads to receive a higher rank.

S8: Annoyed expert. Expert users notice duplicates and are annoyed by answering the same questions over and over again. Sometimes these users create links to existing answers instead of answering a question twice.

4.2.3 Important Reasons for Near-Duplicates

As a result from the above mentioned eight scenarios we abstract four major reasons for the existence of duplicate content in forums:

R1: Low precision of existing forum search engine answers. Posting users often search before they post to retrieve an immediate answer. However, the precision of current search engines is insufficient, e.g. results often do not match the keyword query. Groups of near-duplicates help to

shorten result lists, e.g. more results can be displayed on a single page to the searcher.

R2. Lack of transparency. Often inexperienced users are overwhelmed by the forum size and do not know where to post their question. In our case, the corporate Web Forum incorporates 60 channels (sub-forums), each channel encompasses tens of thousands of threads and each thread holds up to ten postings. Worse, with a growing Web size forum more duplicates are created. Near-duplicate detection may break this vicious circle.

R3. Information overload with new postings. In study approximately 5000 new postings are created daily. To provide room for new threads, existing threads are shifted to the end of a channel; even if these postings are not yet answered they vanish from the attention of the community. Near-duplicate detection can show a user existing threads for a question before he creates a duplicate. Thereby, near -duplicate detection may help to reduce the amount of daily created data.

R4. Slow response time. Our system did not support even the little data load of 5000 postings daily. As a result, *impatient submitters* became annoyed. Removing or preventing near-duplicates reduces the load on the forum, forum operators, forum hardware.

4.3 Overview on Detecting Near Duplicates

In this section we design our process for detecting near duplicates in threads. First, we provide a model for forum threads and then classify features that are used detection algorithm to identify duplicates. Process to identify near-duplicates in a community forum consists of the following steps:

- **Preliminary data loading.** Raw data is loaded from an external source, such as a database, a file or a data stream. Raw data incorporates unstructured forum content and may contain additional structured data, such as extracted entities, links, information about the posting author.

- **Step 1: Pre-processing**. Preprocessing cleanses raw content and prepares threads. We remove stop words; eliminate punctuation and lower case words [30].

- **Step 2: Finger print generation**. Finger print generation produces a tiny abstract data structure, called *finger print*, for each thread. Finger prints are created from text-, extracted entity-, and structure-based features. Mappings from features to threads are stored in an inverted index structure.

- **Step 3: Indexing and Grouping fingerprints.** This step identifies similar threads and combines them into a group. Each group is identified via its finger print, which is the highest common denominator its members share. New threads are inserted into a group if the finger print of the group is similar to finger print of the thread candidate.

- **Step 4: Ranking and selecting thread group representatives**. In this step we identify a representative thread that may represent an entire group in the search results. In our current implementation we select the thread from each thread group that contains most characters.

4.4 Finger Print Generation

Figure 20 shows our design model of a thread. Each thread includes additional information, such as title or date, is assigned to an author and contains textual and other features. Each contribution in a thread links to a textual body, which may include additional domain specific entities or links. We distinguish between contributions (or also called postings) of the type question and contributions of the type answers. Given a set of date-ordered contributions in a thread we call the first contribution a question contribution while we call succeeding contributions answer contributions. A finger print represents features in a compact data structure. For instance, Figure 21 shows a four dimensional vector in which each dimension corresponds to exactly one of the following four feature types:

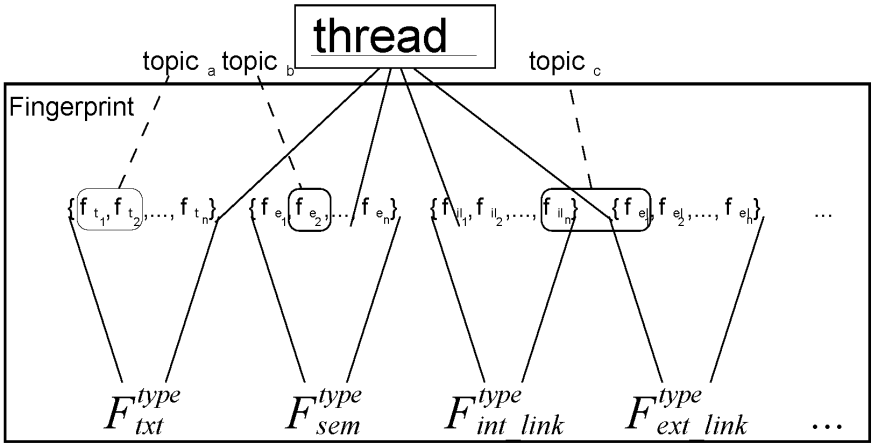


Figure 21: Example for a conceptual feature model

4.4.1 Text-based Features

We use features of this type to identify near-duplicate threads based on similar text passages, overlapping textual link representations, or overlapping representations of extracted entities:

Domain-independent text features (F^{type}_{txt}). Users in a corporate Web forum frequently utilize common terms to describe their problems, such phrases that origin from the software documentation or words that they copy from application error messages. After removing stop words and punctuation we capture these *lexical textual features* as a set of most frequent 1-grams.

Domain-independent internal links ($F^{type}_{int\_link}$). Instead of retyping an answer, users may create a link to an already answered thread *within* the same forum. The link is complemented with some textual comment, such as: *'Please check this link'* or *'For RFC Sender problems please refer to this thread'*. We call such a posting a referral posting and call the thread the posting is assigned to the answering thread. Similar to link graphs in the Web, referrals create a graph structure over threads in a forum. Our intuition is that similar threads share identical links. Therefore for each internal link we extract the reference and the anchor text description and label the reference as internal link.

Mixed model similarity	$\mathrm{Sim}_{\mathrm{Mixed}}(t,g) = \frac{S_{\mathrm{Text}}(t,g)^2 * S_{\mathrm{Struct}}(t,g)^2}{\sqrt{S_{\mathrm{Text}}(t,g)^2 + S_{\mathrm{Struct}}(t,g)^2}} * \sqrt{2}$
Similarity measure for text-based features	$S_{\mathrm{Text}}(t,g) = \begin{cases} \frac{\mathrm{Fingerprint}(t) \cap \mathrm{Fingerprint}(g)}{\mathrm{Fingerprint}(g)}, & Fingerprint(g) \neq 0 \\ 0, \text{ otherwise} \end{cases}$
Similarity measure for structure-based features	$S_{\mathrm{Struct}}(t,g) = \begin{cases} 1, & \lvert\mathrm{Fingerprint}(t) \cap \mathrm{Fingerprint}(g)\rvert \geq 1 \\ 0, & otherwise \end{cases}$

Figure 22: Similarity measurement functions for different feature types

Domain-independent external links ($F_{ext\_link}^{type}$). Sometimes posting answers in threads include links to *external* Web resources outside of the forum, such as links to files, blogs, home pages, wiki entries or other pages. Analogue, to domain-dependent internal links, for each external link we extract the reference and the anchor text description and label the reference as internal link.

Domain-dependent Semantic Features (F_{sem}^{type}). Often users refer to identical logical entities in the forum with a different lexical representation. For bridging this lexical mismatch we 'normalize' common entities with the help of information extractors [155][156][134] and assign a semantic type to a lexical representation of tokens. Our implementation focuses on (but is not limited to) the following entities:

- Products, e.g., "Exchange Infrastructure"
- Java exceptions, e.g., "java.lang.ClassNotFoundException"
- ABAP exceptions, e.g., "CX SY ZERODIVIDE"
- Referrals to software documentation, e.g., "SAP NOTE 934848"

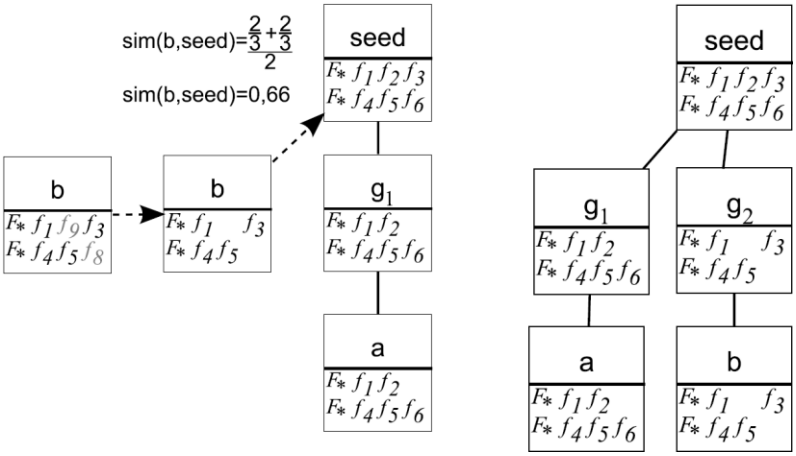


Figure 23: Example of creating a thread group

Extending these features is feasible, such as by syntactic approaches, machine based information extraction approaches and rule based approaches [28].

4.4.2 Structure-based Features

These domain-independent features incorporate posting length and thread structure.

Lengths of Thread ($F^{type}_{only\_ref}$). Our initiation for this feature is the following: The relation between a thread that contains a referral posting and the linked answering thread is much stronger if the referencing thread contains a question and a referral posting only; hence the referencing thread is two postings long. Therefore we capture the length of each thread and also capture if an internal link appears in the second posting.

Structural Patterns within Forum (F^{type}_{ch}). If two threads belong to the same channel, we expect that they are more likely near-duplicates candidates. For capturing this intuition, for each thread we label the channel as another feature.

4.4.3 Mixed Model for Fingerprint Similarity

An important functionality is to measure the similarity between the fingerprint of a thread group and another thread that may potentially belong to the thread group. A similar problem of measuring the similarity between two sets of objects is well discussed in the literature, see [30]. We describe threads with a set of structural and textual features and can apply existing metrics to our problem work. However, because of different feature types we propose the following mixed model with equation $Sim_{Mixed}(t,g)$, see also Figure 22*:* A thread t belongs likely to a thread group g if we observe for their textual similarity $S_{Text}(t,g)$ and for their structural similarity $S_{Struct}(t,g)$ equally high values. The term $\sqrt{2}$ serves as normalization factor with $0 \leq Sim(t,g) \leq 1$.

For instance, Function $S_{Text}(t,g)$ returns a high similarity value if the fingerprint of thread t shares many textual features of thread group g,, such as many named entities, links or many lexical features. If a thread t is overloaded with additional information but still contains many features of a thread group g we also consider the thread as a potential candidate for thread group g. Function $S_{Struct}(t,g)$ implements this intuition with the *Jaccard Similarity* [42] and fires a binary decision: A thread t and a thread group g are near duplicates if both share at least a single structural feature, such as they belong to the same channel or both have the same thread length.

4.5 Hyper-Graph-based Thread Grouping

Problem analysis. Assigning a near duplicate thread to an existing thread group requires generating a set of potentially overlapping groups of threads (aka subsets of the set of all available threads) where each subset may share threads with other subsets. The general problem of creating subsets of objects is well known: Authors of [10] realized that the complexity for creating a hyper-graph for duplicate content in the Web is exponential if each object is compared with any subset. However, in our scenario this worst case scenario only appears if the following two conditions become true:

1. **Each subset holds all threads.** This unlikely condition becomes true, if each thread shares at least one feature with every other thread. One option to avoid this problem is to select that are discriminative enough Figure 24to describe a small group of threads, such as links or extracted products that might appear in a thread.
2. **Fingerprint representations for all threads share on common feature.** This worst case scenario occurs if all representations (fingerprints) of all threads are different from each other. In this case each combination of two threads may create a new group.

Hence, both conditions are unlikely to become true for our thread grouping problem. Therefore we apply the algorithm proposed by authors of [10] to our problem.

Implementation. We now introduce our implementation of the hyper-graph algorithm [10] for identifying groups of near-duplicate threads. The algorithm expects two input parameters: (1) A complete fingerprint for each thread (called *seed*) and (2) a user defined parameter *th* that indicates a similarity threshold between a thread and a thread group. The output of algorithm *ComputeHyperGraph* is a set of groups that contain near-duplicate threads. The execution model of algorithm *ComputeHyperGraph* contains the following steps:

Step 1 − Create initial groups G from seeds. Generate for each thread seed a thread group $g_{seed} \in G$. Assign all features of thread *seed* to the feature space of g_{seed}.

Step 2 − Create candidates $h \in H_{seed}$ for each seed thread. Create for each group $g_{seed} \in G$ a set H_{seed} of other threads that share at *least a single feature* with g_{seed}. Hence, H_{seed} represents a set of possible near-duplicate candidates for g_{seed}.

Step 3 − Iterate over candidates $h \in H_{seed}$ and execute these steps:

- **a.** For each candidate thread $h \in H_{seed}$ compute similarity $sim(h,g_{seed})$.
- **b.** Determine fingerprint(h) and fingerprint(gseed).
- **c.** ***If 'subset thread' assign thread to group****:* If $sim(h,g_{seed}) > th$ and $fingerprint(h) \subseteq fingerprint(g_{seed})$ then g_{seed} covers all features of h. Therefore h is a member of group g_{seed} and is marked as sub-thread.

***Else if 'overlapping thread' assign thread to group**:* If $sim(h,g_i)>th$, and *fingerprint(h)* is not a subset of $fingerprint(g_{seed})$, h is inserted as member of group *g*. However, h might also be a member of other groups different from g_{seed}. Therefore, the fingerprint of *h* is compared to each remaining group $g_i \in G$, $g_i <> g_{seed}$. If similarity $sim(g_i,h) > th$, thread h is inserted into group g_i

Otherwis thread and group are disjoint: If $sim(h,g_i)<=th$, h is considered as a disjoint thread to *seed*.

Step 3 *− **Select next candidate thread h, go back to step 2.***
Step 4 *− **Select next seed thread, go back to step 2.***
Step 5 *− **Return:*** seed threads and assigned thread groups

Example. Consider a thread *b*, a seed thread *seed* and a threshold *th =0.5*.

Figure 23 presents the fingerprint of the *seed* for a thread group g_1 which contains features f_1 .. f_6. The seed forms a group g_1 with a thread *a*. Thread *b* has a finger print of f_1,f_9,f_3,f_4,f_5,f_8 and a similarity *sim(seed, b) = 0.66*. Since this similarity exceeds parameter *th* and *fingerprint(b)* is not a subset of fingerprint(g_1), thread *b* is assigned to a new group g_2. $fingerprint(g_2)$ is equal to intersection of features from *fingerprint(seed)* and *fingerprint(b)*.

4.6 Experimental Results

In this section we describe our data set, our evaluation methodology, our experiments and evaluate different parameter settings for detecting thread groups.

4.6.1 Data Set and Methodology

Data set. SAP AG runs one of the largest developer communities with more than one million registered users. Our experiments are based on a sample of 54570 threads and overall 219.000 postings that we collected from the SDN forum (see http://sdn.sap.com) of the SAP AG between 2007 and 2008. From this set we chose 2587 threads (ca. 5%), that were created in the timeframe of one week, as our Gold-Standard. Among these threads we manually identified 437 threads which featured only a question,

while 2150 threads provided a question and at least a single response (Figure 25).

Evaluation measurements. It is not possible to determine all near-duplicate pairs for several thousands of threads by hand. Therefore we cannot measure directly the recall of found thread groups. Following authors of [36], we compare our algorithms based on two metrics: (1) precision and (2) the number of duplicate threads each method was able to identify.

$$Precision = \frac{|Detected\ thread\ groups\ \cap\ Thread\ groups\ \ in\ gold\ standard\ |}{|Thread\ groups\ \ in\ gold\ standard\ |}$$

Measuring the precision requires human evaluation: After we run an experiment with one of our parameter settings, we evaluated resulting thread groups by two independent experts, who labeled detected thread groups as either correct or incorrect.

Parameter settings. We varied set of features types and the similarity thresholds of our algorithm from Section 4.4 and Section 4.5.

Baseline: This approach identifies exact duplicate threads only. A thread is considered an exact duplicate if two threads candidate threads share exactly the same initial posting. This approach serves as base line in our experiments.

Link: This approach simulates the assumption that equal questions are solved by referring to the same external resources. Thereby, the approach identifies two threads as near-duplicates if they share at least two common external or internal links.

Text-75 and *Text-80:* We conducted two experiments with text-only features on question postings. For these approaches we expect similar results existing near-duplicate algorithms for machine replicated Web-text, such as approaches presented in [31][37]. We conducted an experiment with a similarity threshold $th=0.75$ and $th=0.8$ and called these experiments "Text-75" and "Text-80".

Exp.	#Detected thread groups	Group size (in Threads)			Precision in (%)	Overall size of grouped threads
		#2	#3..5	#>5		
Exact-Duplicates	18	18	0	0	100	36
Link	5	5	0	0	20.0	10
Text-75	**139**	96	29	14	48.8	**272**
Text-80	88	76	9	3	57.8	186
ALL-Thread-1	10	10	0	0	**80.0**	19
ALL-Thread-2	5	5	0	0	**80.0**	10
ALL-Thread-N	128	118	8	0	74.3	194

Figure 24: Postings per thread before (left) and after detecting near duplicate threads (right)

ALL-Thread-1, ALL-Thread-2, ALL-Thread-N: Finally we conducted three experiments in which we considered all features. Most groups only contain a single or two postings. For instance groups which contain threads with one posting only have little value for the searcher and should be filtered out by the forum search engine. Therefore, in experiment *ALL-Thread-1* we investigate if our algorithm can detect these groups. Next, we examined groups that contain threads with two postings only in experiment *ALL-Thread-2*. Such threads often include a question and an obvious answer or a link to an answer. These threads are frequent in the data set and of high value for the answer-seeking user. Finally, in experiment *All-Thread-N*, we examined groups that contain any thread and without considering the posting length.

4.6.2 Evaluation

Figure 24 provides an overview of the results. We explain each of our seven experiments in detail now.

Exact Duplicate. In this experiment we identified 18 exact duplicate thread groups. Each group contains of two threads with exact matching question postings. A closer inspection revealed that 12 thread groups contain threads that are created by the same user. These threads may potentially result from aggressive or inexperienced submitters that generate many exact duplicates and 'spam' the forum (see Section 4.2).

Text-Only. We observe a significant drop of the precision from 58% for *Text-80* to 48% for *Text-75*. Our closer inspection of thread groups revealed that text-only features perform well for short, concise question postings that follow simple syntactic structures. Obviously, a text-based similarity approach do does not perform well for longer question postings, such as postings which encompass complex syntactic structure. Another important source of errors are short postings that contain multiple informational intention, such as a [*How to*] or [*What is*] query posting (see also Section 2.3.2). For instance, consider the posting that contains the tokens '*What is a smartform?'*. Several threads with multiple answer postings lexically matched this informational question. However, we observed that very few threads actually had the same intention; here to locate a wikipage or a thread explaining the term *smartform*. Text similarity measurements perform well on short postings, such as postings in which users often re-used old postings by changing a few words to create a new thread.

External-Links. In another experiment we studied the assumption that equal questions share exactly the same links. This feature alone performed poor. Only 1.4% of the threads from our test data set where grouped as near duplicates, while the precision was as low as 20%. We identified as major reason for the low precision the frequent usage of links to general purpose pages, such as *https://forums.sdn.sap.com*.

All Features. Our experiments *All-Thread-1* and *All-Thread-2* resulted in a precision of 80% each. However, the setting *All-Thread-1* could only detect 10 thread groups and *All-Thread-2* detected 5 thread groups only. To reach a higher recall in our next experiment we removed the condition on grouping only threads of a specific length. In experiment *All-Thread-N* threads of any posting length were combined into thread groups. In addition all features are utilized. In this experiment we detected 126 thread

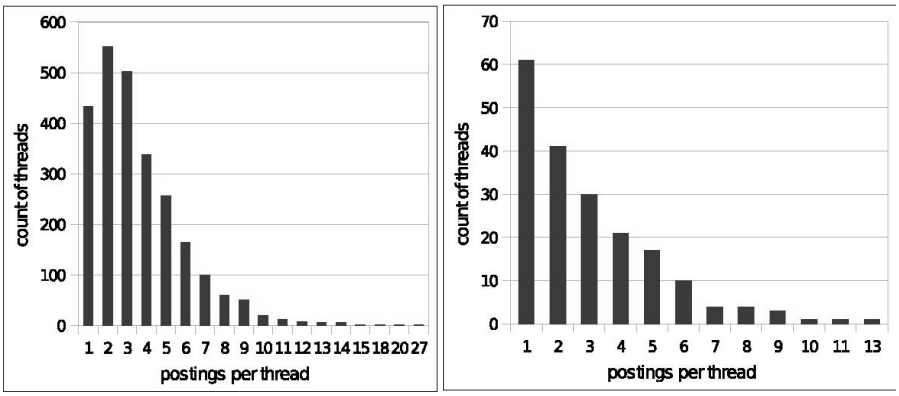


Figure 25: Average distribution of postings per thread before (left) and after near duplicate detection (right)

groups with a small drop in precision only to 74.33%. Recapture, that this experiment does consider semantic concepts, such as extracted products, error messages etc. semantic concepts generalize lexical information into more general semantic concepts. Hence this feature type can significantly enhance the size of detected threads.

4.6.3 Additional observations

Simple metrics, such as exact-match or text-only similarity matches are insufficient and do only produce decent results. We summarize our observation as follows:

Algorithm works especially well for shorter threads. Figure 25 shows the distribution of threads for varying postings lengths. Short threads are more common: The average length of a thread is between four and five postings; however most common are threads with a length of two, three and one posting. Our text-and structure-based metrics work especially well for identifying near-duplicates on these shorter threads. We assume that these threads have a clear intention in the question, while longer threads are often disturbed by additional postings that do not match the topic.

Few exact duplicates. Algorithm *All-Thread-N* detected 194 near-duplicate threads in our sample of 2587 threads. However, only among 36 threads we spotted with an exact duplicate matching approach. Hence, forum content is created by human beings and incorporates a high variety of

different expressions for a semantically identical question or answer posting.

High thresholds produce fewer results with better precision. Increasing the threshold parameter also increases the precision of found duplicates, but also decreases the count of found duplicates.

4.7 Related Work

Our work is relates to duplicate detection algorithms clustering techniques for mostly text-based Web content. In this section we compare our approach against these approaches.

Duplicate detection for Web content. Authors of [31] propose to use text-based features with an algorithm for creating multiple finger prints (called shingles) to represent a page in the Web. Each finger print may contain letters, words, or sentences. Finally, a random subset from all shingles of a page is selected and a hash function is applied. The *I-Match algorithm* [33] uses lexical and statistic information for fingerprint creation as well. First, the algorithm inspects the collection of words for every document and applies different filtering techniques, such as measuring the inverted document frequency value of each document. Next, the algorithm creates a hash finger print, which contains filtered words only, for each document and inserts document finger prints into a hash map. All documents with equal hash values are considered near-duplicates. Authors of [37] and [39] extend these ideas and propose methods to choose shingles based on important domain-specific lexical information. In their work they propose statistical approaches for identifying discriminative words and phrases from Web pages. We extend these ideas and include semantic and structural feature types, which are typical for enterprise forums. For instance, besides lexical information we incorporate additional features from fact information extraction techniques, such as [28], for generalizing lexical content, and consider internal forum link structures and external outgoing links.

Measuring similarity. Authors of [32][40][35] compare the similarity of k-bit finger prints with the Hamming Distance. If finger prints of two documents differ in at most f-Bit positions, two documents are considered to

be similar. Authors of [37][31][33] measure the similarity of a bag of words based on set similarity for instance, with the Jaccard distance. Web forums require multiple to incorporate techniques for different feature types. Therefore we propose a mixed model that includes set similarity measurements and other techniques.

Improving runtime performance. Most related to this work is the work presented by authors of [34]. Their method merges pre-selected, similar subsets of clusters in a cluster collection. Our solution is somehow similar; since we create multiple graphs for each feature type and then prune the graph to cluster that have a high combined threshold from all graphs. In contrast to the method presented in [34], our approach identifies mostly near-duplicates that share the same ancestor document and therefore the method is only able to identify if one document is a modified copy of another one. Unfortunately, this is in most cases not true for threads; rather in threads two users create a new thread for the same question independently. Therefore the semantics of these threads capture a identical topic, however the syntax and lexical expression mostly differs. Our approach captures these differences with additional features and similarity measurements.

4.8 Conclusion

The process of the creation of duplicate content in the general Web and in an enterprise forum is different. In an intensive user study, we identified both technical factors, such as low precision search engines, and human factors, such as aggressive or lazy users, as major reasons for the existence of duplicate postings in enterprise forums. As a result, in corporate forums semantically similar threads might exist, however, these threads differ in their syntactic and lexical representation. As a result of our study we developed a feature model incorporating text-based features, features based on extracted entities for products, and structure-based features. In addition we designed a mixed model computes and compare finger prints that are derived from these features. Our experimental study on several thousand threads of a real-world forum confirms that our algorithm detects near duplicate threads with high precision in corporate forums.

Part II

Ad-hoc Web Research

5 Situational Business Intelligence

Traditional business intelligence has focused on creating multi-dimensional models and data warehouses, where after high modeling and creation cost structurally similar queries are processed on a regular basis. Contrary, so called "ad-hoc" queries aggregate data from one or several dimensional models, but fail to incorporate other external information that is not considered in the pre-defined data model.

In the second part of this book we focus on a different kind of business intelligence, which spontaneously correlates data from a company's data warehouse with text data that is derived in an ad-hoc fashion from the internet. Such situational applications are usually short-lived programs created for a small group of users with a specific business need. We will showcase the state-of-the-art for situational applications. We will also present challenges that we address in this second part of the book to arrive at a platform for situational business intelligence over text data from the Web.

The content and ideas of this chapter are published in [133][136][137].

5.1 Introduction

The long tail phenomenon is often observed with the popularity of consumer goods, web pages or tags, used in “Flickr” or “MySpace”. Interestingly, current enterprise applications are characterized by a long tail distribution as well: A small number of business critical applications are maintained by the IT-department. Such applications typically require high availability, high scalability and are requested by a large number of users. Examples for such business critical systems are mostly systems that manage business transactions e.g., accounting, *customer relationship management (CRM)*, *enterprise resource planning (ERP)* and simple *online analytical processing application.* Besides of these business critical applications a “long tail” of *situational applications* exists. These are cre-

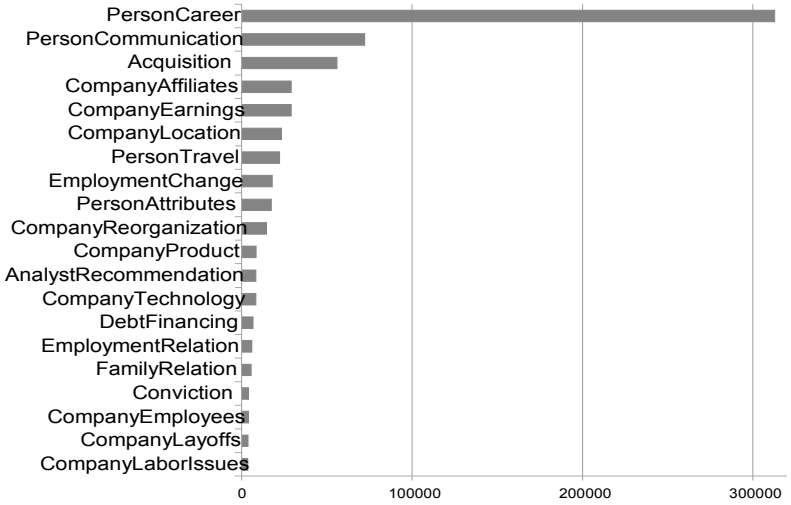


Figure 26: Distribution of 20 different fact types in the Reuters News corpus. The X-axis depicts the number of documents that contain an instance of the fact type. The whole corpus consists of 731,752 labeled documents.

ated to solve a short term problem and are often developed ad-hoc and independent from the IT-department. However, the growing amount of unstructured text in the Web, intranets or user-generated content in blogs or reviews [38] needs to be integrated with structured information from a local ware house in an ad-hoc application.

Neither conventional search engines nor conventional BI tools address this problem, which lies at the intersection of their capabilities. We envision that situational business intelligence applications will tackle this problem. They tap into the wealth of text data in order to determine new trends and give an enterprise a competitive advantage. In the second part of this book we focus on a major building block of situational business intelligence applications called information extraction algorithm, which identifies relevant concepts and relationships within and between documents.

In situational business intelligence, the value of information decreases over time. Hence, the time for *a-priori* building semantic indexes may prevent BI users from getting a fast answer. Therefore another building block is a robust data layer for retrieving, analyzing and building such index strucures. This technology enables an information worker to analyze and

extract information *ad-hoc and at query time* from large corpora, such as a sample of several 10.000s documents from the web returned by a search web engine.

The rest of this chapter is structured as follows: In Section 5.2 we propose a typical use cases for a situational business intelligence applications and identify major requirements. Section 5.3 shows the core building blocks of the system for running situational business intelligence applications and Section 5.3.2 outlines research challenges for handling text data which we will solve in the remaining chapters of this book.

5.2 Situational Business Intelligence on Text Data

The next generation of business intelligence applications requires analyzing and combining large data sets of both structured and unstructured data. According to current estimates, professional content providers (webmasters, journalists, technical writers, etc.) produce about 2GB of textual content per day, whereas all user-generated content on the web amounts to 8-10 GB per day [38]. In this section we introduce our example scenarios and to our query answering process.

5.2.1 Example Scenario

Companies more and more tap into the analysis of information about customers or competitors on company driven forums or external blogging sites. Incentives for analyzing the user-generated content are to increase the knowledge about potential thread from competitors and to minilize own risks. Often such analysis activities are initiated by non-IT people, e.g., product or marketing managers. The following use case tackles a set of ad-hoc analysis question of a business man in the aerospace industry:

Which companies collaborate with Boeing? Are these organizations also collaborating with Airbus or Fokker? Do employees of these companies have a criminal record? Who published the information?

Let's envision the following process of a human who tries to answer these questions with a set of individual informational lookup queries on a Web search engine:

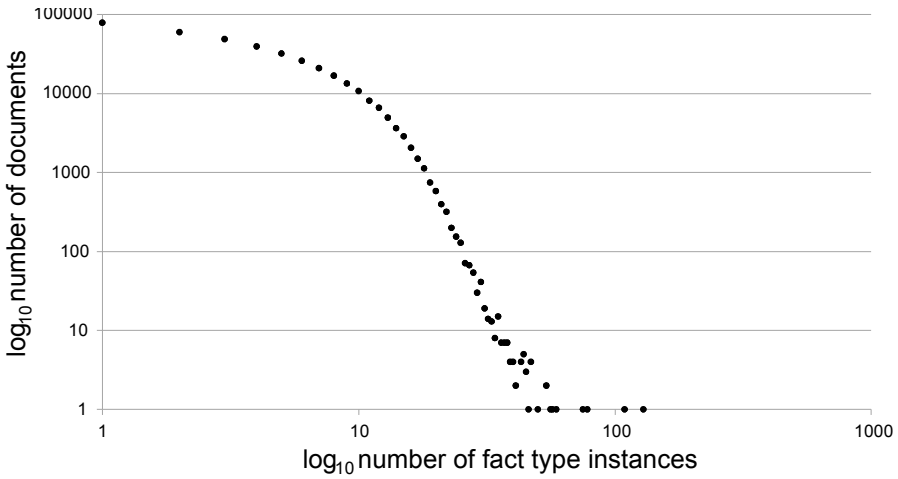


Figure 27: The figure shows a powerlaw distribution of extractable facts over documents from the Reuters News Corpus: Few documents contain more than 10 facts while the majority of documents contain very few facts.

Select an inital set of authorative data source: Most Web search engines return a set of ranked pages, where the search engine determines the authoritativenss of each site. In this scenario we assume that the human would incorporate her domain knowledge about potential sources into the search query. Therefore, the human selects a set of 'trusted news magazines', like Bloomberg, cnn.com or ft.com.

Formulate selective lexical predicates (keywords) for executing a ranked index scan against the interface of a Web search engine: Next, the human would 'generate' (in his mind) selective keywords, such as [boing], [fokker], [airbus] for the query from the introduction. The Web search engine will return a set of canidate results and the human would eventually refine these keywords to [boeing collaboration] or [engineer boeing] to fine tune results. Finally, the human would receive candidate pages from a Web search engine.

Rerank retrieved documents, filter out irrelevant documents: Not all of the top ranked candidate pages likely include factual information about the complex query demand of the human. Therefore the human would select some of the highest ranked pages (in the hope that these pages also

contain dense factual information) and would filter out pages which are likely spam or likely do not contain any factual information that is required for answering the query.

'Extract' factual information from text: From the top-ranked remaining pages, the human system would 'extract' facts, for instance into an excel sheet or a piece of paper by reading pages.

Forumulate additional sub-queries for retrieving missing values: Next, the human would detect missing information in facts and would 'generate' (in her mind) additional sub-queries for each row that misses a value.The human would execute these queries, eventually against Web sites that complement the sites from the initial step, for retrieving documents that likely contain the missing values.

Verify and cross validate information: Next, the human would cross-validate retrieved facts gainst a list of already acquired data from an in-house RDBMS. Finally, for previusloy unkown facts, the human will trace back manually the textual source.

5.2.2 Understanding Fact Distributions in a News Corpus

An interesting challenge is building a SBI system, which assists a human in formulating and answering the complex queries from the last section. Before we describe major components of such a system we analyze typical characteristics of fact distributions in news data.

Reuters news corpus as data set. We utilize the Reuters Corpus Volume I (RCV1). This standard evaluation corpus consists of 810, 000 Reuters, English Language News stories documents from August 1996 to August 1997. Since such a large corpus is impossible to label manually, we utilized the OpenCalais Information Extractor [20] to label entities and facts in the documents. Overall, this extractor extracted 866,684 entities and 1,798,875 facts out of 70 different fact types and labeled 731,752 documents in total. For our evaluation we selected 20 fact types from the domains *person*, *product* and *company*. The attributes of these fact types vary in both their number and type.

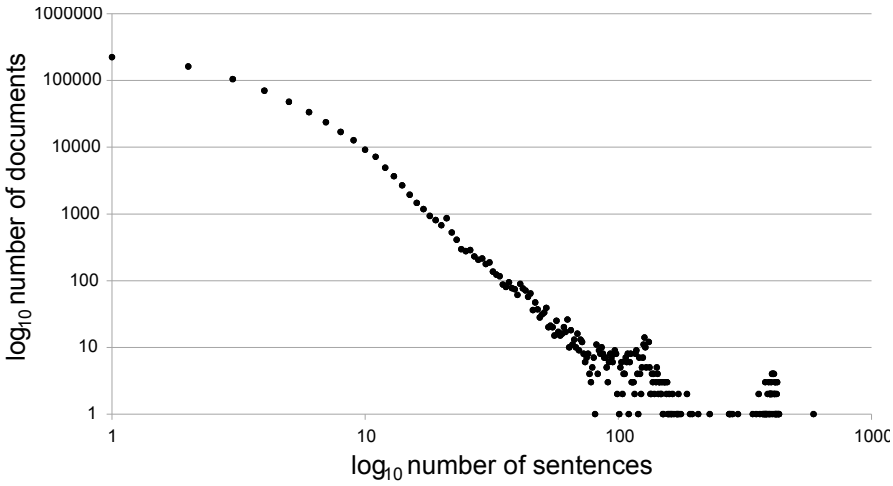


Figure 28: The figure shows the distribution of sentences over documents in the Reuters News Corpus. Many documents contain 10 or less sentences, while vey few documents contain more than hundred sentences.

Distribution of different fact types. In Figure 26 we list the number of documents that contain individual fact types in the Reuters Corpus. Most fact types are quite rare and only appear in a small fraction of the documents. For example, while the fact type *PersonCareer* still appears in about 300, 000 documents, instances of other fact types, such as *DebtFinancing* or *CompanyLayoffs* are only contained in a tiny fraction of the corpus.

Distribution of facts within documents. Figure 27 shows another insightful statistic: We plot the number of documents that contain a certain number of facts (on a double-log scale). While the overwhelming majority of documents only contain one or a couple of fact instances, just a very tiny set of documents contain many facts. Merely eleven documents in the corpus contain 50 or more fact instances. Of course it would be most desirable to use exactly those documents to identify and extract discriminative keywords.

We also investigated a potential connection between the number of facts in a document and the number of sentences it contains; we also plot the number of sentences in a document Figure 28 which also appear to approxi-

mately follow a power law distribution just like the number of fact instances inside a document. An estimated correlation coefficient of about 0.3 between the number of sentences and the number of facts in a document supports our hypothesis.

Missing attribute values in higher order facts. Our work in [123] shows the majority of facts have between two or three or more attributes. However, our research also shows that current fact extraction technologies often return incomplete facts that miss important attribute values. These missing attribute values occur because the information is represented intrinsically in the text, but the implemented extraction mechanics do not grasp the structure and/or semantics in which the information is represented by the author of the text. Another reason is that the information does not exist in the text. Well known facts are not mentioned in the text, e.g., the fact, that "New York" is a city in the US is often assumed and not explicitly stated by the authors.

5.2.3 Do you own a fresh copy of the Web?

Another interesting and relevant question is data ownership. Currently, very few, mostly US-based, companies hold an oligopoly on a fresh copy of the Web of the western hemisphere, such as Google, Bing/Microsoft (former Yahoo technology) and Alexa/Amazon.com. Other companies hold parts of the social web, such as linkedin.com or facebook.com. Finally, baido.cn, alibaba.com and yandex.ru serve Asian and Russian markets. Hence, services for running complex queries need to adapt to the interfaces and need to optimize the query demand against content access restrictions of these search engines. Moreover, financial costs will become an important factor, since Google and Bing started a pay-as-you-go api-model on their search services.

A company X that likes to investigate powerful new search services has to decide between two broad strategies: The first strategy is to let specialzed services, such as Google and Bing, own and maintain a fresh and indexed copy of the Web. In that case, the company X can investigate niche algorithms that these data owners will likely not support in the future. The company X will then execute queries through the api-interface of the Web search engine provider. Thereby, most index scans from the query are exe-

cuted on the remote Web index, while the integration logic happens at the client site of company X, where the query is also formulated.

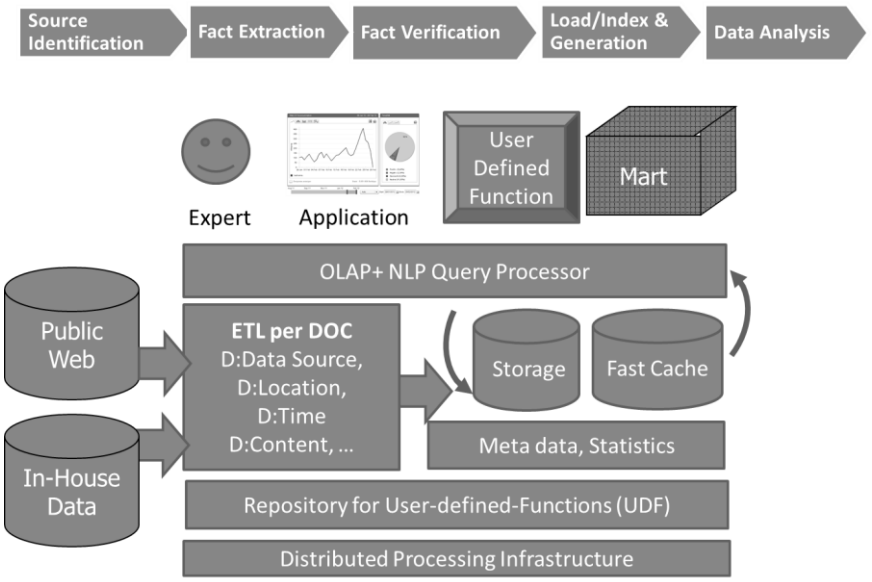


Figure 29: The figure shows a high level view on a situational business intelligence system. The upper part illustrates the basic process that follows our example query from the introduction. We envision that queries are posed by human experts. Applications, or a logic that is hidden in a user-defined function, can also trigger queries. These queries are executed on both, Web data and in-house data. This data is homogenized, persisted and indexed. The query processor utilizes meta-data and statistics, such as data distributions observed from the behavior of user defined functions, for optimizing the query execution process.

The second strategy is that company X obtains a fresh copy of the Web and maintains this copy in an own index. Because of the sheer endless amount of available data, the complexity of the index and the query processor, very few organizations will be able to execute this strategy. On the other hand, in this strategy data, the query processor and the client application exchange messages without the access limitations of the first strategy.

5.3 The Query Processor and Index of an SBI System

In this section we show a draft of the architecture of a situational business intelligence system. We present recent technologies for query processing and indexing at the scale of the Web and in an interactive, ad-hoc fashion.

5.3.1 Interactive Ad-hoc Queries

A business intelligence system needs to analyze large amounts of data in an ad-hoc fashion. Large amount of data result from a Web search or a Web crawl, where in a worst case millions of web search results or hundreds of GB of user-generated content need to be analyzed. To worsen the situation a business intelligence system must be able to answer an ad-hoc query usually in a few seconds to a few minutes.

Processing a large of data in a short amount of time is computationally intensive. The architecture of such a system is fundamentally different from a client/server architecture and storage subsystems, which traditional database systems utilize.

Distributed data storage: A distributed file system has to take the role of shared memory and storage system. Cached results and metadata have to be stored and indexed on the distributed file system of the cloud. Google's GFS [54] is optimized towards provide performance and fault tolerance. Hadoop[1], the open source MapReduce framework of the Apache Project, combines a distributed file system (HDFS) and a MapReduce runtime environment. However, current file systems do not provide random data access beyond file level. Key-Value based data stores, like Google's BigTable [47] or the open source product HBASE, overcome this limitation. These sytem use the principle of query and storage load optimizations from distributed hash tables. Thereby these systems permit simple lookup queries for a particular item, such as a URL. Key-value stores organize data in a schema free fashion which allows for simple extensions of storing schemata and sparse data, a feature which is crucial for ad-hoc text extraction tasks where often no schema is a-priori provided.

---

[1] http://hadoop.apache.org (last visted 12.08.2011)

Parallel execution engine based on the map/reduce paradigm: Due to the large number of nodes, Cloud Computing is especially applicable for parallel computing task working on elementary operations, such as *map* and *reduce*. MapReduce programs are executed in a runtime environment that is responsible for the distribution of computing tasks. These environments handle errors transparently and ensure the scalability of the system. The work proposed in [50] and [65] provide details on the MapReduce programming paradigm and its applications.

Higher level query abstration and query processor: Recently, data processing languages have been designed to reduce the efforts to implement data analysis applications on top of a distributed system. These languages are often tailored towards mass data analysis on flat data models, such as found at query log file analysis. Examples for this type of query language are PigLatin [58], Scope [45] or the Cascading framework[2]. The language JAQL[3] supports a semi-structured data model and provides basic data analysis operations. On cloud architectures, these queries need to be translated into functional data flow programs. These batch programs store crawled data into a key-value store, apply linguistic processing of a set of text documents or apply ata analysis operations, such as value aggregation.

Columnar, compressed, nested indizies for aggregation queries: Neither a distributed file system nor a key/value store (like HBASE or BigTable) is interactive enough for processing aggregation queries, which are particluary common for situational business applications. One reason is that, because of the additional complexity for ensuring data availability, the processing latency in HDFS is too slow. Worse, the storage model of a ke/value store does not support aggregation operations, which go beyond simple set and get methods of a key/value stores. The sytem Dremel [122] overcomes this limitation. First, the sytem provides a nested data model. Many datasets that orign from a Web crawl and text mining operation have 1000s of fields deep nested structures. Normalization these data sets are

[2] http://www.cascading.org (last visted 12.08.2011)

[3] http://jaql.org (last visted 12.08.2011)

prohibitive – in fact, reading a whole record is impractical, since each record may contain multiple KB of data. Dremel is optimized towards select project and aggregation queries and uses a columnar representation of nested data. Another novel aspect of Dremel is the 'serving tree architecture', which is particularly well-suited for answering aggregation queries.

5.3.2 Optimized Execution of UDFs for Text Mining

A particular important and interesting problem in situational business intelligence systems is the optimizated execution of user defined functions (UDFs), such as UDFs for text mining. These functions take as input a text, apply severval complex transforrmations and return relational data. Optimizing such UDFs is particluary difficult for many reasons:

UDFs for text mining are a niche product and have a low selectivity. Because of the potentially sheer endless variety of the human language, most text mining UDF vendors focus on a specific vertical domain, such as extracting information about companies or products. Therefore UDFs are often highy domain-specific and are often tuned towards a specific class of text sources, such as news text. Figure 26 shows the example distribution of factal information in the Reuters News Corpora. We observe that many fact types are extremely rare and only appear in less then a promille of the entire document collection. Moreover, we observe in Figure 27 that only very few documents are particluary fact rich; often a document only contains few facts that a particular UDF can recognize. We therefore assume a low selectivity for UDFs that extract these rare facts. This low selectivity prohibits full table scans over large document collections. Rather, this low selectivity would require focused index scans for retrieving documents that likely return factual information of a specific type.

UDFs for text mining enscapulate and hide their mechanics. Unfortunately, most UDF vendors hide and enscapualte the mechanics of their text mining UDFs. Economic reasons motivate vendors to hide these mechanics. One reason is to avoid product substitation by the buyer of the UDF. If the buyer could compare in an automated fashion the behavior of two UDFs, the buyer would likely select the UDF that returns results with the highest accuracy and that can be bought for the lowest price. Another reason for UDF vendors to hide their mechanics is their fear of unauthorized

copying of the UDF mechanics. Training a UDF for text mining is a labor intensive process and the vendor is interested in a fair compensation for this work.

Mismatch between data disributions in text and mechanics of a UDF. The enscapulation and hiding of UDF mechanics does not allow the buyer to match the UDF with the right text data. Rather, this insufficient knowledge may cause the buyer to potentially select a mismatching UDF for processing a specific document corpus. As a result of this mismatch, the relational output data from the UDF is likely inaccurate and/or incomplete.

A potential solution to these problems is to obsvere the behavior of the text mining UDF. One example is to observe discriminative lexico-syntactic patterns in documents for that the mechanics of UDF indicate factual information. We can use these patterns for retrieving documents from text-index that likely match the mechanics of the UDF. Since each UDF is trained on lexical and syntactic text data, this index should contain at least common lexical and syntactic tokens as index key that point to documents. In Section 6 and Section 7 we introduce approaches that implement this strategy.

Estimate the join order for incomplete data from the Web. Because of the incomplete nature of data on the Web, text mining UDFs may also return incomplete tuples. Worse, missing key attributes in tuples may hinder the execution of a join operation between two tuples. Therefore a query processor must estimate how likely the missing key attribute ccan be retrieved by processing additional documents. In Section 8 we present a solid theoretical model and powerful, adaptive query planning techniques to iteratively complete these tuples.

Text data, UDFs and index may reside in different legal organizations. Often, the text data and the index and the UDF reside in different legal organizations. Most UDFs are developed by many niche vendors, while few companies hold a monopole for hosting a large web-based text index. Creating a niche UDF requires domain knowledge. Therefore the set of the niche vendor is disjucnt with the set of companies that host aWeb index. As a result, text data is shipped to a server that hosts the index API to build

the index and is also shipped to another server that offers an API for the text mining UDF. Because of the nature of text data, shipping text data is partucluary costly. Moreover, a text mining vendor may artificially limit the user's access to the UDFs via the application programmer interfaces. Therefore another important problem is avoiding data shipping, such as by filtering out documents that a UDF likely cannot process. In Section 9 we present our approach for such a document filtering technique.

5.4 Related Work

The internet is a source of lots of valuable information. There have been several attempts to leverage this data. The Cimple project [51] provides a software platform which aims to integrate all information about specific user communities of the internet. The system starts with a high quality seed, consisting of initial data sources, relations and expert domain knowledge. From then on, it crawls it's given web sources, extracts valuable information, integrates it and provides a web front end for browsing the joined information. Similar to Situational Business Intelligence systems, the Cimple platform autonomously crawls the Web, extracts and integrates information. It differs from the concept of Situational Business Intelligence in that it is rather collecting information than allowing for extensive analyses. The Avatar project [21] aims to provide semantic search over large data corpora such as corporate intranets or email archives. When answering a user query, the system is looking for documents whose content matches the intent of the search query. This requires three semantic analyses. First, the semantics of a document needs to be determined, second the intent of the query must be identified and finally the match between the documents semantic and the query intent must be checked. The matching step is done by a look-up in a semantic search index, which is build a-priori by applying information extraction techniques on all documents of the corpus. In contrast to Situational Business Intelligence systems the Avatar system does not analyze documents is an ad-hoc fashion. Furthermore, it only provides search functionality rather than complex analysis features. The work in [44] presents methods to compute OLAP queries over uncertain data. Such data might result from applying information extraction techniques. Therefore, these methods might prove beneficial in the context of Situational Business Intelligence.

5.5 Conclusion

We introduced a novel a class of applications for answering situational business intelligence queries over text data. These applications process ad-hoc queries in a fast fashion. We also discussed serval major technological challenges that the data base community should solve, such as parallel processing of aggregation and join queries against a distributed index and the management of user defined functions for text mining.

The envisioned path for ad-hoc OLAP style query processing over textual web data may take a long time to mature. In any case, it is an exciting challenge that should appeal to and benefit from several research communities, most notably, the database, text analytics and distributed system worlds. In the following chapters we will discuss our strategies for enabling the query processor to drastically increase the execution speed of user defined functions for text mining.

6 Self -Supervised Search for Attribute Values

Often users are faced with the problem of searching the Web for missing values of a spread sheet. It is a fact that today only a few US-based search engines have the capacity to aggregate the wealth of information hidden in Web pages that could be used to return these missing values. Therefore exploiting this information with structured queries, such as join queries, is an often requested, but still unsolved requirement of many Web users.

A major challenge in this scenario is identifying keyword queries for retrieving relevant pages from a Web search engine. We solve this challenge by automatically generating keywords. Our approach is based on the observation that Web page authors have already evolved common words and grammatical structures for describing important relationship types. Each keyword query should return only pages that likely contain a missing relation. Therefore our keyword generator continually monitors grammatical structures or lexical phrases from processed Web pages during query execution. Thereby, the keyword generator infers significant and non-ambiguous keywords for retrieving pages which likely match the mechanics of a particular relation extractor.

We report an experimental study over multiple relation extractors. Our study demonstrates that our generated keywords efficiently return complete result tuples. In contrast to other approaches we only process very few Web pages.

The content and ideas of this chapter are published in [129][130].

6.1 Introduction

Situational Business Intelligence Applications (see Chapter 5) require methods to mine the Web for actionable business insights. For example, consider the following query:

```
-- Assign search results and extractors to temporary tables

DEF Acquisition(Acquirer, Acquired, Status, DateString)
    AS OpenCalais.Acquisition
    FROM yahoo.com USING FIRST 10

-- Define result attributes and join predicate

SELECT    MyCompany.Company, Acquisition.Acquired,
          Acquisition.Status, Acquisition.DateString
FROM      MyCompany, Acquisition
LEFT OUTER JOIN Acquisition ON   MyCompany.Company = Acquisition.Acquirer
```

MyCompany.Company	Acquisition.Acquired	Acquisition.Status	Acquisition.Datestring
United Natural Foods	Canadian Food Distribution	announced	Monday, August 5 2002
United Natural Foods	Food Distribution Ass	known	Jun 14, 2010
United Natural Foods	Boulder Fruit Express	known	November 2001
United Natural Foods	Blooming Prairie	announced	Wednesday, August 7 2002
United Natural Foods	SunOpta Distribution Group	known	May 19, 2010
Oracle	MySQL	planned	February 18th, 2006
Oracle	Sun Microsystems	announced	April 20, 2009
Oracle	Tacit Software	announced	November 4, 2008

Figure 30: Example query and result excerpt

Given a list of companies defined in table MyCompanies, augment information from news pages on the Web about acquisitions, company headquarter, size of employees, product customers and analyst recommendations.

It is a fact that today only a few US-based search engines have both the capacity and the know-how to index and to aggregate the Web for answering such queries. Because of this monopoly, previous systems for executing such queries [67][68][69] assumed a full access granted to all indexed pages for answering the query from above. For instance, a possible scenario for these systems is a query that is directly executed on the entire index by an employee of the Yahoo search engine.

However, the majority of Web users do not have full access to these pages. Moreover, due to the nature of business models of Web search engines it is unlikely that a full access to the entire index is ever provided for each Web searcher. Because of the plethora of possible relation extractors, it is very

unlikely that a Web search engine will ever provide extracted tuples for all of these extractors and for each Web site.

Contributions: Ideally, Web searchers could query the Web like a relational database. For realizing this interesting and novel application scenario the chapter focuses on the specific problem of complementing a given list of values with information (values) from the Web. Our approach automatically identifies discriminating keywords to return a small set of relevant web pages using an existing search engine, like Yahoo Boss, from which tuples for a structured query are extracted. Other aspects of the overall process of augmenting given tuples with missing values from the Web have been presented in a previous publication [9].

Self-supervised Web search for obtaining missing values in a table: Extracting tuples from pages is costly and time consuming. For instance, state-of-the-art extraction services (such as OpenCalais.com) require multiple seconds to download a page and to extract relations. Ideally, generated keywords strategy return only pages that likely contain a missing relation. Thereby, keywords should help to retrieve only pages that also match the mechanics of the extractor, otherwise relevant pages are retrieved but relations are not recognized by the extractor. Keywords should be generated without analyzing a large set of sample pages from the search engine. Finally, keywords should adapt constantly to new pages published on the Web.

To solve this problem we designed a self-supervised keyword generation method. Our method returns highly selective keywords: It omits *ambiguous* keywords (keywords that appear with multiple relationships) and only returns most *significant* keywords (keywords that frequently correlate with a particular relationship).Our method is efficient since keywords are observed and extracted only from pages that are already retrieved during query execution. Contrary to the method proposed in [71], our method does not require processing additional pages before executing a structured query. Finally, generated keywords are constantly updated with words and grammatical structures of Web pages that are discovered during query processing.

Prototypical implementation and experimental study: We implemented our algorithms for the in-memory database *HSQLDB [77]*. In a preliminary study we test our algorithms findings on structured queries invoking multiple relationship extractors. We compared and identified keyword generation strategies that can effectively reduce the amount of processed documents.

The rest of the chapter is organized as follows: Section 6.2 introduces our novel application scenario and reviews our existing work. In section 6.3 we introduce our novel algorithm to generate discriminative keywords. In section 6.4 we evaluate the performance of our algorithms. Section 6.5 compares our contributions to related work. Finally, in Section 6.6 we give a summary and discuss future work.

6.2 Augmenting Tables with Information from Search Results

In this section we give a brief overview on our query processor for executing queries of the type 'augment specific-k items' on Web search results.

6.2.1 Answering structured queries from Web search results

The long term vision of our work is to answer structured queries from natural language text on Web pages (see also Chapter 5 and our work in [136][137][133]). In this chapter we focus on queries of the type AUGMENT specific-k items. In this query type a list of known entities is complemented with relations that are extracted from Web pages. Consider again the query from the introduction:

Example: Augment specific-k items*: "Given a list of companies defined in table MyCompanies, augment information from pages of cnn.com about acquisitions, company headquarter, size of employees, product customers and analyst recommendations.*

This query takes as input a table with one or multiple companies. For each company pages are retrieved from the search engines CNN.com that likely contain information about acquisitions, size of employees, product customers or analyst recommendations. In addition, we identified two additional, similar query types:

Example: Discover any-k items*: Projects (or) funders) of climate adaptation projects in Africa from the Web.*

The goal of this query is to find names of projects and/or the names of funders associated with such projects or funders who work on climate adaptation in Africa more generally (even the name of the funder without the name of the project). However, this query does neither specify a search engine nor a Web page nor a list of already known entities, such as companies or project names. Queries for discovering any-k relations enable users exploring and discovering previously *unknown* relations without manually searching, browsing and reading thousands of Web pages. Users can browse and refine these tables further, e.g., by verifying the authenticity of surprising relations with additional search queries.

Example: Verify structured data with information from Web pages*: The PPI database of the 'World Bank' includes lists of projects. Engineer a search that finds all privately financed water and sewerage projects in Colombia since 1984 (the PPI database lists names of 51 distinct projects).*

The goal of this query is to complement existing structured data with relations from the Web and to identify 'linage' information (such as Web pages or Blog authors) for retrieved relations.

Neither Web search engines nor business intelligence tools can answer this query today which requires an intersection of their business (and user access) models, content data bases and technical capabilities. For answering these queries our work address the following key-challenges.

Timely answer: Extracting relations from pages is costly and time consuming. For instance, state-of-the-art relation extractors (such as OpenCalais.com) require multiple seconds for downloading a page and for extracting relations. One possible approach is utilizing parallelization techniques for executing multiple queries in parallel [83]. The approach presented in this chapter avoids forwarding irrelevant pages to relation extractors by generating highly discriminative keyword queries.

a) Obtain extracted sentences, entities and relations from pages.

Acquisition	**<Company_Acquirer:Tata Motors>** had acquired **<Company_Acquired:JLR>** for $2.3 billion.
Acquisition	**<Company_Acquirer:Brunswick Corp>** on Monday said it has sold **<Company_Acquired: Sealine>**.

b) Generalize sub-query-dependent attributes in sentences.

Acquisition	**<Company_Acquirer>** had acquired **<Company_Acquired>** for $2.3 billion
Acquisition	**<Company_Acquirer>** on Monday said it has sold **<Company_Acquired>** .

c) Extract sub-query-independent lexico-syntactic features.

Acquisition	**<Company_Acquirer>** [VBD/had] [VBN/acquired] **<Company_Acquired>** [IN/for] [$/$] [CD/2.3] [CD/billion].
Acquisition	**<Company_Acquirer>** [IN/on] [NNP/Monday] [VBD/said] [PRP/it] [VBZ/has] [VBN/sold] **<Company_Acquired>.**

d) Classify sentences and extract sub-query-dependent words and phrases.
(here <Company_Acquirer> * <Verb Group><Company_Acquired>)

Acquisition	**<Company_Acquirer>** had acquired **<Company_Acquired>**
Acquisition	**<Company_Acquirer>** has sold **<Company_Acquired>**

e) Rank and select discriminative phrases for sub-query (here DCM> 0,1)

Missing Relation	Existing Relation Attribute Type	Generated Keywords	Missing Relation Attribute Type	DCM
Acquisition	Company_Acquirer	had acquired	Company_Acquired	0,4
Acquisition	Company_Acquirer	has sold	Company_Acquired	0,27

f) Generate keyword query for sub-query with discriminative phrases
Parameters: *se=Yahoo.com; Acquisition.Company_Acquirer = "SAP AG"*

+"SAP AG" +("had acquired" OR "has sold")

Figure 31: Generating discriminative keyword queries

sult tuples may frequently occur. One important quality criteria is result tuple completeness. For instance, queries invoking joins across multiple relations require complete values for join attributes. Moreover operators for cleansing, fusing, merging, verifying [72], aggregating and ranking [66] require complete tuples to create authentic results. Unfortunately, relation extractors frequently return incomplete tuples. The rationale for this incompleteness is to indicate human evaluators the existence of a relation on a Web page, even though precise values for each attribute of the relation are not extracted. Worse, extractors may frequently fail because of missing textual data in the Web page.

6.2.2 Expressing Queries

Relation extractors and search engines as table generators*:* Our query language uses relation extractors as table generators which are integrated with data manipulation operations. With this functionality a user can select a powerful combination of qualitative search engines and of high-precision extractors that likely return relevant result tuples answering the structured query. Based on our language, representations of tuples in natural language from Web pages are mapped to the explicit integrated schema defined in the user query. The table generation is done by a *DEF*-clause (which is similar to the *WITH* clause in SQL). A *DEF*-clause defines the extractor, the search engine, the top-pages and optional keywords used to create tuples for a temporary table. In the temporary table we store attribute values of the tuple, pages from which attribute values are extracted and the completeness of each tuple.

Left outer join*:* Our query language allows Select-Project-left outer join query structures. The outer table for the join should already contain relations. For executing a left outer joining on these relations users can define a join predicate. For instance, the join predicate can be an object identifier that is shared by objects in the outer table and relations that are returned from the relation extractor. Otherwise a fuzzy join can be executed on the textual representation of the object.

Example: Figure 30 *illustrates the representation of the example from the introduction query in our language. The query augments a set of companies in table MyCompanies with tuples of the relationship type Acquistion*

Lexico Syntactic	-> Example Relationship	-> Observed Canidate Phrase
<A>VP<A>	ACQ(Acquirer, Acquired)	<ACQ. Acquirer>had acquired<ACQ. Acquired>
<A> VP<A>	HQ(Company, Location)	<HQ. Company>headquarter in<HQ.Location>
<A>VP + Prep <A>	Prod(Product, Company)	<Prod.Product>manufactured by<Prod.Company>
<A> to + infinitive <A>	Prod(Product, Company)	<Prod.Company>to produce<Prod.Product>
<A>VP<A>NP	Award(Person, Company)	<Award.Person>recieved<Award.Company>grant
<A> and | , <A>VP	Marraige(Person1, Person2)	<Marriage.Person1>and<Marriage.Person2>marry

Figure 32: Lexico-syntactic pattern for the English Language

(Acquirer, Acquired, Date, Status). The query defines the table generator 'Acquisition' which retrieves top-10 pages from the search engine "yahoo.com" and forwards pages to the extractor OpenCalais.Acquisition. Tuples from the 'left' table MyCompanies are joined with relations returned from the extractor on the text of the company attribute.

6.2.3 Processing Queries

In [130] we proposed a query processor for executing join operations on search results from Web search engines. This technique is applied for answering queries of the type 'augment specific-k items'. We briefly repeat the two main steps:

- ***Iterate over outer table:*** Similar to a nested-loop join our query processor scans the outer table (which contains the specific-k items) row by row. For each row in the outer table we generate a keyword query for obtaining a missing relation.
- ***Query index of search engine, retrieve pages, extract and combine relations:*** The keyword query scans the index of billions of pages of a Web search engine. This operation is executed in a timely fashion by leveraging the powerful infrastructure of existing search engines. A keyword query returns pages which are forwarded to a relation extractor. Relations extracted on the page are combined with rows of the outer table according to the structured query.

Query termination: The adaptive process of generating keyword queries, extracting relations, combing relations into result tuples is terminated when all rows in the outer table are processed. During query processing the keyword generating component learns how likely a keyword query returns a relevant page.

6.3 Continuous Keyword Query Generation

In this section we introduce our strategy to identify, generalize and extract meaningful keyword phrases from Web pages.

6.3.1 Overview and Requirements

The average Web user is often not aware of relevant keywords for returning pages from an extractor can return tuples for a particular relationship. Even if such keywords are known, determining keywords that best fit the mechanics of a relation extractor is difficult.

Our keyword generation strategy is based on the key observation that human language has already evolved words and grammatical structures for relationships. Our approach monitors the ability of the 'mechanics' of an extractor to capture these words and grammatical structures for a particular relationship. Figure 31 gives an overview on this process: First, we observe if sentences share instances of the same relationship also share similarities in textual content. Next, we keep phrases from these sentences that indicate the existence of an instance of a specific relationship (i.e., an *acquisition*) by using an efficient *lexico–syntactic* classifier. Then, we determine for each phrase their clarity with respect to other relationships and their significance for a particular relationship. From phrases that achieved both, a high clarity and a high significance, we generate a keyword query for a specific search engine and relationship type.

Our method is designed towards an online classification setting: It is completely unsupervised, requires no parameter tuning and requires little computation overhead. As an important highlight the set of keywords is no longer static (i.e. defined by a human or learned from a static corpus [71]) but is constantly updated by new phrases which are observed during query exaction.

6.3.2 Step 1: Extract Candidate Phrases

A page may contain many irrelevant words for extracting a particular relationship. Therefore, we consider only words from sentences in which an extractor identified a relation and attribute values. We generalize attribute values to the name of the attribute type returned by the relationship extrac-

tor. Next, we obtain lexical information (phrases of words) appearing before, between and after these attribute values and we generate syntactic information (part-of-speech tags) for each sentence.

Example: *Figure 31a-c shows sentences which contain a relation of the type Acquisition (Acquisition.Acuqirer, Acquisition.Acquired). In each sentence, attributes and attribute values of the extract relation are labeled (Figure 31a). Next, attribute values in each sentence are generalized to their corresponding attribute type (Figure 31b). For instance, the values 'Tata Motors' and 'Brunswick Corp' are generalized to the attribute type 'Acquisition.Acquirer'. Finally, words before, between and after attribute values are labeled with part-of-speech information (Figure 31c).*

6.3.3 Step 2: Classify and Filter Phrases

Labeled sentences may still contain words that are not relevant for extracting relations of a particular type. For instance, in the sentence: "*<Acquisition.Aquierer> has sold <Acquisition.Acquired> for $2.3 billion."* the adverbial information "for $2.3 billion" is not relevant for the type *Acquisition(Acquisition.Acuqirer, Acquisition.Acquired)*. One option to identify the structure of complex and compound sentences is a time consuming deep linguistically analysis of each sentence. Fortunately, recent research [73] has shown that 95% of binary relationships are expressed in English sentences by a set of few *relationship independent* lexico-syntactic patterns that can be detected with a shallow analysis (part-of speech tagging). For instance, the pattern *<A><verb-phrase><A>* enforces an instance of an attribute type that is followed by a group of verbs that are followed by another instance of an attribute type. We keep only phrases that follow at least one of these patterns. Next, we label remaining phrase with the relationship type that is returned from the relation extractor.

Example: *Figure 32 shows our set of relationship independent lexico-syntactic patterns, five example relationship types and six candidate phrases. For instance, for relationship Prod(Product, Company) two phrases in diathesis voices are extracted. Figure 31d shows two phrases '<Acquisition.Acquirer> had acquired <Acquisition.Acquired>' and '<Acquisition.Acquirer> has sold <Acquisition.Acquired>' for relationship type ACQ(Acquirer, Acquired).*

6.3.4 Step 3: Identify Discriminative Phrases

A crucial task of the keyword generation algorithm is to determine (1) if the phrase is ambiguous for multiple relationships and (2) if the phrase is significant for a particular relationship. For instance, the phrase *"announces"* is ambiguous, since it frequently appears together with relations of multiple relationships, such as *Acquisition*, *CareerChange* or *CompanyExpansion.* In addition, for the relationship *Acquisition* multiple phrases appear, such as *announces (97), acquires (78), compete (28) ... completes (18) ...rebuffed (1).*

Identifying highly significant and unambiguous phrases for a relationship is similar to the problem of identifying discriminative features for document classification [78]. Originally, the discriminative category matching approach (DCM) measures the (1) relative importance of a feature for describing a document and (2) for describing and the relative importance of a feature to describe a category of documents. We transfer these ideas to our scenario and measure the significance and clarity of a phrase for returning only pages which likely contain relations of a specific type.

Relative significance of a phrase for a relationship: We consider a phrase as significant, if it appears frequently among all phrases for a particular relationship. The significance is relative to the set of observed pages. The relative $Significance_{i,R}$ of a phrase i for a relationship R is given by the following equation:

$$Significance_{i,R} = \frac{\log_2(f_{i,R} + 1)}{\log_2(P_R + 1)}$$

The term $f_{i,R}$ denotes the number of sentences where phrase i was observed for relationship R. P_R denotes the number of *all* sentences where a phrase for *any* relationship R is detected. Since a phrase which appears n times cannot imply that the phrase is n times more important we use a logarithmic scale rather than a linear scale.

Relative clarity of an extracted phrase across relationships: Consider an extractor which recognizes multiple relationships. We consider a phrase as *ambiguous* if the *same* phrase is frequently observed for *multiple* relation-

ship types $R_1 \ldots R_n$. A phrase achieves a high *clarity,* if we frequently observe the phrase only for a *single* relationship. The following equation captures this intuition. The C*larity*$_i$ is relative to observed relationships and Web pages. We denote *Clarity*$_i$ of a phrase *i* as:

$$Clarity_i = \begin{cases} \log \frac{n * \max\limits_{j \in \{1..n\}} \{Significance_{i,R_j}\}}{\sum_{j=1}^{n} Significance_{i,R_j}} * \frac{1}{\log n}, & n > 1 \\ 1, & n = 1 \end{cases}$$

In the equation above, *Signficance*$_{i,Rj}$ describes the significance of a phrase *i* for a specific relationship R_j. The variable *n* counts the number of relationships $R_1 \ldots R_n$ for which the significance of phrase *i* is observed. *Clarity*$_i$ has some interesting and intuitive characteristics: *Clarity*$_i$ of phrase *i* drops, if individual significance values of phrase *i* for multiple relationships are growing. Contrary, the *Clarity*$_i$ of phrase *i* is high, if we observe a high significance for a single relationship and low significance for other relationships. If we observe a phrase only once and only for a single relationship we set *Clarity*$_i$*=1*. We normalize $0 \leq Clarity_i \leq 1$ with the term *1/log(n)*.

Discriminative phrases for a relationship. A phrase has a high discriminative power, if we observe for both values, *Significance*$_{i,R}$ and *Clarity*$_{i,}$ *equally high values*. The following equation captures this intuition. We identify the most discriminative phrases *i* for a relationship *R* as with the metric *Discriminative*$_{i,R}$ as follows:

$$Discriminative_{i,R} = \frac{Significance_{i,R}^2 * Clarity_i^2}{\sqrt{Significance_{i,R}^2 + Clarity_i^2}} * \sqrt{2}$$

The term $\sqrt{2}$ serves as normalization factor with $0 \leq DP_{i,R} \leq 1$.

Example: *We observe two phrases with* $f_{has\ sold,\ Acquisition}=15$ *and* $f_{had\ acquired,\ Acquisition}=43$. *We compute* $Significance_{had\ aquired\ ,Acquisition}$ *as:*

$$Significance_{had\ aquired,\ Acquisition} = \frac{log_2(43+1)}{log_2(15+43+1)} \sim 0.93$$

Analogue, we compute $Significance_{had\ aquired,\ CompanyInvestment} \sim 0.17$. *Then we compute* $Clarity_{had\ acquired}$ *as:*

$$Clarity_{had\ acquired} = log\frac{2*0.93}{0.17+0.93} * \frac{1}{log(2)} \sim 0.78$$

Finally, we compute $Discriminative_{had\ acquired,Acqusition}$ *as:*

$$Discriminative_{had\ aquired,\ Aquisition} = \frac{0.93^2 * 0.78^2}{\sqrt{0.93^2+0.78^2}} * \sqrt{2} \sim 0.61$$

In a similar way we compute $Discriminative_{had\ acquired,\ CompanyInvestment} \sim 0.03$. *We conclude that the phrase 'had acquired' is more discriminative for the relationship 'Acquisition' than for the relationship 'CompanyInvestment'.*

6.3.5 Step 4: Generate Search Engine-specific Keywords

We create the keyword query by concatenating the set of discriminative phrases for a particular relationship into a string. The string is complemented with the attribute value of the join predicate. Each search engine has a slightly different syntax to express conjunctions, disjunctions and phrases. Therefore, the generation of this string depends on the search engine *e* that was specified in query *Q*. The number of phrases considered by the search engine *e* is an unknown parameter to the user. We observed that most search engines tolerate at least five phrases. Future research is required to observe this parameter in a more automatic fashion.

***Example**: Figure 31f shows an example keyword query that is generated for the search engine boss.yahoo.com. It contains the two most discriminative keywords for the relationship Acquisition. Following the syntax of boss.yahoo.com the symbol '+' denotes a conjunction and the symbol "OR" a disjunction.*

ac: Acquisition(Acquirer, Acquired, Date, Status); 'acquisition'

ar : AnalystRecommendation(Company_Rated, Company_Source, Financial_Trend); ' analyst recommendation'

ce : CompanyEmployeeNumber(Company, Employeesnumber); 'company employees number'

cl : CompanyLocation(Company, City, LocationType); 'company location'

cp : CompanyProduct(Company, Product, Producttype); ' company product'

Figure 33: Relation extractors and join predicates for benchmark queries

Continuously update values for Significance$_{i,R}$, Clarity$_i$ and Discriminative$_{i,R}$. Our method is designed towards an online classification setting: It is completely unsupervised, requires no parameter tuning and requires little computation overhead. During query processing, we constantly update observations about common keyword phrases and values for *significance*, *clarity* and *discriminative*. Thereby, our set of keywords is no longer static (i.e. defined by a human or learned from a static corpus at query definition time) and reflects additional phrases that are observed Web pages discovered during query execution. Note, that the DCM has been shown to provide similar high classification accuracy as a comparable classifier based on a Support Vector Machine (SVM) in many other classification scenarios.

6.4 Experimental Study

We designed a prototype for evaluating our keyword generation strategies based on the in-memory database HSQLDB [77]. In this section we report on our results.

6.4.1 Evaluation Setup and Metrics

Queries and data set: Currently no benchmark for extracting and joining tuples across Web search results exists. In this study we tested our system on 5 queries which are shown in abbreviated form in Figure 33. As 'left table' we took a list of 200 randomly chosen companies from the Fortune list [80]. Our benchmark queries utilize 5 relationships with 10 different

attribute types. The corresponding join predicate is underlined in each benchmark query. All queries utilize the extraction service OpenCalais.com [75] and the search service Yahoo Boss [74].

***Competitor strategies**:* The most interesting comparative experiments are concerned with different keyword generation strategies for the same query. We tested the following strategies:

V (Attribute value): Keywords are generated from the text of the attribute value. With this strategy we consider a naïve user that takes as input for a search the name of a given attribute value (here a company). This strategy serves as our baseline.

V+RT (Attribute value + RT-Name): Authors in [76] propose generating keywords from the attribute value and the name of the relationship. Names for relationship types are represented in OpenCalais.com by concatenating multiple words into a single string, for instance '*AnalystRecommendation'*. This string unlikely appears on a Web page. Therefore we manually tokenized names of relationship types into meaningful word sequences, such as *'analyst recommendation'*.

V+GK (Attribute value + generated and dynamically updated keywords): We generated keywords with our method from Section 6.3. In this strategy keywords are updated during query execution.

V+EK (Attribute value + keywords defined by experts): We utilized user generated Keywords. Only humans are familiar with the sheer endless expressiveness of common words and structures for expressing relationships of a particular domain. Therefore we consider as most competitive 'opponent' for our keyword generation strategy a human. Our human opponent is aware with technical limitations of state-of-the-art relationship extraction techniques. Contrary, the typical user of a search engine rarely combines such profound knowledge in practice. To simulate our human opponent, we studied the 'mechanics' of information extraction systems Textrunner [73] and DIAL [79] that are used by the service OpenCalais.org. In addition we observed words and structures for expressing relationships from common news sites (that are frequently returned from our queries). Based on this knowledge, we asked experienced engineers of our

system to manually craft and tune the best possible keywords. Figure 35 presents those user-defined keywords.

Test environment: We implemented these strategies for the java-based in-memory database HSQLDB [77]. Experiments are conducted on a T61 laptop, with 4 GB of RAM, a core duo CPU with 2.2 GHz, Windows 7 professional and Java 1.6_017.b04. Each benchmark query was executed six times and the average value was computed across experiments.

Measurements: In each experiment we compared the different strategies. For instance, we are interested in the average costs to complete a single tuple for a query *(P/RT)*. Keeping this amount low significantly reduces monetary and processing costs[4]. In addition, we measure how many tuples from the left table are completed *(RT)* with a given amount of costs (in this experiment set to 2000 pages). A query may return irrelevant pages. For instance, a query may return pages from which tuples are extracted that could not be joined. Therefore we measured the proportion of the overall size of extracted tuples *(Buf)* vs. extracted tuples that are part of the query result *(%RT/Buf)*. We also measured the proportion of relevant pages among the top-10 pages that contained a relevant tuple with *the mean average precision (MAP)* [81].

$$\mathrm{MAP(Q)} = \frac{1}{|\mathrm{Q}|}\sum_{\mathrm{j=1}}^{|\mathrm{Q}|}\frac{1}{|\mathrm{m_j}|}\sum_{\mathrm{r=1}}^{\mathrm{N}}\big(\mathrm{P(r)\ x\ rel(r)}\big)$$

MAP is a common metric to average the effectiveness of rankings from multiple keyword queries for each benchmark query Q. Among evaluation measures, MAP has been shown to have especially good discrimination and stability. In the formula above m_j denotes the number of relevant pages which are returned for a keyword query. Finally, we compared the speedup of each strategy in terms of pages per result tuple to our base line.

---

[4] For instance, Yahoo Boss charges currently $0.30 dollar per 1.000 requests for the top-10 search results and OpenCalais [75] charges $2000 per 3.000.000 pages that are extracted with their service (April 2011).

The list below shows our measurements and the abbreviations used in our diagrams:

Q-ID **:** ID of Benchmark Query
#A **:** Distinct attributes in result tuples for query
RT **:** Returned complete result tuples for 2000 pages
Buf **:** Complete and incomplete tuples in buffer
Usage (%) : Proportion of complete tuples among all tuples in the buffer
P/RT **:** Processed pages per single complete result tuple
Speedup : **(**P/RT Baseline) / **(**P/RT Other strategy)
Terminated : Either 2500 pages or 200 rows have been processed
MAP (10) **:** Mean average precision (only among top-10 pages)

6.4.2 Experimental Results

Strategy 'V+GK' is a clear winner. We first compare quantitative measurements for different keyword generation strategies. Figure 35 presents our results: Strategy 'V+GK' consumes the fewest number of pages to obtain a single result tuple (P/RT). We observed a moderate speed up of strategy 'V+GK' in contrast to our baseline strategy 'V' by factor 2.5-5. Strategy 'V+GK' is also efficient: It returned the largest result tuples size (RT) for our maximum of 2500 processed pages. For instance, for query *ac* we observed that 292 distinct relations for relationship type analyst recommendation are joined with companies from the left table. Moreover, we observe the most efficient buffer usage for strategy 'V+GK'. For instance, for query *cp* and *ce* more than half the tuples returned by this strategy contain values for all attributes. Except for query *cl,* we observed the highest mean average precision value, *MAP(10),* for strategy 'V+GK'. The comparable high value for *MAP(10)* indicates that our strategy likely returns Web pages among the top-10 search results.

Comparing generated keywords against expert keywords: Generated keywords capture tenses and reflections of verbs which are rather unusual for human-generated keyword queries. Figure 35 shows generated keywords of the form 'to+infinitive' for relationship *Product* and keywords of the form 'noun+prepositions' for relationship *Employees*. We conclude that generated keywords are general applicable for various combinations extractors and search engines. Finally, and most importantly, generating

Q-ID	RT	EK (Top 5)	GK (Top 5)
ac	"acquistion"	"acquire", "merger", "sale", "proposed", "said"	"acquires", "buys", "acquisition of", "sold", "takeover of"
ar	"analyst recommendation"	"downgrades", "upgrades," "rates", "recommends", "buy"	"upgraded", "buy from", "rating on", "rates", "upped"
ce	"company employee size"	"employees", "employees in", "employs", "employee size", "hires"	"employs", "employs over", "employs about", "employs around", "employing"
cl	"company location"	"located", "headquarter", "based", "founded", "resides"	"based in", "located in", "headquarters in", "corporation of", "incorporated"
cp	"company product"	"launched", "installed", "introduces", "puts to market", "provides"	"introduces", "launches", "launched", "introduced", "presents"

Figure 34: Keyword phrases

keywords requires no involvement of the querying user requires no prior training and is therefore highly suitable for our online query setting.

Further observations: For the 'cl' query V+GK significantly outperforms V+EK despite a largely overlapping set of keywords. A closer investigation reveals that compound keywords of V+GK retrieve likely pages that contain multiple relations of the type ‘company location’. Also, we observed the fraction of top-10 pages which likely contains these relations is significantly larger for V+GK. In outer future work we will systematically compound and other keyword types for retrieving multiple relations per page. Moreover, we will investigate a classifier that only forwards promising pages from top 10 pages to a relation extractor.

6.5 Related Work

We briefly present the state-of-the-art for generating keyword queries in ‘text databases’, which despite the difference between Web search results

Strategy	Q-ID	#A	RT	Buf	Usage (%)	P/RT	Speedup	Terminated	MAP(10)
V	ac	4	89	396	22	28	1,00	max documents reached	0,07
V+RT	ac	4	211	802	26	12	2,38	max documents reached	0,21
V+GK	ac	4	**292**	1037	**28**	**9**	**3,28**	max documents reached	**0,29**
V+EK	ac	4	106	529	20	24	1,19	max documents reached	0,17
V	ar	4	0	200	0	--	--	max documents reached	0,00
V+RT	ar	4	17	**202**	8	139	--	max rows processed	0,01
V+GK	ar	4	**46**	218	**21**	**54**	--	max documents reached	**0,05**
V+EK	ar	4	32	209	15	78	--	max documents reached	0,01
V	ce	2	27	**208**	13	90	1,00	max rows processed	0,03
V+RT	ce	2	63	225	28	40	2,26	max documents reached	0,05
V+GK	ce	2	**141**	277	**51**	**18**	**5,07**	max documents reached	**0,17**
V+EK	ce	2	76	233	33	32	2,85	max rows processed	0,07
V	cl	3	37	**217**	17	68	1,00	max documents reached	0,05
V+RT	cl	3	71	252	28	35	1,92	max documents reached	0,06
V+GK	cl	3	**92**	251	**37**	**27**	**2,48**	max documents reached	0,10
V+EK	cl	3	85	253	34	29	2,30	max documents reached	**0,20**
V	cp	3	45	**217**	21	54	1,00	max rows processed	0,04
V+RT	cp	3	64	227	28	38	1,44	max rows processed	0,06
V+GK	cp	3	**146**	280	**52**	**17**	**3,14**	max documents reached	**0,11**
V+EK	cp	3	118	259	46	20	2,72	max rows processed	0,08

Figure 35: Experimental results

and text databases is closest to our work. In addition, we briefly review related work in the area of text analytics.

Automatic keyword query generation. A query processor requires discriminative keywords that likely return relevant pages from a Web search engine. Previous methods [71] obtained such keywords from a trained SVM-classifier. To train the classifier a large sample of training documents is collected prior executing a structured query from a 'text database' (with full access to all index pages). Next, tuples for each document in the sample are extracted. When a tuple is discovered in a document, words from the document are marked as positive examples for training a classifier. Finally, words are cross validated with documents from the training sample. The highest scoring words are used as keywords to obtain successive documents from the 'text database'. This approach has three major drawbacks for our setting: Training an SVM-based classifier is expensive, since costs are quadratic to the number of training examples. Next, training the classifier requires full access to documents which is an unrealistic assumption in

our scenario. Finally, the set of training and test pages is not fixed in our scenario, but may vary from user query to user query. Contrary, in our approach initial keywords are used to randomly download pages. Keywords are constantly updated when new pages are returned by the search engine. Therefore, generated keywords also match the language of new Web sites that are discovered during query execution. Another idea is presented in [76] where candidate keywords are extracted from semantically meaningful attribute type names or predicate values from a structured query. We do not expect meaningful attribute type names in a structured query. Rather we trust that Web page authors have already evolved common words and grammatical structures for describing important relationships in human language.

Information extraction and text analytics. The CIMPLE project [51] describes an information extraction plan with an abstract, predicate-based rule language. More recently, open information extraction techniques generalize relationships with relationship-independent lexico-syntactic patterns [43]. We rely on the system Open Calais [75] that uses both approaches.

6.6 Conclusion

Exploiting textual information with structured queries is an often requested, but still unsolved requirement of many Web users. A major challenge is to define discriminative keywords to return a small set of relevant pages from which a relation extractor returns tuples for a structured query. In this chapter we discussed and solved this challenge by generating keywords with a self-supervised algorithm. We report from our study that our keyword generation strategy effectively returns result tuples while processing very few pages.

7 BlueFact: Fact-Aware Document Retrieval

Exploiting textual information from large document collections such as the Web with structured queries is an often requested, but still unsolved requirement of many users. We present BlueFact, a framework for efficiently retrieving documents containing structured, factual information from an inverted full-text index. This is an essential building block for information extraction systems that enables ad-hoc analytical queries on unstructured text data as well as knowledge harvesting in a digital archive scenario.

Our approach is based on the observation that documents share a set of common grammatical structures and words for expressing facts. Our system observes these keywords using structural, syntactic, lexical and semantic features in an iterative, cost effective training process and systematically queries the search engine index with these automatically generated keyword phrases. Next, BlueFact retrieves a list of document identifiers, combines observed keywords as evidence for a factual information and infers the relevance for each document identifier. Finally, BlueFact forwards the documents in the order of their estimated relevance to an information extraction service. That way BlueFact can efficiently retrieve all the structured, factual information contained in an indexed document collection.

We report results of a comprehensive experimental evaluation over 20 different fact types on the Reuters News Corpus Volume I (RCV1). BlueFact's scoring model and feature generation methods significant outperform existing approaches in terms of fact retrieval performance. BlueFact fires significant fewer queries against the index, requires significant less execution time and achieves very high fact recall across different domains.

The content and ideas of this chapter are published in [126].

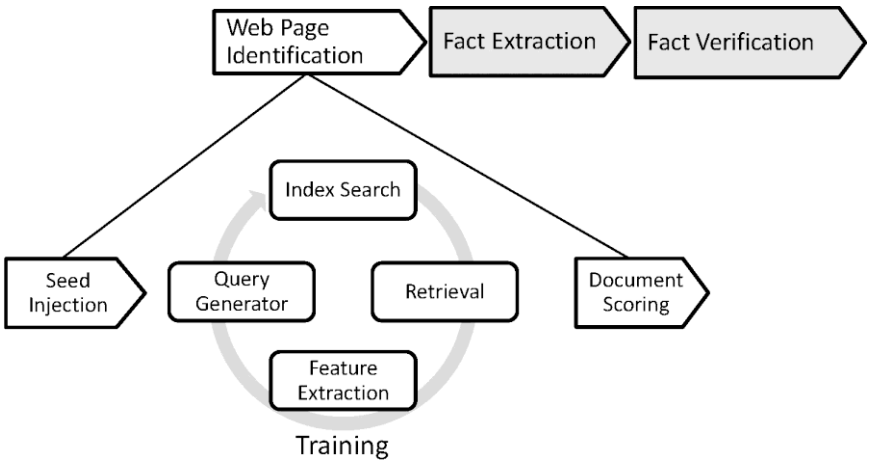


Figure 36: Data Flows and Operators for enabling structured ad-hoc analytical queries over unstructured text

7.1 Introduction

The availability of relevant information on markets, industries and technologies is crucial for any decision maker in today's world. Researching and compiling reports on such topics still requires skilled analysts and consultants even though most of the information necessary is most likely publicly available in the form of text documents. These unstructured texts represent the largest and fastest growing source of information available to businesses, governments and authorities today. It would thus be highly desirable to be able to leverage these sources and automatically extract and aggregate the information contained in them. Imagine a scenario where a company needs factual information about the criminal record of managers in the company. This information is highly important for the company, but might not be published in textual or structured form on information aggregators on the Web, such as Wikipedia.org, or Trueknowledge.com. Rather, this rare factual information is hidden as unstructured Web text in the 'long tail' on very few Web pages of news agencies or blogs.

We envision building a system that enables structured ad-hoc analytical queries over unstructured text data as outlined in Chapters 6 and 8. In such an ad-hoc query scenario we assume an analyst who formulates structured queries against an indexed document collection. Figure 36 illustrates the

data flow in this system. First relevant data sources such as documents or web pages have to be identified, second information extraction algorithms extract all facts contained in the data and third a separate component attempts to verify the extracted facts based on the diversity of sources that support a fact as well as their authority.

In this chapter, we focus on the first component, called "Web page identification" (see also in Figure 36). This component aims to identify and retrieve relevant documents that are likely to contain a fact of a certain fact type (e.g. the factual relation *Conviction* with the attributes *Person, Charge,* and *Date* in our introductory example). Prior efforts focused on generating fact-type-specific keyword queries for full text search engines (see also Chapters 6). We extend these approaches towards additional features, a sound theoretical retrieval model and a broad evaluation study on a standard text corpus. We present "'BlueFact"', an *efficient retrieval system of relevant pages those are likely to contain a fact*. This component is a crucial part of a system that enables ad-hoc analytical queries on unstructured text data, such as presented in Chapters 8, 9 and 10.

We present a comparative analysis of different methods to sample documents, to identify, extract and score potential keyword phrases (e.g. *"bought" or "acquired"* for fact type *acquisition*) and to score documents that likely contain facts. Our analysis is based on the Reuters Corpus Volume I (RCV1) from which we tagged 731, 752 documents with factual information with the OpenCalais information extraction service [75]. Furthermore, we re-implemented QXtract [71], a prominent prior approach to solve the fact-aware document retrieval problem, as well as five new feature generation methods that utilize lexical, syntactic, statistical and semantic models to generate keyword queries. We show in a comprehensive experimental evaluation that BlueFact drastically outperforms QXtract and other approaches in terms of fact recall, execution performance and is robust for multiple domains.

The remainder of this chapter is organized as follows: We discuss the theoretical base in Section 7.2 and derive features that indicate fact rich documents in Section 7.3. Section 7.4 introduces to our fact-aware document retrieval model and Section 7.5 shows our fact retrieval algorithm. After

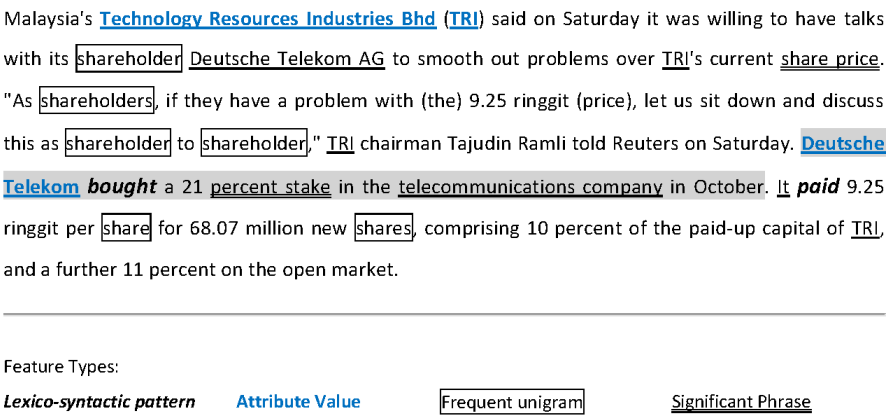


Figure 37: An example text from the Reuters News corpus annotated with feature candidates and a fact *Acquisition(Deutsche Telekom AG, TRI)*

that, we present in Section 7.6 the experimental setup and evaluate the performance of our algorithms. Finally, we summarize our work and propose future work.

7.2 Fact-Aware Document Retrieval

So far, only a few works on the distribution of facts in text collections and the identification of potential features that might indicate facts in text documents have been published. Prior approaches, such as QXtract [71], identify features in a rather ad-hoc manner. In this work we present for the first time a comprehensive and systematic analysis of the distribution of facts in a standard corpus of text documents and a formal approach to retrieve only documents that contain a fact. Next we present the evaluated feature extraction methods, our approach to inferring the relevance of documents and finally discuss our algorithm.

Formal Problem Statement. Information retrieval systems such as Web search engines usually index document collections with inverted indices. Words w out of a vocabulary W are mapped to documents d of a document collection D, where documents are usually treated as bags of words. A keyword query w executed against an index I will thus return a set of

documents which contain the keyword: $D_w = \{d \in D | w \in d\}$. Our goal is to leverage such an index to retrieve the set of facts

$$F = \bigcup_{d \in D} \text{extract(d)}$$

that can be extracted from the document collection while only processing a minimal subset of relevant documents $D_{min} \subseteq D$ containing all facts in F.

Keywords as evidence. Unfortunately such an index does not allow querying the system with a keyword *w* to retrieve a set of facts contained in the documents. When an inverted index structure returns a document for a given keyword, all that is known with certainty is that the keyword occurs at least once in this document. The challenge is to use this limited information as evidence about the otherwise unknown document collection and to somehow infer which documents are likely to contain a fact. To solve this problem we define a new function retrieveFacts(W_{seed}, F , I) which retrieves facts F of a particular type contained in a document collection. To be able to solve this problem, one need to estimate the probability that a document d actually contains a fact and process the most promising documents first. To infer this probability, we probe the index with a small set of extracted keywords and use the results as evidence to estimate the relevance of individual documents. Function retrieveFacts only requires a small set of handcrafted initial seed keywords W_{seed} as input to retrieve all facts from a document collection indexed with an inverted index I. In our notation, a fact is an instance of a semantic relation between two or more entities (e.g. *Company*, *Product*, *Location* ...). The type of a fact is thus the type of semantic relation that a fact expresses (e.g. *CompanyProduct* or *PersonCareer*).

7.3 Feature Extraction Methods

Based on our systematic corpus analysis in the Section 5.2.2 we identified the following feature extraction methods to extract potential keyword queries indicating a fact as feature candidates. We start with the method *user defined seed facts* to create an initial set of sample documents. Figure 37 shows a representative excerpt of a document in the Reuters news corpus that could be in the seed sample. The figure illustrates all potential feature

candidates available in this text. Previous approaches, such as Qxtract, only execute this one sampling phase to bootstrap a set of seed documents. We extend this approach and adaptively expand the set of sample documents with new feature extraction methods leveraging lexical, syntactic, statistical and semantic information. The methods are executed in an order such that the methods requiring the most sample documents are executed last.

7.3.1 Semantic Information

User defined seed facts. Similar to QXtract, this method utilizes handcrafted seed facts, such as *{"SAP AG", "Waldorf"}* or *{"IBM Research", "Hawthorne"}* as examples for the fact type *CompanyLocation* as keyword queries. These user chosen facts represent instances of a specific fact type.

Fact-type name. As proposed by the authors of [76], this method exploits the semantics of the fact-type name. We convert this name into a string representation, e.g. we tokenize the representation of the fact-type *CompanyProduct* into the conjunctive query *"company AND product"*.

Attribute values of retrieved facts. Analogue to [76], we use attribute values of facts extracted in phase one (e.g. *Company = "Oracle", "Microsoft",* . . .) to identify further instances of a fact type containing the same attribute value. Utilizing this semantic information returned from the information extractor, enables our system to identify further potential lexical expressions for the same fact-type and do not require statistics like the ensuing methods.

7.3.2 Statistical Information

Frequent unigrams. We extract the top-k most frequent unigrams from relevant documents and use them as new search queries. These terms help us to retrieve all relevant documents, thus increasing recall while only modestly decreasing precision.

Significant phrases. We extract the most frequently collocated pairs of words from relevant documents. For observing these frequent collocations we use the approach of [107] and the implementation in [108].

7.3.3 Syntactic Information:

Lexico-syntactic patterns. Recent work in techniques for open information extraction [73] shows that most semantic relations (facts) between two entities are expressed utilizing only few syntactic patterns, such as <E>VRB<E> or <E>VRB, PRP <E>. We utilize these features to identify sentences that contain a fact (see also Chapter 6.3.2). Together with the attribute values they contribute to a set of keywords that clearly indicate the presence of a semantic relation in the text.

However such information might not suffice to reach all relevant documents in the corpus. To overcome this potential limitation we also extract words (unigrams) and significant phrases that frequently co-occurred with factual statements in text.

7.4 Inferring the Relevance of Documents

QXtract essentially uses the search engine rank as an estimate and processes the documents in the order of their rank. However, our preliminary experiments (see also next section) suggested that this estimate is rather inaccurate and many irrelevant documents have to be processed by the information extractor.

In our approach we attempt to generate a more accurate estimate of the relevance of documents by introducing an adapted version of relevance feedback [109] to re-rank the search results, so that the pages that are most likely to contain extractable facts are ranked highest. We then re-rank all retrieved documents before they are forwarded to the extractor. In particular, we view the fact retrieval problem as an inference problem. After generating an initial sample of documents utilizing seed keywords, we extract potential features to be used as keyword queries from these sample documents. Our system is a generic framework which allows evaluating different feature generation methods. We extract words and phrases as features with lexical, syntactic and statistical feature generation methods. In a next phase we evaluate the usefulness of each feature and use this information to infer the relevance of the retrieved documents. Finally we use this information to re-rank the documents by their estimated relevance.

7.4.1 Formalizing the Inference Problem

Let Y be a random variable that encodes whether a given document actually contains a fact and X be another random variable from a joint distribution P (X, Y) encoding with which of the evaluated keywords a document has been found (thus encoding the occurrences of keywords in this document). Let us assume that the distribution is the same across the document collection. Our fact retrieval problem can then actually be expressed as inferring the (conditional) probability that a given document d contains a fact, given the keyword queries with which it has been found. This means we have to estimate the conditional probability of Y (whether a document contains a fact) given the keywords X which returned the document P (Y |X). By applying Bayes rule, the conditional probability can be rewritten as:

$$P(Y \mid X) = \frac{P(X|Y)P(Y)}{P(X)} \propto P(X|Y)P(Y)$$

By assuming that the keyword queries X with which the document has been found are conditionally independent given the label y, one can easily take the maximum likelihood estimates of the conditional probability P (X|Y) and the prior P (Y) from the evaluated keyword queries and then use these estimates to compute the probability that a document contains a fact for all unprocessed documents.

7.4.2 Method I: Naive Bayes Decision Function.

In a sense we see the sampled documents as an annotated corpus thus consists of training examples (xi,yi) which have been generated from the joint distribution P (X, Y). Estimating the conditional probability is then similar to learning the decision function of a *Naive Bayes* classifier. Note that this approach is quite different from the approach of *QXtract* where the decision functions of classifiers are used to rank unigrams of words to be used as keywords queries. We utilize these functions to score and re-rank documents based on the sample documents retrieved with the already generated keywords.

7.4.3 Method II: Fast Scoring Heuristic

In addition to the probabilistic inference method we also implemented a fast scoring heuristic where the effectiveness of a keyword query q is estimated with the F-Measure:

$$F = \frac{(\beta^2 + 1) * P * R}{(\beta^2 + 1) * P + R}$$

which is the weighted harmonic mean of precision P and recall R, a well-known measure in the context of information retrieval. It resembles the ability of a query to retrieve many facts (high recall) and only relevant pages that contain facts (high precision).

We define precision *P(q)* as the fraction of relevant documents retrieved via query q:

$$P(q) = \frac{|D_q \cap D_{relevant}|}{|D_{sample}|}$$

We estimate the recall *R(q)* as the fractions of facts f ∈ X which have been extracted via query q:

$$R(q) = \frac{|X_q \cap X|}{|X|}$$

To obtain the final *EvidenceValue* of a query q we weight its F-measure with the average F-Measure over all queries from this feature generation method m to adjust for the different overall effectiveness of the individual feature generation methods:

$$EvidenceValue(q) = F_\beta^{avg}(m) * F_\beta(q)$$

To obtain the *FactScore* of a document, we aggregate the EvidenceValues for each keyword query q that retrieved the document d:

$$FactScore(d) = \sum_{q \in Q_d} EvidenceValue(q)$$

7.5 The retrieveFacts Algorithm

We now introduce the *retrieveFacts(W_{seed}, F , I)* algorithm. Figure 38 lists this main algorithm of the framework in pseudo code which implements three phases.

Phase 1: Training and probing the index. The goal of this phase is to obtain reasonable statistics on the effectiveness of feature candidates that can ultimately be used to score documents according to their relevance for fact extraction. Furthermore, an extensive list of all potentially relevant document IDs (IDdoc) is obtained in the process and stored in the buffer Q. Hence, in the first phase the algorithm evaluates different feature generation methods *m* to extract promising words or phrases of single words from the relevant documents of the sample.

The algorithm starts with an empty set of sample documents D_{sample} in line 2 and a small set of initial, potentially relevant, words Wseed. For instance, these words could be derived from few user-defined tuples or from schema information of the fact extractor. In the first pass of the feature evaluation loop, we populate D_{sample} by querying the search engine with the provided set of seed keywords W_{seed}. These documents are forwarded to the information extractor to obtain any facts contained in the sample and to label each documents as either relevant or irrelevant. After this step we have a labeled set of sample documents.

For each subsequent feature generation method *m* (see Section 7.3) the algorithm proceeds as follows: First the algorithm extracts the top k feature candidates q from the sample documents in conjunction with the facts X extracted from them (line 4). Next, the algorithm queries the index to receive all document IDs (IDdoc) matching the query (line 5) and retrieves the top n ranked documents (line 6), which are then added to the sample documents (line 7). The important information, which document IDs were retrieved with query q of feature type m, is stored in our buffer Q (line 8). The algorithm forwards all n new pages to the extractor to obtain the contained facts (line 10). If facts could be extracted from a document, they are added to the fact buffer X and the document is marked as relevant (lines 11-13). This procedure is repeated for all surveyed feature categories *m* (see also Section 7.3).

Algorithm 1 RetrieveFacts(W_{seed}, F, I)

```
1: k ← 100; n ← 10; Q ← ∅; X ← ∅;
2: D_sample ← ∅; D_relevant ← ∅;
   {Phase 1: Training and probe phase}
3: for all m ∈ FeatureGenerationMethod do
4:    for all q ∈ m(D_sample, D_relevant, W_seed, F, X, k) do
5:       ID_doc ← Search(q, I);
6:       D_crawled ← Retrieve(Top(ID_DOC, n));
7:       D_sample ← D_sample ∪ D_crawled;
8:       Q ← Q+ < q, m, ID_Doc >;
9:       for all d ∈ D_Crawled do
10:         X_New ← Extract(d, F);
11:         if |X_new| > 0 then
12:            D_relevant ← D_relevant + d;
13:            X ← X ∪ X_New;
14:         end if
15:      end for
16:   end for
17: end for
   {Phase 2: Score and re-rank document identifiers}
18: ID_Score ← ∅;
19: for all < q, m, ID_doc >∈ Q do
20:   for all id ∈ ID_doc do
21:      score_Id = FactScore(id, q, m);
22:      if ID ∉ ID_score then
23:         D_scored ← D_scored+ < ID_score, score >;
24:      else
25:         score_id ← score_id + GetScore(ID_score, id);
26:         Update(ID_score, id, score_id);
27:      end if
28:   end for
29: end for
   {Phase 3: Collection Phase - retrieve documents and extract
   facts}
30: D_ranked ← SortDesc(D_scored);
31: while budget do
32:   X ← X ∪ Extract(Retrieve(Next(D_ranked)), F);
33:   Decrease(budget);
34: end while
```

Figure 38: Algorithm retrieveFacts(W_{seed}, F, I)

Phase 2: Score and re-rank document identifiers. As outlined in Section 7.1, we try to estimate the number of facts that can potentially be extracted from a document. We make this judgment based on prior evidence, namely

we have a comprehensive graph that links already evaluated keyword queries q to the documents that have been retrieved with them and the facts in X that could be extracted from these documents.

Therefore in phase 2 we iterate through each query q, estimate its effectiveness and finally score document ids based on their estimated usefulness (lines 16-27). This scoring can be done either with our fast scoring heuristic or by actually solving the inference problem (see Section 7.4). Note, that this scoring does not require downloading the content of the document since it only bases on observed features for a document from index I.

Phase 3: Collect facts from re-ranked documents. After all the document ids have been scored the entire list of document ids is sorted (line 30). The corresponding pages are then forwarded to the information extractor in the order of their relevance until the budget is exhausted or all pages have been processed (lines 30-33).

7.6 Experimental Setting

In this section we report metrics we use to evaluate alternative fact retrieval systems from Sections 7.1 and 7.7 discuss how to set optimal parameters for fact scoring models (Section 7.4).

7.6.1 Evaluated Fact Retrieval Systems

We experimentally compare different fact retrieval systems on a Desktop PC with 8 GB RAM, a Quad Core CPU with 3 GHz and a SATA hard disc with 7200 r/pm. To be able to efficiently searching the document collection we created a Lucene index [109] on the evaluation corpus and implemented the following five fact retrieval systems in Java 1.5 and stored experimental results in the in-memory database HSQLDB 2.0 [77].

- *Baseline.* This simple baseline forwards documents from the corpus to the extractor in a random order.
- *Qxtract.* We implemented system QXTRACT as described in Section 7.7.

		$k = 10$	$k = 50$	$k = 100$	$k = 200$
$n = 5$	MAP	0.01448	0.01306	0.01232	0.01171
	Var(MAP)	0.00126	0.00113	0.00102	0.00097
	Recall	0.98972	0.99947	0.99989	0.99999
$n = 10$	MAP	0.01611	0.01256	0.01186	0.0111
	Var(MAP)	0.0015	0.00104	0.00097	0.00089
	Recall	0.98371	0.99917	0.99999	1
$n = 20$	MAP	0.01591	0.01212	0.01125	0.01038
	Var(MAP)	0.00142	0.00103	0.00091	0.00081
	Recall	0.97879	0.99939	0.99995	1
$n = 50$	MAP	**0.01705**	0.01136	0.01041	0.00942
	Var(MAP)	0.00177	0.00096	0.00081	0.00068
	Recall	0.97978	0.99959	0.99998	1
$n = 100$	MAP	0.01661	0.01089	0.01	0.00918
	Var(MAP)	0.00168	0.00089	0.00075	0.00063
	Recall	0.98018	0.99956	0.99998	1
$n = 200$	MAP	0.01524	0.01033	0.00962	0.00924
	Var(MAP)	0.00135	0.00079	0.00064	0.00057
	Recall	0.97966	0.99955	0.99996	1

Figure 39: Parameter setting: MAP and recall for different parameter settings of *k* (top keyword feature candidates) and *n* (top documents). Please note that this is the MAP summed over all documents not just the top 10 or 100 which is why the MAP score appears to be uncommonly low. Only the absolute difference of the individual MAP scores is of interest.

- *BlueFact.* We implemented system BlueFact with the fast scoring heuristic as described in Section 7.4.3.

- *Naive Bayes.* As an alternative scoring function for Blue-Fact we implemented the decision function of a Naive Bayes classifier (see Section 7.4.2).

- *Ideal.* This implementation represents the ideal fact retrieval system. It only forwards documents that contain one or multiple facts to the extractor and processes documents with most facts first.

We evaluate each fact retrieval system on the Reuters news corpus (see also Section 5.2.2) and selected 20 representative fact-types from the domains *person*, *product* and *company*.

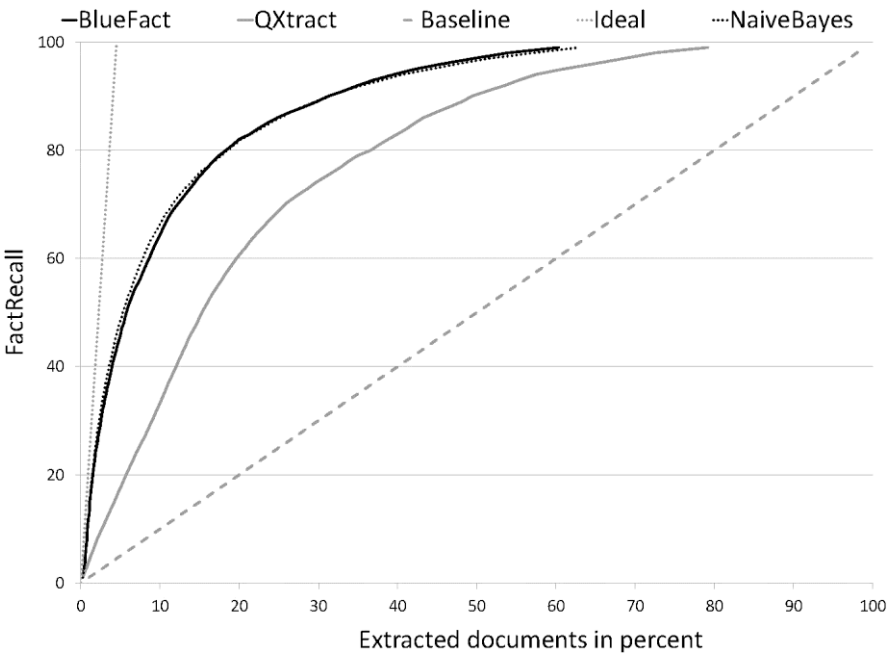


Figure 40: Fact recall for Reuters new corpus averaged over 20 fact types.

7.6.2 Evaluation Measurements

Each fact retrieval systems consumes computing resources, such as hard disc resources for document retrieval or CPU cycles for fact extraction as well as the score computation. For a fair comparison of different fact retrieval systems we measure the following execution costs:

- *Fired queries.* Each query causes costs for accessing the index. Therefore we measure the number of queries each systems poses against the index.
- *Extracted documents.* Each system causes significant extraction costs. Therefore we measure the amount of forwarded documents to the extractor. We compare the processed document fraction with the overall size of documents in the Reuters corpus.
- *FactRecall.* From each document the extractor might return relevant facts. For each system we measure the amount of returned

facts for processing the entire document collection. We define *FactRecall(n)* as the fraction of all facts X_{all} of a specific fact-type that have been extracted after processing n documents.

$$FactRecall_{system}(n) = \frac{|X_n \cap X_{all}|}{|X_{all}|}$$

- *Execution time.* For each system we measure the time for training, scoring, retrieving documents as well as for extracting and storing facts. Execution costs may vary for different document lengths or fact extractors. We sampled 1000 documents from our corpus and measured the absolute execution time to obtain a fair estimate of the time necessary for sending the document to the extraction service and extracting all facts from it.

7.6.3 Parameter Setup

Algorithm retrieveFacts requires two input parameters for the training phase. These parameters may vary for different search engine implementations: Parameter k limits the size of different keyword queries for each feature generation method and parameter n denotes the top-documents which the system retrieves for each generated keyword query from the index. For determining the optimal setting, we perform a grid search and systematically vary parameters between $1 \leqq k \leqq 200$ and $1 \leqq n \leqq 200$ and measure the mean average precision (MAP) over n returned documents for k different keyword queries:

$$MAP = \frac{\sum_{i=1}^{n} Average\ Precision\ (d_i)}{n}$$

We define the first document that is processed after our re-ranking or the seed-sampling phase in QXtract as rank 1 and sum over the entire recall curve.

Optimal parameters. From Figure 39 we observe that $k = 10$ different keyword queries per feature generation method are sufficient to retrieve nearly 98 % of relevant documents in the collection. Worse, if we increase k (thus we allow a wider spectrum of phrases to search for factual infor-

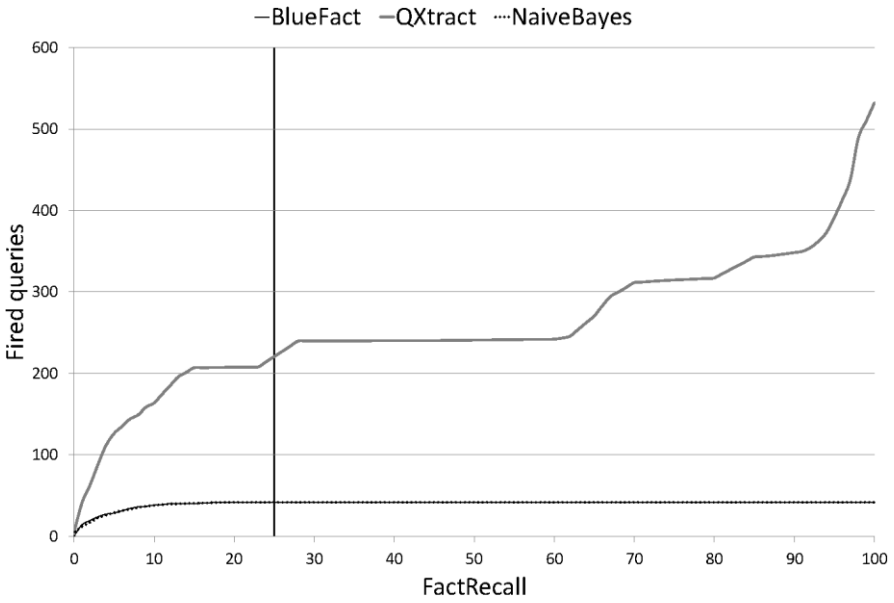


Figure 41: Number of queries fired against the search engine index from systems QXtract, BlueFact with a heuristic and BlueFact with a naive base scoring model. The graph of our two scoring models, BlueFact and Navive Bayes, overlaps. The vertical line marks the threshold of 25% fact recall as suggested by the authors of QXtract.

mation), we drastically decrease the amount of relevant documents that our search engine returns. For parameter n, which varies the amount of returned documents, we observe an optimal value for n = 50. We explain this behavior with the precision of the implemented document ranking strategy which returns in our setting most relevant documents among the top-50 documents.

For a fair comparison we equally set these parameters for all three systems, *Qxtract*, *BlueFact* and *Naive Bayes*. In addition we followed the recommendation of authors of QXtract which suggest five seed tuples to sample an initial training set of 100 documents. We utilized these parameters for systems BlueFact and Qxtract.

Experimental Results

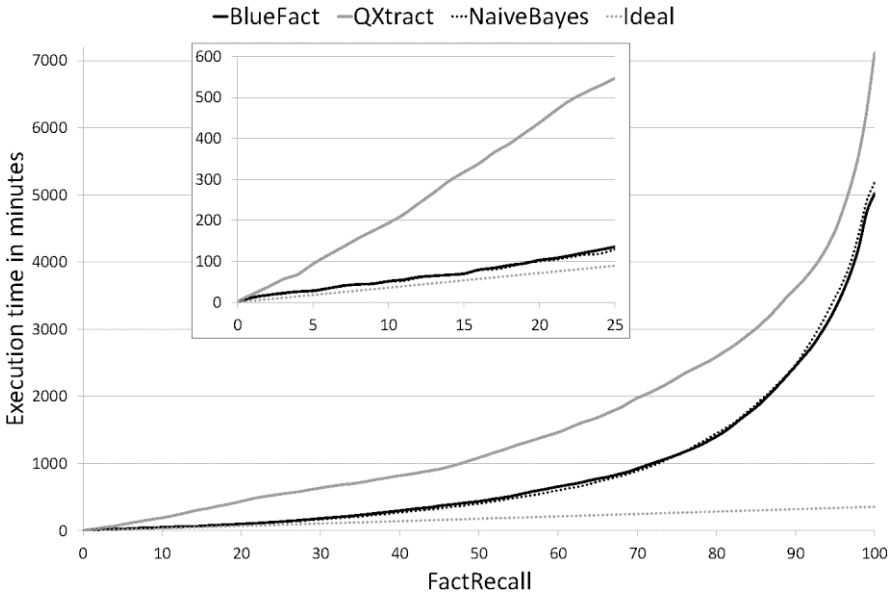


Figure 42: Execution time vs. fact recall for systems QXtract, BlueFact with a heuristic and BlueFact with a naive base scoring model. The small snapshot shows the first 25% of fact recall, where our system almost approaches the ideal execution time.

BlueFact's simple heuristic scoring model outperforms QXtract drastically in terms of fact retrieval performance, fires significantly less queries against the index, requires significantly less execution time, is robust for different domains and archives a high fact recall.

7.6.4 Comparative Analysis

BlueFact significantly outperforms other approaches. Figure 40 compares aggregated fact recall curves for 20 different fact types of systems QXtract, BlueFact with a heuristic scoring and BlueFact with the a Naive Bayes scoring against the baseline scenario and an ideal scenario. We observe that both BlueFact scoring models show almost identical performance. Both BlueFact scoring methods drastically outperform the system QXtract and show a steeper increase on the recall curve in Figure 40 than

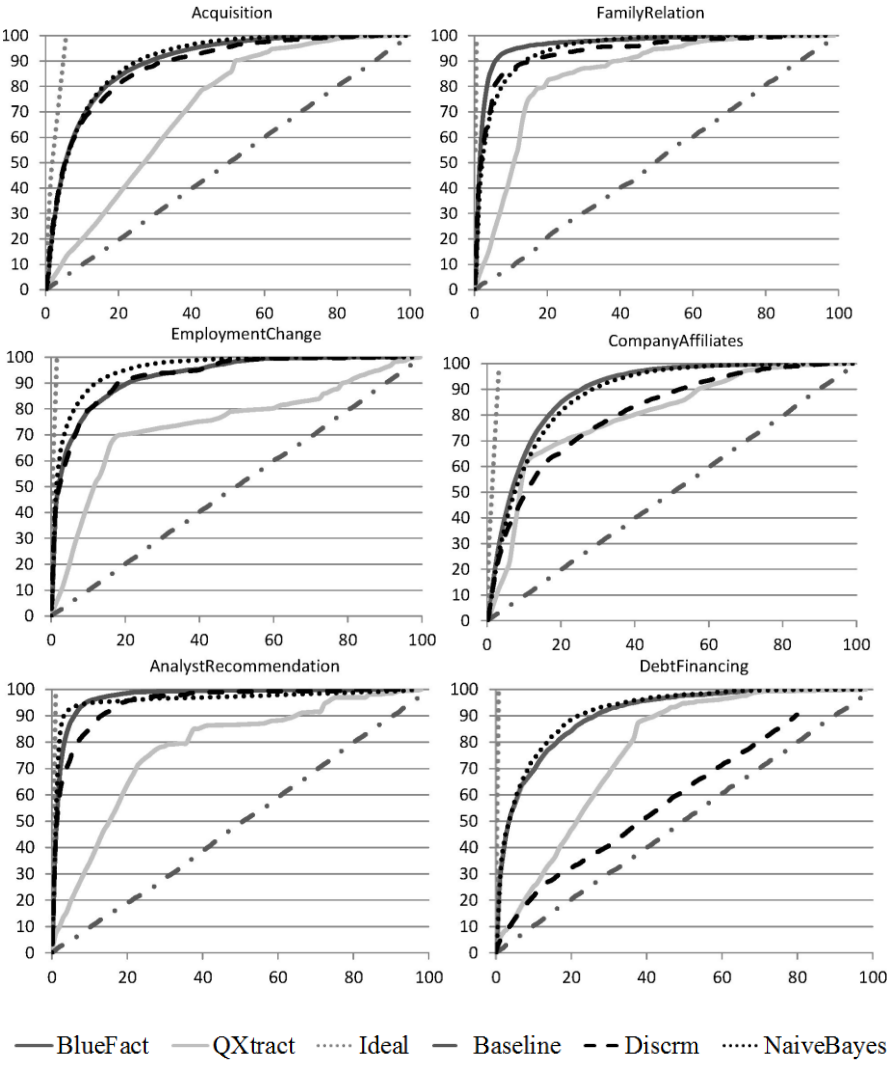


Figure 43: Evaluation results for six fact-types: The X-axis of the diagrams display the percentage of processed documents. The Y-axis shows the percentage of extracted facts. The system results lie between the ideal curve, which resembles perfect foresight where only relevant documents are forwarded to the extractor and the baseline where documents are forwarded in a random order. *Discrm* describes configuration of BlueFact that only uses lexico-syntatic features.

that of QXtract up to a recall level of about 70%. BlueFact only needs to forwards 4% of the document collection to retrieve 45% of available facts,

while QXtract processes 15% of the documents to reach the same fact recall. Likewise, after processing 10% of the document collection, BlueFact already reaches a recall of 70%, while QXtract achieves only 32%.

BlueFact reaches perfect recall with fewer documents. Blue-Fact scoring methods and QXtract discover nearly 100% of available facts on our collection (see Figure 40). However, BlueFact only processes 62% of the entire document collection to the fact extractor, while QXtract forwards more than 80%.

BlueFact radically limits queries to the search index. Figure 41 compares average query costs and fact recall. BlueFact queries the index only in the training and probing phase. In the scoring and collection phase, BlueFact scores collected document identifiers based on observed features and retrieves the highest scored documents first. Therefore, BlueFact no longer poses queries against the index after the initial phase which ends in Figure 41 after Blue-Fact reached 10% fact recall. This interesting feature becomes crucial when BlueFact is applied against a Web search engine with limited querying capacity. Contrary, QXtract relies on ranked document lists from the search engine and needs to continuously fire queries against the index. Worse, QXtract heavily utilizes generated keywords from the OKAPI generation method to reach a high fact recall. Unfortunately, this method tends to return many ambiguous keywords. As a result, index access costs for QXtract increase drastically when a high fact recall is required. Authors of QXtract suggest to overcome this shortcoming by only querying the system until a maximal fraction of 25% of the document collection has been retrieved, which is illustrated by a vertical line in Figure 41.

BlueFact significantly drops execution time. Figure 42 shows query execution times for the evaluated systems. We observe that both scoring models of BlueFact nearly overlap and significantly outperform QXtract. For instance, BlueFact requires 135 minutes for achieving 25% fact recall for our corpus and 20 fact types while QXtract requires more than 513 minutes for this task; we observe this trend to a lesser extend also for higher fact recall. Figure 42 also depicts the time of the '*ideal*' fact retrieval system, where only documents that contain facts are processed by the in-

Fact Type	BlueFact(Top 5)	Qxtract(Top 5)
Acquisition	acquisition of, sale of, to sell, purchase of, to acquire	share, offer, rival, trade, deal
Analyst Recommendation	coverage of, rating on, analyst, recommendation on, stock to	stock, bear stock, month stock, stock price, shares
Company Affiliates	employs, employees, workers in, people, jobs	company, percent strike, company united, minister company, industry united
Debt Financing	exchange commission, shelf, subordinated, to issue, securities	million, percent primary, issue Friday, million percent, million yield
Emplyment Change	named, executive, resigned, elected, become	broadcaster, executive, told, dollar conference
Family Relation	friend, estranged, former, children, married	former, house family, chief time, divorced, love

Figure 44: Top-5 keywords for fact-types of Figure 43.

formation extractor. In the upper left corner we show the snapshot for the first 25% of the facts contained in the corpus. In this interval both BlueFact scoring models only require slightly more execution time than the *ideal* extractor and are significantly faster than QXtract.

BlueFact is robust and efficient for different fact types. Figure 43 shows detailed results for six exemplary experiments for fact types from the domains person and company. Across all fact types BlueFact significantly outperforms QXtract. Even across domains *person* and *company*, BlueFact identifies relevant documents with high precision.

7.6.5 Detailed Discussion

BlueFact's scoring is independent from search engine scoring. BlueFact's retrieval model combines observed words and phrases into a single fact score. The score indicates how likely a document contains facts and is computed independently from the scoring model of the search engine. Next, BlueFact re-ranks document identifiers based on this score and finally processes the most promising documents first. Contrary, QXtract's scoring model only relies on the score from the search engine which may

feature generation method	**average F_1**
fact-type name	0.09
attribute values	0.39
unigrams	0.46
significant phrases	0.49
lexico-syntactic patterns	**0.59**

Figure 45: Average F1-Measure for each evaluated feature category. We computed the F1-Measure with precision and recall estimates after we queried the search index with top 10 feature candidates and retrieved top 50 documents.

be tuned for general retrieval tasks but not for fact retrieval.

Lexico-syntactic features. Recent research [73] shows that these features represent the most discriminative phrases in sentences that contain a fact. For instance, the lexico-syntactic classifier described in Chapter 6.3 extracts compound phrases that represent predicates in English sentences, such as *"sale of"*, *"workers in"* or *"to issue"* (see also Figure 44).

MAP@Rank	BlueFact	Qxtract	BlueFact/Qxtract
1	**0.37**	**0.21**	**1.76**
2	**0.34**	**0.18**	**1.89**
3	**0.26**	**0.13**	**2.00**
4	**0.22**	**0.11**	**2.00**
5	**0.22**	**0.10**	**2.20**
6	**0.22**	**0.10**	**2.20**
7	**0.21**	**0.09**	**2.33**
8	**0.21**	**0.09**	**2.33**
9	**0.20**	**0.08**	**2.50**
10	**0.20**	**0.08**	**2.50**
20	**0.17**	**0.06**	**2.83**
50	**0.15**	**0.05**	**2.00**
100	**0.11**	**0.05**	**2.20**
200	**0.10**	**0.05**	**2.00**
500	**0.07**	**0.05**	**1.40**
1000	**0.05**	**0.05**	**1.00**

Figure 46: Mean average precision (MAP) for BlueFact with a heuristic scoring and for QXtract. Typical search engines limit access to the first 1000 pages per search result, this limitation is reflected in our MAP scores.

Consider again Figure 43, the curve labeled *Discrm* shows Blue-Fact using only these lexico-syntactic patterns as a feature generation method. We observe that lexico-syntactic patterns are quite effective in retrieving documents that are likely to contain a fact. However, for fact types *CompanyAffiliates* and *DebtFinancing* these keywords perform poorly and do not suffice to reach all relevant documents in the collection.

Complementary feature extraction methods. BlueFact reaches potentially missing, relevant documents through keywords that are generated with additional methods. Figure 45 lists the average F1 measure for each feature generation method. For each feature generation method we observed precision and recall estimates after the top k feature candidates retrieved top n documents in *Phase I* of our *retrieveFacts* algorithm. Our results in Figure

45 show that lexico-syntactic patterns, significant phrases, unigrams and attribute values are effective to retrieve documents that are likely to contain a fact.

Consider the fact type *DebtFinancing* in and the generated keywords in Figure 44. With the help of lexico-syntactic patterns, BlueFact generated the keywords *to issue* and *commission on*. As the curve DSCRM in Figure 43 illustrates, these phrases alone do not suffice to reach all relevant documents in the collection. BlueFact complements these keywords with frequent unigrams, such as *holdings* or *treasuries*, attribute values, such as *exchange commission* and significant phrases, such as *treasury bill*. Together, all keyword generation methods allow BlueFact to retrieve missing documents from the 'long tail', in particular where a *DebtFinancing* fact is expressed with phrases that BlueFact could not capture with the lexico-syntactic patterns alone. As a result BlueFact reaches 100% fact recall for this fact type. Contrary, QXtract is limited limited to the statistical keyword generation methods from Ripper, SVM and OKAPI. This limitation prevents QXtract from extracting compound phrase structures as keywords.

BlueFact's iterative sampling strategy is effective. Because of the additional feature types, BlueFact observes a richer set of potential keywords per processed document. Because of this additional information, BlueFact generates a wider spectrum of phrases and is able to sample more relevant documents for succeeding feature extracting iterations. Contrary, the training sample of QXtract is based on user defined seed keywords only. Depending on the 'representative power' expressiveness of these seed keywords for a particular fact type, the initial document sample may contain very few relevant documents. Consequently, QXtract cannot observe a sufficiently focused set of highly discriminative keywords with the same size of seed documents. Our results confirm this intuition: Compared to BlueFact, costs for processing the entire document collection are almost twice as high for QXtract (see Figure 41 and Figure 42).

7.6.6 Applications and Limitations

We apply BlueFact to an ad-hoc query and an archive scenario. In an *ad-hoc query scenario* we consider an analyst who formulates structured que-

ries against an indexed document collection. This interactive scenario requires low execution costs for scoring, for retrieving and for extracting facts from high ranked documents (sees also the vision in Chapter 5 and applications in Chapters 6, 8 and 10). An important criterion in such a scenario is the mean average precision (MAP) that measures how many of the top ranked results include a relevant document averaged across multiple queries.

Figure **46** compares the MAP score of BlueFact and QXtract. BlueFact archives up to a 2x higher MAP score for top-200 documents and a 3x higher MAP score for top-50 documents. Hence, BlueFact enables this important scenario through comparable high MAP scores for high ranked documents, low index access costs and the applicability for multiple domains.

In an *archive scenario* we consider a digital archive provider that extracts and collects in an off line step a comprehensive fact base. In this scenario a 100% fact recall is crucial, while extraction cost should remain as low as possible. Figure 42 shows that BlueFact archives 100% fact recall but still requires fewer queries than QXtract. Limitations. BlueFact relies on a standard inverted index that is based on the bag-of-words model. Typically, these index structures do neither provide containment information for logical document structures, such as sentences, paragraphs nor semantic structures for potential attributes values (see [111]). In our future work we will investigate such semantical enhanced index structures to improve BlueFacts feature generation and document scoring model.

7.7 Related Work

We discuss relevant related work in the areas of focused fact retrieval from full text indices and open information extraction.

Keyword query generation with QXtract. The authors of QXtract [71] pioneered the work on automatically generating keyword phrases for fact retrieval from full-text indices. The system exe cutes the following stages in a one time learning process: sample seed documents, observe/score phrases in these documents, query a search engine with the most promising phrases and forward the retrieved pages to an information extractor.

Sample seed documents. QXtract requires a small set of manually specified 'seed' facts to retrieve an initial set of sample pages for training. The size of this sample influences the precision of the learning process; the authors suggest between 100 and 2500 pages. After sampling, the pages are forwarded to the extractor to identify which documents are relevant for the extraction task.

Observe and score phrases. QXtract utilizes three classification approaches for observing and scoring unigram terms from the documents in the seed sample: An SVM-based classifier, the OKAPI system [112] and the rule-based classifier Ripper [113] are trained on the set of seed documents. Each approach returns a list of terms, ordered by their significance for classifying relevant and irrelevant documents. The authors of QXtract propose a threshold based technique to assemble and select relevant phrases as keyword queries out of these terms.

Determine document retrieval order. Next, QXtract retrieves a ranked set of documents from the index for each generated keyword query, where the queries are selected from the three lists in a round robin fashion. All documents retrieved for a phrase are forwarded to the fact extractor. The process continues with selecting the next highest ranked phrases from each list and forwarding the resulting documents to the extraction service. It terminates once no more phrases exist in any list. The authors evaluated their approach on the two fact extractors *CompanyHeadquarter* and *DiseaseOutbreak* and achieved a recall of about 20% while processing 10% of the documents in the test corpus. In [88] the authors describe a framework incorporating the idea of automatic keyword generation as an execution strategy.

Phrase generation from SQL queries. Authors of [76] extract and rank potential keywords from attribute names and values of a structured query. They apply a greedy approach that selects terms based on their relevance measures informativeness and representativeness. Our approach makes use of a similar idea: We trust that Web page authors developed a set of common words and grammatical structures to express important factual information in natural language.

Self-supervised keyword generation. In the last chapter we published an approach that applies a self-supervised keyword generation method to extract meaningful phrases which often correlate with factual information in sentences. This approach utilizes open information extraction techniques to generalize fact-type-independent lexico-syntactic patterns from sentences. In this paper we use these patterns as one potential feature generation type. Our extensive experimental evaluation on 20 fact types shows, that lexico-syntatic pattern significantly help to retrieve relevant pages. However, our experiments also discovered that additional feature types are necessary for a robust domain independent keyword generation.

In addition to our work in the last chapter we presented a coherent theoretical reflection of the inference problem as well as an evaluation of the execution time and execution costs of the system and an analysis of the sensitivity of the parameters on the final result.

Fact extraction. Multiple projects, such as CIMPLE[70], AVATAR [60], DIAL [79], DoCQS [111], PurpleSox [114] describe an information extraction plan with an abstract, predicate-based rule language. Our approach tries to ’re-engineer’ common patterns from observing fact extractors and other patterns that appear frequently with factual information on a page as features.

7.8 Conclusion

It is a fact that today only a few companies have the capacity to aggregate the wealth of information hidden in large document collections such as the Web. Exploiting this information with structured queries is an often requested, but still unsolved requirement of many Web users. Ideally, information extraction systems leverage information from an existing indexed document collection to process only relevant documents that are likely to contain a fact. Unfortunately, manually formulating and updating such keyword queries is difficult and time consuming.

We presented the BlueFact framework, which applies machine learning techniques to generate keyword queries for systematically retrieving all facts in an indexed document collection. In a comparative evaluation for 20 different fact-types over a standard Reuters News corpus we show that

BlueFact' s scoring models drastically outperform QXtract and other approaches in terms of fact retrieval performance, fire significantly fewer queries against the index, require significantly less execution time and achieve very high fact recall across domains. We also discuss how to apply Blue-Fact to an ad-hoc query as well as to an archive scenario.

BlueFact is open for additional feature types and different fact scoring models. Another orthogonal direction of work is the design of supervised classifier (see Chapter 9) that filters out irrelevant pages. Potentially, one could implement operators for retrieving complete factual information which we will introduce in the next Chapter.

8 Self-Supervised Search for Any-k Complete Tuples

In this chapter we propose a novel query processor for systematically discovering any-k relations from Web search results with conjunctive queries. *The 'any-k' phrase* denotes that retrieved tuples are not ranked by the system. For realizing this interesting scenario the query processor transfers a structured query into keyword queries that are submitted to a search engine, forwards search results to relation extractors, and then combines relations into result tuples. Unfortunately, relation extractors may fail to return a relation for a result tuple. We propose a solid information theory-based approach for retrieving missing attribute values of partially retrieved relations. Moreover, user-defined data sources may not return at least k complete result tuples. To solve this problem, we extend the Eddy query processing mechanism [85] for a situational business intelligence scenario with a continuous, adaptive routing model. The model determines the most promising next incomplete row for returning any-k complete result tuples at any point during the query execution process.

We report a thorough experimental evaluation over multiple relation extractors. Our experiments demonstrate that our query processor returns complete result tuples while processing only very few Web pages.

The content and ideas of this chapter are published in [125] [127].

8.1 Introduction

A typical task of a Web user is discovering relations between entities from Web pages; i.e. consider the following query:

Extract and join relations between products, acquisitions and employee sizes of companies from top 100 pages of CNN.com as seed data into re-

Joining 2 Relationships (Information about Persons)

2-06: Attribute(Person,Born,Gender)
⋈ Career(Person,Company, Position); w*ashingtonpost.com(100); yahoo(10)*

2-07: Conviction(Person, Charge)
⋈ Career(Person,Company, Position); *cnn.com(100); yahoo(10)*

Joining > 2 Relationships (Information about Companies and Persons)

3-01: Product(Company,Product) ⋈ Career(Person,Company, Position)
⋈ Attribute(Person,B_Place, B_Date); money.*cnn.com(100); yahoo(10)*

3-08: Recommendation(Analyst,Company, Trend,Date)
⋈ Bankruptcy(Company, Status, Date)
⋈ Product(Company,Product); *ft.com(100); yahoo(10)*

4-01: HQ(Company,Location) ⋈ Employees(Company,EmployeeSize)
⋈ Product(Company, Product) ⋈ Bankruptcy(Company,Status,Date);
cnn.com(100); yahoo(10)

5-01: Born(Person,Born) ⋈ Conviction(Person,Conviction)
⋈ Employer(Person,Company) ⋈ Travel(Person,Location)
⋈ Career(Person,Company, Position); *nytimes.com(100); yahoo(10)*

Figure 47: Benchmark queries for two and more relation extractors

sult tuples. Retrieve complementary pages from Yahoo.com until 500 complete result tuples exist.

This query might be issued by an analyst who is observing the market for interesting acquisition options. The query intention is retrieving *at least 500 tuples* about any company, its products, its headquarters, and its employee information from Web pages.

This query might be issued by an analyst who is observing the market for interesting acquisition options. The query intention is retrieving at least 500 tuples about any company, its products, its headquarters, and its em-

ployee information from Web pages. Each day new pages emerge in the Web that may contain a textual representation of facts for answering these questions. Strategic decision makers may frequently research the Web for questions like these. Often answers to these queries might not be published in textual or structured form by Web 'information aggregators' like Wikipedia.com, Freebase.com, Trueknowledge.com or Yago [66]. Rather, this rare factual information is hidden in unstructured Web text on a few Web pages of news agencies or blogs. Unfortunately, collecting factual answers from these pages with a general Web search engine is still a dreaded process for a user.

One option to populate a fact base is to crawl a large document collection. For instance Google Squared [103] populates its data base with facts from the large corpus of the general Web. The system extracts these facts from tables, lists and from text with open information techniques. However, in Chapter 7 and in [126] we observed that only a small fraction of a large archive de facto contains factual information. Hence, strategies that might execute a full scan over the entire archive can drastically waste processing and storage resources (see also [88]).

Another option is discovering facts in retrieved pages from ad-hoc keyword search. Unfortunately, this is still a tedious task, since Web search engines do not return aggregated factual information. Due to the nature of the business models of commercial Web search engines, it is unlikely that full access to internal databases will ever be granted to individual users. Typically only the top 1000 results for a given search query can be retrieved. The heuristic of a search engine user is: type in keywords as queries, 'extract' relevant facts from the top-k documents, filter out relevant facts and compile facts into a structured fact table. Therefore the user typically repeats this process multiple times in order to complement missing attribute values and to enhance the chance to discover unseen facts. We call this explorative process discovering any-k tuples, where tuples are equivalent to rows in the spreadsheet.

Discovering any-k tuples through Web search. Our system automatically fulfills the task of retrieving a set of any-k complete result tuples. Each query returns a table with attributes from each relation. The query proces-

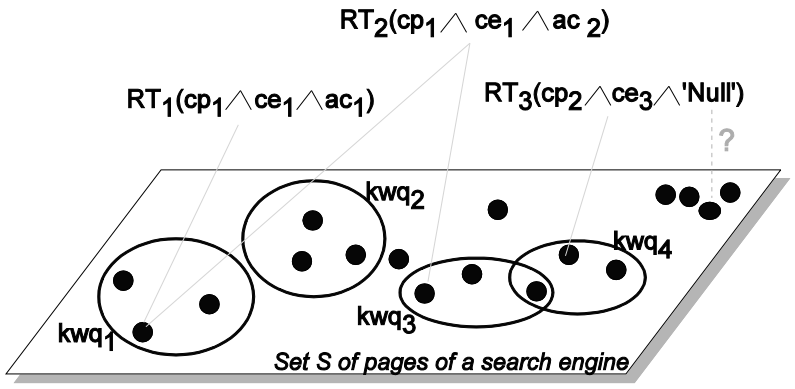


Figure 48: Consider four keyword queries kwq1…kwq4 that retrieve tuples for answering the example query Q 3-01. Each keyword query returns a set of three result pages; these pages are forwarded to an extractor that returns tuples cp∈CompanyProduct, ce∈CompanyEmployees and ac∈Acquisition. For instance, kwq1 returns three pages from which the query processor extracts tuples cp1, ce1, ac1 and combines them into result tuple RT1. Query kwq2 does not return any relevant pages. Then, the query processor issues query kwq3, extracts tuple ac2 from retrieved pages and combines tuple ac2 and previously retrieved tuples cp1, ce1 into result tuple RT2. Query kwq4 returns tuple cp2 and ce3 which are combined by the query processor into incomplete result tuple RT3.

sor populates the table with initial result tuples that are extracted from top-100 pages of primary source (such as ft.com). Next, the query processor queries a complementary source (such as the search engine Yahoo) for retrieving missing attribute values. Thereby, the query processor makes predictions on the likelihood that specific pieces of missing information from incomplete tuples can be found at the complementary Web source and, based on that, optimizes the sequence of Web requests that need to be made in order to obtain k complete result tuples. This set may then be forwarded directly to the user or may be forwarded to an application for specific filtering [72], [31] or ranking [66] approaches.

Our contributions: The long term vision of our work is to answer structured queries from natural language text on Web pages, called Web text.

For realizing this novel application scenario we give in this Chapter three core contributions:

(1) *End-to-end Web query processor:* We propose a *query processor* to systematically discover any-k complete result tuples from Web pages. In this chapter we focus on select-project-(full outer) join queries that integrate tuples from two or more relation extractors. Our query processor leverages the index of a Web search engine for retrieving tuples from potentially billions of pages.

(2) *Solid theoretical framework ensures any-k complete tuples:* We view completeness as a major measure of data quality and thus aim to return any-k complete result tuples. For solving this very practical problem we propose a solid information theory-based approach for identifying the most promising source for retrieving missing attribute values of partially retrieved tuples.

(3) *Novel extension to the Eddy operator for Web querying.* We design our model as a novel extension of the Eddy operator for the Web querying domain. An Eddy is a dataflow operator in traditional a database system that allows an adaptive tree reordering in a tuple by tuple fashion. Our extension implements a continuous, adaptive routing model for determining the most promising next Web source to query at any point during the execution. We implement our query processor for the in-memory database *HSQLDB* [77]. In a broad study we test our theoretical findings on multiple relation extractors and identify optimization strategies that effectively reduce the amount of processed pages.

Overview: In the following sections we outline our query processing algorithms for any-k complete tuples in detail (Section 8.2). Then we report on the performance of our algorithms (Section 8.3) and discuss related work (Section 8.4). Finally, we summarize our contributions (Section 8.5).

8.2 Query Processor

In this section, we present a framework for the study of the tuple discovering problem. In Section 8.2.1, we abstract important problems for automating the discovery of complete tuples on the Web. Based on this interaction

model, we present a generic query routing algorithm for an adaptive query processor in Section 8.2.2. Finally, in Sections 8.2.3-8.2.5, we discuss routing policies that can ensure complete result tuples and significantly reduce query processing costs.

8.2.1 Problem Analysis

Theoretically, the problem of discovering any-k result tuples on Web search results can be formalized as follows: We assume a query processor is downloading a subset of pages from a set of pages S indexed by the search engine (see the rectangle in Figure 48). We represent each Web page in S as a point (dots in Figure 48). Every potential query kwq_i from the query processor returns a subset of S. From each subset of S we may be able to extract none, one or multiple relations. The query processor joins these relations from one or multiple subsets into result tuples RT_k. Unfortunately; some result tuples may be missing certain attribute values that could not be extracted (denoted by 'NULL'). However, the set of pages S may contain additional pages (denoted by a dotted line) from which the query processor could extract these missing values.

Ordered sequence of keyword queries: Our goal is to find an ordered sequence of keyword-queries which likely return pages from which the query processor can discover any-k result tuples. Extracting relations from pages is costly and time consuming. Therefore we would like to minimize the cost by processing as few pages as possible.

NP-complete problem: Determining an ordered sequence of keyword queries that minimizes the costs for discovering any-k result tuples is equivalent to the problem of answering a query against a schema using a set of views. This problem is NP-complete for conjunctive queries and conjunctive view definitions [84].

How should a query processor determine an ordered set of keyword queries? In practice, we face the following difficulties that we need to address to solve the formalization from above:

Continuous, adaptive routing model: Query optimizers in current database systems are designed to pick a single efficient execution plan for a

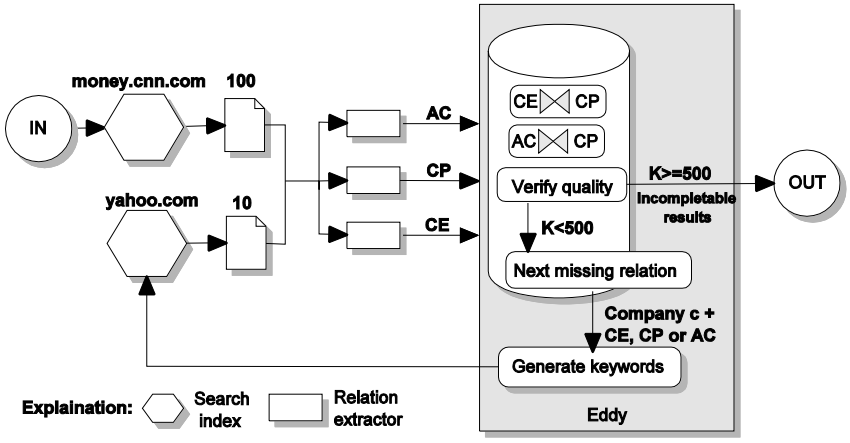


Figure 49: We illustrate the quality based routing policy for query 3-01. First, the system retrieves 100 initial pages from money.cnn.com, forwards pages to relation extractors AC, CP and CE that extract and return relations from these pages. The Eddy joins relations and determines if 500 complete result tuples exist. Next, the Eddy generates a keyword query for retrieving a missing relation. The keyword query retrieves 10 pages from complementary search engine 'yahoo.com'. The Eddy continues this process until 500 or more result tuple are retrieved or of no more result tuple can be completed.

given query based on statistical properties of the data. However, in our setting the query processor transforms a structured query into a set of keyword queries to extract an initial set of result tuples. Next the query processor chooses an incomplete result tuple and generates a new keyword query to retrieve the missing attribute values. This adaptive process of generating keyword queries, extracting tuples and joining them into result tuples must be continued by the query processor until at least k result tuples are complete.

Estimate how likely a keyword query returns a missing attribute value: For optimizing query execution costs the query processor must estimate how likely a keyword query and an extractor will return a missing attribute value for a particular result tuple. The probability that a keyword query returns a missing attribute value is correlated with the search engine rank of the source Web page, the content of that particular page, the over-

all set of indexed pages of the search engine and the mechanics of the extractor.

- *Biased search engine ranking.* The ranking mechanism of a search engine can deny a page from appearing within the top pages [68]. Furthermore, search engines like Yahoo or Google limit the access to their database to top-1000 pages per keyword query.
- *Insufficient content.* There may be relations for which no textual content exists on the Web. Consider a query joining two relations of type *CompanyProduct* and *Bankruptcy*. In reality, every company has at least one product, but only a few have declared bankruptcy, thus these particular result tuples can never be completed.
- *Failing extractors.* Relation extractors can return incomplete tuples. The rationale for this incompleteness is to indicate the existence of a relation on a Web page to human evaluators, even though precise values for each attribute of the relation could not be extracted. For instance, relation extractors may fail because of a *vocabulary mismatch* between the Web page and the rules of the extractor. Object reconciliation [82] may fail if not enough context information for resolving objects exist.

In the next Sections we present our solution for each requirement.

8.2.2 Generic Routing Algorithm

Our generic routing algorithm is implemented as an extension to an Eddy [85] which can potentially adapt at the granularity of a missing relation. An Eddy processes a query by routing relations through operators that are specific to that query. However, an Eddy does not actively request missing relations to meet user defined quality requirements, such as returning complete result tuples. Therefore we extend Eddies with a technique called *quality based routing (QBR)* that adds quality requirements into routing decisions. With QBR an Eddy automatically identifies tuples of insufficient quality and generates additional queries for obtaining missing relations. QBR is low-overhead and it is adaptive, revisiting its decisions as changes in result tuples are detected.

The following paragraphs describe this continuous process.

Obtain initial set of pages. First, the query processor bootstraps top-k pages from the data source that is specified in the INITIAL_DATASOURCE clause in the query. If the data source does not return any page, query processing is terminated. Otherwise the query processor proceeds with step 1.

Step 1: Generate result tuples. In this step the query processor forwards pages to relation extractors and combines new tuples with potentially existing result tuples.

- *Extract tuples from pages.* The query processor forwards pages to relation extractors. For each page none, one or multiple tuples are extracted.
- *Combine new tuples with existing result tuples.* The query processor enables n-way full outer joins for combining new tuples into result tuples. If k is relatively small (such as in our scenario) this operation can be executed in memory. For relatively large result tuple sets (such as millions of result tuples), the join-operation can be optimized with existing adaptive query processing techniques, such as progressive optimizations [86] or lottery scheduling [85].
- *Verify quality and terminate query if necessary.* The full outer join may create new result tuples or can add new attribute values to existing result tuples. Therefore the query routing policy monitors the quality of result tuples. The query processing is terminated either if no more result tuples can be completed or if any-k result tuples meet the required quality requirements (see Section 8.2.5).

  Otherwise, the query processor proceeds with step 2.

Step 2: Retrieve missing attribute values. In this step a single missing attribute value is identified. The query processor generates an additional keyword query for obtaining additional pages that likely contain the missing values.

- *Identify next missing attribute value.* The query routing policy identifies a candidate result tuple that does not meet the quality requirements of the query. From the candidate result tuple a missing attribute value is identified that can be likely obtained. (See sections 8.2.3-8.2.5).

- *Generate keyword query.* We utilize the keyword generation strategy from chapter 6. First, the strategy observes if sentences which share instances of the same relation also likely share similarities in textual content. Next, we only keep phrases from these sentences that indicate the existence of a relation using an efficient lexico–syntactic classifier [73]. Finally, we determine for each phrase their clarity with respect to other relations and their significance for a particular one [78]. For instance, for retrieving missing attributes *B_PLACE* and *B_DATE* for the tuple *ATTRIBUTE('Martin Winterkorn', Null, Null)* our keyword generation strategy outputs the keyword query *+'Martin Winterkorn' +('born' OR 'born in' OR 'born at' OR 'native of').*
- The query processor executes the keyword query against the search engine from the COMPLEMENTARY_ DATASOURCE clause and proceeds with step 1 again.

In the reminder of this section we propose multiple routing policies to customize our generic routing algorithm.

8.2.3 Baseline: Zig-Zag Routing Policy

The Zig-Zag routing policy [69] selects a random result tuple that misses an attribute value. Despite its simplicity, a Zig-Zag routing policy can cause relatively high costs. For instance, a Zig-Zag routing policy is unaware of the quality of result tuples. Therefore this policy may process result tuples that only achieved a low degree of quality and at the same time may ignore result tuples for which it would be rather easy to complete them. Hence, this policy frequently picks a result tuple that cannot be completed with additional information from the Web. Finally, this policy can create a death spiral by repeatedly selecting and creating new low quality result tuples.

Therefore we consider the Zig-Zag routing policy as a baseline scenario since it is expected to perform sub-optimally.

8.2.4 Choosing 'good' join attributes

In this section we propose our novel approach for estimating the likelihood of returning a missing tuple. Our idea is to compute the information gain ratio (IGR) for each combination of a relationship-type and a join predicate

that can be potentially used by the Eddy routing algorithm. The IGR gives an estimate whether a particular combination will return a relation or not.

Identifying join attributes: The query processor explores correlations between values of an attribute A_E from which a keyword query is generated and values of an attribute $A_1,...,A_n$ of a relation $R_E(A_1,...A_n)$ that is returned from extractor E. In our setting the observed correlation between two attribute value distributions might not reflect the 'real' distribution that could be observed on a much larger sample of Web pages. We reflect this uncertainty with a specification from Information Theory which is based on the concept of *gain ratio* [87], described next. The IGR is typically used in decision-tree learning algorithms (such as ID3 [87]) to determine the attribute that best classifies a given data set. We utilize the IGR for identifying the most promising join attribute and the most promising relationship-type for retrieving missing tuples.

Definition 1 (Entropy): *Let $c_1...c_p \in D.C$ be instances of attribute C of relationship-type D(C,B) where the attribute values c are different from NULL. We use the entropy [*87] *of D.C, denoted by H(D.C), which is an information-theoretic metric, to capture the information content of D.C. H(D.C) is defined as:*

$$H(D.C) = -\sum_{c \in c_1...c_p} \frac{|D.C = c|}{|D.C|} * \log_2 \frac{|D.C = c|}{|D.C|}$$

where $c_1...c_p \in D.C$ are part of previously extracted instances of the relationship-type D.C. These instances where extracted using n keyword queries created from the attribute values $b_1...b_n \in D.B$

Definition 2 (Gain ratio): *Let D.B be an attribute that is used to generate keywords for retrieving additional pages from the search engine. Let $b_1...b_n$ be the distinct values of D.B. Following* [87]*, we define as Information Gain the increase in information about pages containing the attribute $D.C$, gained by the additional knowledge about values of attribute $D.C$:*

$$IG(D.C, D.B) = H(D.C) - \sum_{b \in b_1 \ldots b_n} \frac{|D.B = b|}{|D.B|} * H(D.C|D.B = b)$$

Following [87]*, we normalize the information gain to get the Gain ratio.*

$$Gain\ ratio(D.C, D.B) = \frac{IG(D.C, D.B)}{H(D.B)}$$

8.2.5 Completeness-based Routing Policy

Overview: To overcome the shortcomings of the Zig-Zag policy, we propose a new greedy routing policy called *completeness based routing policy (CBRP).* A CBRP returns any-k complete result tuples. When a CBRP is chosen as routing policy, only those result tuples which already achieved a high degree of completeness are chosen for further processing. For each of these result tuples, the *likelihood* of retrieving a missing attribute is estimated with the *Gain ratio* and incorporated into the selection decision. Finally, keyword queries are only generated for result tuples that *de facto* can be completed. Thereby, fewer pages need to be processed by relation extractors and the overall system throughput is significantly higher compared to that of a Zig-Zag policy. We implement these requirements as follows:

Step 1: Avoid result tuples that cannot be completed: A keyword query may return irrelevant pages on which an extractor cannot spot a missing relation for a result tuple. In that case, we assume that the missing relation is either not available on the Web or cannot be captured by our system. Therefore, we label the result tuple as incompletable and exclude it from the query routing process. If a subsequent keyword query happens to return a Web page from which a missing relation for the result tuple in question can be successfully extracted, we retract the label and include the result tuple into the routing process again.

Step 2: Maximize completeness: If a result tuple already achieved a high degree of completeness only a few missing attribute values have to be retrieved from the Web to complete it. Therefore our routing policy prefers result tuples that have achieved a maximum degree of completeness. We define result tuple completeness with respect to a query Q as follows:

Definition 3 (Completeness): *Consider a set of attributes A_Q of a query Q. Function fullness(a) returns 0 if attribute $a \in A_Q$ contains a NULL-value (if*

no value has been extracted for an attribute) and otherwise returns 1. We define 'completeness$_Q$' of a result tuple t as the average fullness over all attributes A_Q of a query Q.

$$completness_Q(t) := \frac{1}{|A_Q|} \sum_{a \in A_Q} fullness(a)$$

If a result tuple t reaches a completeness$_Q$(t)=1, the result tuple is 'complete', otherwise the result tuple is 'incomplete' .

Step 3: Maximize Gain ratio: Multiple incomplete result tuples may achieve the same maximum degree of completeness. For these candidate result tuples the query processor enumerates combinations of join attributes and relation extractors. Then, the query processor selects the combination which achieves the maximum Gain ratio to compute the next keyword query for retrieving a missing relation. The maximum gain ratio is selected, because it indicates the combination of a relation extractor and join attribute that most likely returns a missing relation.

Step 4: Continuously update classifier: Each time new pages are returned the query processor updates the Gain ratio. The query processor takes existing result tuples as training data and re-computes the Gain ratio for all combinations of join predicates and relation extractors. The *Gain ratio* for a particular combination of join attributes and relation extractors is high, when the keyword queries computed from the join attribute values return pages from which a relation extractor can easily extract missing attribute values for a relation. Optimizing this process is subject for our future work.

8.3 Experimental Evaluation

We developed a prototype system for discovering any-k complete tuples. In this Section we report our results. Finally, we discuss alternative, quality-driven indicators for our query processor.

8.3.1 Evaluation setup, queries and metrics

Currently no benchmark for discovering any-k complete tuples from Web pages exists. Therefore we conducted experiments on six queries shown in

Figure 47. Our benchmark is tailored towards discovering any-k complete tuples from search engines with a special focus on news Web sites. We expect that Web users frequently visit these pages to discover previously unseen connections between companies and persons. Therefore our queries simulate the human process of reading news pages, discovering initial result tuples from news pages and complementing missing tuples from additional pages. Each query obtains 100 initial pages from various news search engines, such as cnn.com, ft.com, washingtonpost.com or nytimes.com. We expect that news Web pages cover only a fraction of the information that is published on the Web. Therefore, each query complements missing tuples from the index of the *Yahoo search service* [74].

The benchmark queries utilize 11 relation extractors with 14 different attribute types from the extraction service OpenCalais.com [75]. Queries vary in the number of relation extractors, the number of attributes and in join predicates.

Measurements: Our goal is reducing the number of processed pages. For each query we measure the average number of processed pages to retrieve a single complete result tuple (P/RT). In addition, for each query we measure the number of complete tuples (RT) after the system processed 5000 pages. We also measure the proportion of the size of any tuples (Buf) vs. complete result tuples that answer the query (%RT/Buf). Finally, we compared the speedup of each strategy to our base line.

Q-ID:	Query ID
RT:	Complete result tuples per query
Incompl:	Incomplete tuples in buffer
Usage:	Percentage of complete tuples in buffer
Docs:	Processed documents until (k=500)
Docs/RT:	Processed pages per single complete result tuple
Var(Docs/RT):	Variance of *Docs/RT* across 6 experiments per query
Speedup:	P/RT (Baseline) / P/RT (competitor strategy)
Status:	Query returned k result tuples with 5.000 pages?

Setup: We implemented routing policies (see Section 3) and utilized the keyword generation strategy from Section 4 for the java-based in-memory

database HSQLDB [20]. Experiments are conducted on a T61 laptop (Windows 7) with 4 GB of RAM and a core duo CPU with 2.2 GHz.

Evaluated strategies: The most interesting comparative experiments are concerned with different keyword generation strategies and different routing policies for the same query**.**

UK-RND (Expert keywords, Zig-Zag policy): This strategy is our baseline. The strategy utilizes Zig-Zag joins [24] and user-defined keywords (see Table XXX).

DK-RND (Generated Keywords, Zig-Zag policy): This strategy also utilizes Zig-Zag Joins [68] and generated keywords from Section 6.

IGAIN (Generated Keywords, CBRP): This strategy implements the *completeness based routing policy* from Section 3 and utilizes generated keywords from Section 6.

Parameter: We set k=500 (minimal complete result tuple size) and n=10 (top pages). The query execution is terminated either, if *k* is reached or if 5.000 pages are processed.

Validity of experiments: Web search engines might return non-deterministic results even for same keyword queries. Therefore we executed each experiment 6 times. The column Var(Docs/RT) in Table shows the variant of the MAP is an indicator for potential varying search results.

8.3.2 Evaluation results

The following Figures show our results which we obtain the following interesting observations.

Strategy IGAIN is a clear winner. This strategy returns any-k complete result tuples for our queries. Contrary, the Zig-Zag policy is significantly less successful. Strategy DK-RND failed to return at least k=500 tuples for four queries, 2-07, 3-01, 3-08, and 5-01 and strategy UK-RND failed for three queries, 2-07, 3-01 and 3-08. These measurements show that a completeness based routing policy ensures sufficient result tuples.

Q-ID	Strategy	RT	incompl.	usage	docs	docs/RT	Var(docs/RT)	Speedup	Time
2-06	igain	509	7.527	6,76%	2.187	4,29	1,25	**1,14**	4.747
	uk-rand	506	11.336	4,47%	2.476	4,89	0,15	1,00	5.900
	dk-rand	515	12.515	4,12%	2.422	4,70	1,61	1,04	5.862
2-07	igain	518	1.925	26,92%	1.074	2,07	0,43	**4,67**	983
	uk-rand	418	9.739	4,29%	4.047	9,69	38,48	1,00	8.998
	dk-rand	454	13.462	3,37%	4.724	10,41	5,05	0,93	11.652
3-01	igain	565	4.485	12,60%	1.648	2,92	12,66	**5,18**	2.049
	uk-rand	300	21.219	1,41%	4.521	15,09	111,96	1,00	11.999
	dk-rand	377	20.848	1,81%	3.904	10,37	610,51	1,46	11.343
3-08	igain	507	3.849	13,18%	3.414	6,73	37,86	**8,28**	6.885
	uk-rand	90	6.683	1,34%	5.002	55,74	378,30	1,00	13.048
	dk-rand	88	3.491	2,53%	3.814	43,22	703,98	1,29	8.898
4-01	igain	565	1.304	43,32%	797	1,41	0,05	**2,27**	1.350
	uk-rand	699	5.320	13,14%	2.239	3,20	7,12	1,00	6.192
	dk-rand	668	4.083	16,36%	3.162	4,73	0,26	0,68	6.971
5-01	igain	2.360	50.666	4,66%	954	0,40	1,07	**15,16**	3.503
	uk-rand	674	51.824	1,30%	4.130	6,13	32,31	1,00	15.475
	dk-rand	284	31.463	0,90%	3.860	13,62	7,79	0,45	10.679
avg	igain	773	10.387	17,21%	2.041	4,12	14,14	**5,45**	4.000
	uk-rand	429	15.628	5,10%	3.917	15,77	135,99	1,00	10.477
	dk-rand	394	12.731	5,80%	3.830	14,30	190,98	1,01	9.637

Figure 50: The table shows results for queries from Figure 47.

Drastic speedup for IGAIN. The routing algorithm performs the cycle of querying the search engine and extracting missing relations for filling missing attribute values. This process can itself grow into a very time-and resource-intensive process. Strategy IGAIN requires fewer pages per query (*P*/*Q*) and per result tuple (*Docs/RT*) than any other strategy. As a result, we observe a speed up between our baseline UK-RND and strategy DK-COMB of an order of magnitude for queries 3-01, 3-08 and 5-01. On average we observe a drastic speed up of 5 times.

IGAIN is highly effective. The measurement *usage* represents the proportion of complete result tuples to all tuples in the buffer. This measurement indicates that the query processor not only collets incompletable tuples but *de facto* joins relations into complete result tuples. Strategy IGAIN outperforms all other strategies significantly for this measurement.

The measurement *(Docs/RT)* for queries 4-01 and 5-01 is near or even less than one page. This low value indicates that strategy IGAIN often discovers pages that contain multiple relations for answering these queries. Note, for some queries, the size of completed result tuples (*RT*) exceeds k=500.

This occurs when multiple tuples are returned from the last keyword query. Then the size of the full outer join of returned relations and result tuples in the buffer exceeds k.

IGAIN adapts quickly. The diagram in Figure 51 compares the ratio of processed documents vs. retrieved complete tuples, averaged over all queries. In addition, Figure 8 shows details for each query. From the first result tuple on, strategy IGAIN processes significantly fewer pages than other strategies. The trend becomes even more apparent with the growing size of complete result tuples.

Contrary, the Zig-Zag policy in strategy UK-RND does neither utilize information about the completeness of result tuples nor information about the attribute value distributions. Therefore, strategy UK-RND or DK-RND frequently generate a keyword query from a result tuple that has not yet reached a high degree of completeness. As a result, this keyword query is less likely to return missing values to complete the tuple. Worse, the keyword query probably returns many irrelevant pages from which the relation extractor returns new incomplete result tuples. Therefore, the probability that a randomly selected result tuple has a high degree of completeness actually decreases with each keyword query executed from strategy UK-RND and DK-RND.

Costs for Zig-Zag policy are difficult to predict. The table in Figure 50 shows the variance for each strategy. A low variance indicates predictable costs in terms of processed documents per completed result tuple. Only strategy IGAIN shows a low and practically relevant variance across all queries. Contrary, we report for strategies UK-RND and DK-RND a high variance which makes in practice a cost prediction for these strategies difficult.

We conclude that strategy UK-RND (as proposed by authors of [69]) is ineffective, while strategy IGAIN effectively returns missing tuples. Moreover, IGAIN adapts quickly in a cold-start scenario, where no information about promising join predicates exist and allows the query processor to predict query processing costs with low statistical variance.

8.3.3 Towards alternative quality indicators

The work in this chapter focuses on a single quality driven criteria, a completeness driven query processing that aims to complete tuples with retrieve rare, and less often repeated, relations first. The Eddy-based design of our query processor in Section 8.2 allows us to consider additional quality indicators for our routing policies. In this paragraph we will give some preliminary ideas on how to enhance these policies and we will list a few directions for such orthogonal quality indicators:

Due to the nature of uncertain content on the Web we noticed that tuples in results are redundant, contradicting and may even contain few false positives. Therefore an orthogonal direction of interesting research is to exploit this redundancy to cleanse complete tuples and to remove false positives. Authors of [72] envision operators for fusing or for ranking tuples. Moreover, instead of returning complete result tuples, the query processor may return *interesting relations* [66]. Consider a good set of results for the Entity "Einstein" which might include a mix of biographical facts like "*Einstein was born in Germany*" and other interesting facts like "*Einstein's favorite color was blue*". On the other hand, "*Einstein turned 15*" or "*Einstein wrote the paper*" might be less interesting because they express little useful information. Another metric is *nearness* (see also [130]); in this case relations are joined only into a result tuples if they are extracted from the same page or same Web site.

Another important direction is the enrichment of 'existing linked data, such as *Freebase.com.* These services rely on community efforts to submit relations. However, these services may only return relations that are 'extracted' by humans. Queries for any-k tuples can automatically discover complementary tuples and may help humans to discover rare and previously unseen relations.

8.4 Related Work

We review related work on executing structured queries across 'text databases', quality-driven query planning and text analytics.

Queries for specific–k relations: In Chapter 6 we focused on queries for *discovering specific-k relations*. Queries of that type execute a left outer

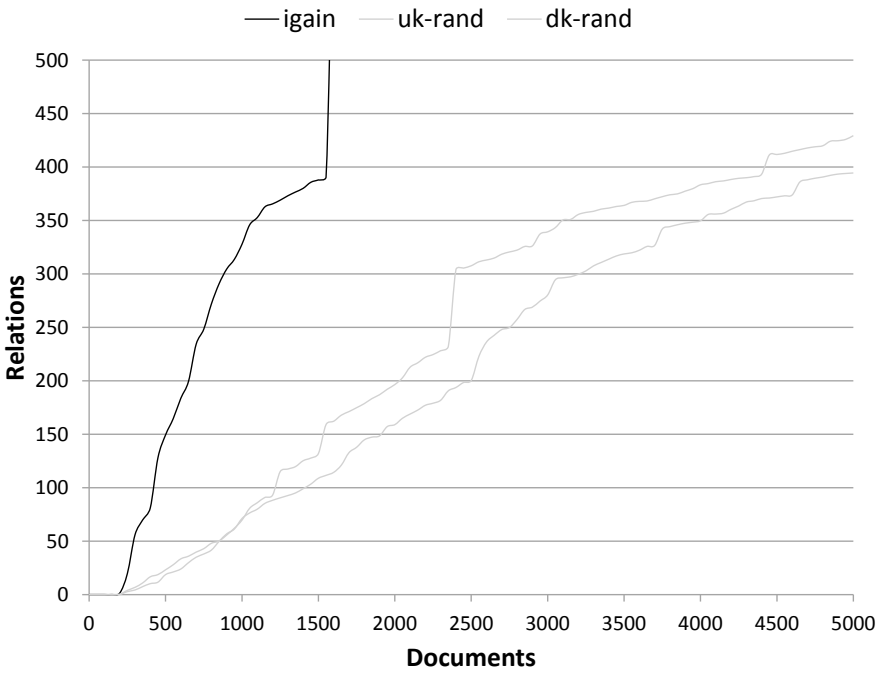


Figure 51: The Figure shows the result size vs. processed documents for different routing policies, keyword strategies and averages across all queries Figure 47.

join across a list of given entities and relations retrieved from Web pages. Contrary, queries for discovering *any-k* relations execute a full outer join on previously unseen relations that are discovered during query execution. Therefore, different routing policies are required. For instance, the routing policy must compute the degree of completeness or the chance of retrieving a missing relation during query execution and must incorporate these statistics into query routing decisions.

Binary joins across relations in 'text databases'. Authors of [67][68][69] propose strategies for executing SELECT-PROJECT-JOIN queries across "text databases" By a "text database" they refer to a document collection that can be accessed through a (filtered) scan (for instance, the entire index of a Web search engine). In order to estimate an optimal

join order for binary joins, attribute value distributions (that typically follow a Zipf-distribution*)* are estimated from a sample of documents from the 'text database' [88]. Downloading large samples requires random access to all pages of a Web search engine. However, our query processor has only sequential, keyword-based access to an ordered list of pages. This list may even change over time for the same keyword query. Consequently, each keyword query returns a single "text database". Estimating statistics for each keyword query is not possible in practice. Therefore we propose routing policies that can adaptively generate multiple keyword queries and select a query plan on a per-tuple-base. If such statistics cannot be computed in practice, authors suggest Zig-Zag joins [69] across binary relations. Our experiments demonstrate that Zig-Zag joins process up to an order of magnitude more pages per result tuple than other routing policies.

Automatic keyword query generation. A query processor requires discriminative keywords that likely return relevant pages from a Web search engine. In Chapter 6 we obtained such keywords in an adaptive, iterative fashion. We observe syntactic and lexical features from sentences in which an extractor extracted a tuple. Based on these features we trained a classifier. Keywords are updated when new pages are returned from the search engine. Therefore, generated keywords match the language of pages that are discovered during query execution. A similar method is presented by the authors of [70]. However, this method requires intensive domain specific training of the classifier prior executing a query as well as expert domain knowledge to generate a set of seed tuples.

Keyword based access to documents. Therefore that approach is not practicable for our setting. Worse, the set of keywords is fixed and does not adapt to new documents that the query processor discovered during query execution. Authors of [76] extract candidate keywords from meaningful attribute names and predicate values from a structured query. We do not expect meaningful attribute names in a structured query. Rather we trust that Web page authors have already developed a small set of common words and grammatical structures to express important relationships in natural language.

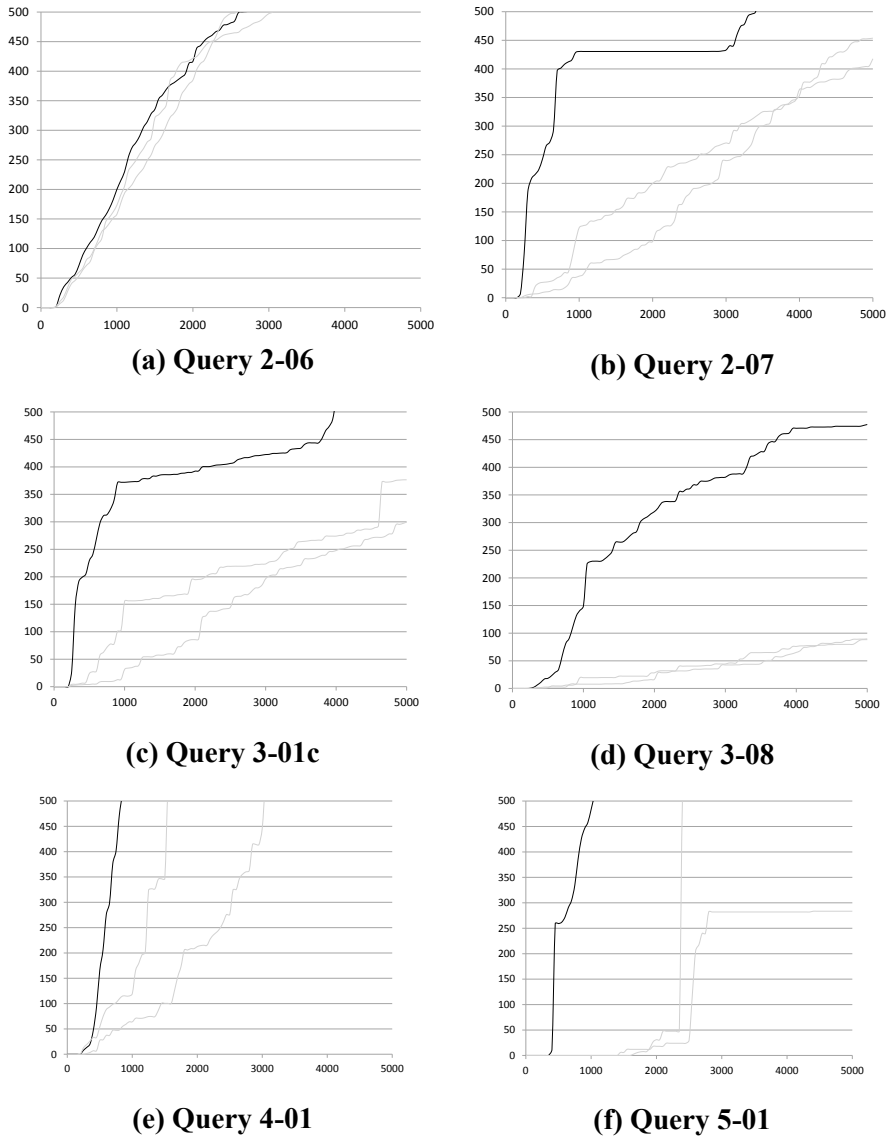


Figure 52: Detailed evaluation results for the queries from Figure 47**.** The dark black line indicates approach *igain*, the grey line approach *uk-rand* and the spotted grey line approach *dk-rand.*

Query planning and query processing. Among the large amount of literature, feedback loops for learning correct cardinality estimates [86] are

most relevant for us. However, these strategies assume full access to structured data in an RDBMS to compute a query plan. Branch and bound based query planning algorithms have been designed for integrating a small set of static databases [88]. In our scenario we are confronted with a sheer endless number of volatile data sources and may discover new sources during query processing. Instead of computing a single plan we update statistics during query execution and route queries adaptively along multiple plans [85]. Top-k query processing approaches [89] merge ranking lists from different search engines. Unfortunately, the ranking of a Web search engine does not reflect the existence of a missing tuple. Instead; we select missing tuples that can be likely retrieved with a keyword query.

Information extraction and text analytics: The CIMPLE [70] project describes an information extraction plan with an abstract, predicate-based rule language. More recently, open information extraction techniques generalize relationships with relationship-independent lexico-syntactic patterns [73]. We assume that these systems and the extraction mechanics are 'black boxes' for a Web searcher. Therefore we observe common patterns from extracted relations that likely reflect the internal mechanics of a relation extractor. We use these patterns for generating keyword queries.

'Semantic search' and 'linked data': Neither current Web search engines nor Business intelligence applications are able to answer queries for any-k complete tuples. As a proof of concept we designed the semantic search engine www.GoOLAP.info (see Chapter 10). The search engine discovers any-k tuples from news pages and blogs to populate its database. Another important scenario is the enrichment of 'linked data' [90]. Linked data services, such as *Freebase* [91], rely on community efforts to submit relations. However, these services may only return relations that are 'extracted' by community editors. Queries for any-k tuples can automatically discover complementary tuples for Freebase.

8.5 Conclusion

Neither current Web search engines nor Business intelligence applications are able to join extracted information from different Web pages and to return any-k complete tuples. In this work we presented two highly relevant

building blocks for solving that problem. First, we proposed a novel query processor for systematically discovering any-k tuples from Web search results with conjunctive queries. Because of the nature of data on the Web, relation extractors may return incomplete tuples. Therefore, we present a solid theoretical model and powerful, adaptive query planning techniques to iteratively complete these tuples. Another major challenge is to define discriminative keywords to return a small set of relevant pages from which a relation extractor returns tuples for a structured query. Manually formulating and updating such keyword queries for every Web site is difficult and time consuming. We discussed and solved this challenge by generating keywords with a self-supervised algorithm from Chapter 6. Our experiments demonstrate that our query processor returns complete result tuples and processes very few Web pages only.

9 Predicting Relations in Documents

Transforming unstructured information from Web pages into structured relations is quite costly and time consuming. Worse, many Web pages may not contain a textual representation of a relation that the extractor can capture. As a result many irrelevant pages are processed by relation extractors. For reducing these costs we propose a relation predictor to filter out irrelevant pages and substantially speed up the overall information extraction process. As a classifier, we trained a support vector machine (SVM). We evaluate our relation predictor on 18 different relation extractors at a per sentence level. Extractors vary in their number of attributes and their extraction domain. Our evaluation corpus contains more than six million sentences from several hundred thousand pages.

Our experimental study shows that the relation predictor effectively forwards only relevant pages to the relation extractor. We report a prediction time of tens of milliseconds per page and observe high recall across domains and a speedup of at least factor two (vs. a relation extractor only pipeline) while discarding only a minimal amount of relations. If only fixed amount e.g. 10% of the pages in the corpus are processed, the predictor drastically increases the recall by a factor of five on average.

The content and ideas of this chapter are published in [128].

9.1 Introduction

Knowledge discovery from the Web is a cyclic process. First, relevant data sources have to be identified and tapped. Second, the data has to be extracted and formalized. Third, data has to be linked and connected and finally will be used in various applications, e.g. to populate knowledge bases that can be used to answer complex analytical questions. Finally, the user feedback may trigger this process from scratch. For instance, a user may discover that some relations are missing. In that case the user may retrigger

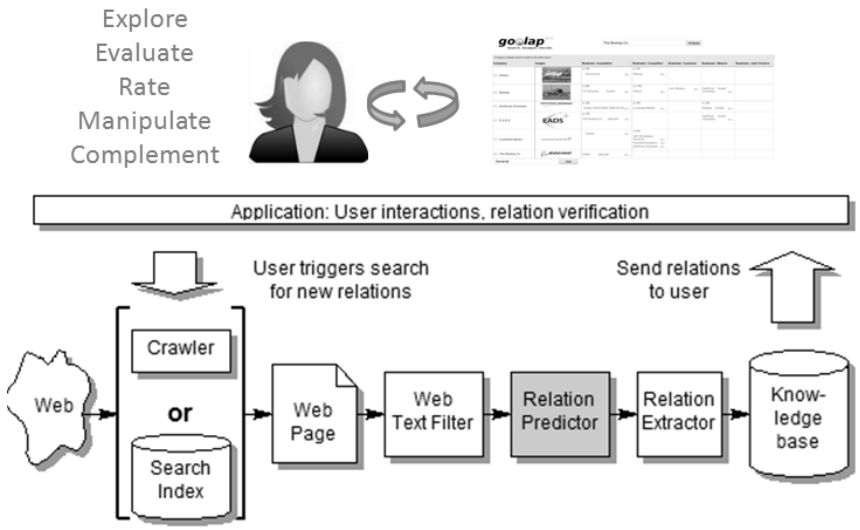


Figure 53: Extraction pipeline with our relation prediction component.

the entire process with a new search for additional data sources (see also Figure 53).

In this chapter we focus on the important part of formalizing information in this life cycle. For instance, formalizing information from Web pages is accomplished through information extraction and reasoning, which includes recognizing named entities and extracting semantic relations between them.

Current relation extraction systems, such as [79], [75][62][60], utilize a computational expensive execution stack of part-of-speech tagging, phrase detection, named entity recognition and co-reference resolution. This machinery is needed to detect the exact borders of attribute values, to identify their correct type and to identify their relation within and across sentences. As a result, processing a Web page may take up to a few seconds [93]. Higher precision is ensured with deep dependency parsing or semantic role analysis. These approaches require processing times of tens of seconds [93]. Recently, extraction system vendors also offer extraction-as-a-service [75]. These systems enable non-natural-language-processing experts to extract relations from Web text. This abstraction comes at the price that users

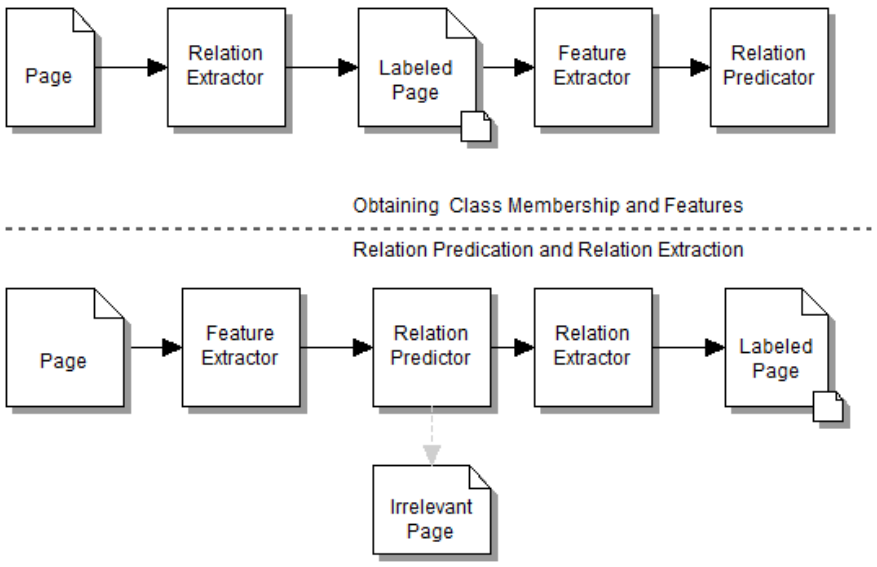


Figure 54: Experimental Setting: The upper pipeline shows the setup for training the predictor. The lower pipeline illustrates the setup used during prediction.

can no longer tune the patterns or rules of the extractor. Moreover, additional costs may appear, such as costs for sending a page through the network or monetary costs.

To obtain relevant Web pages one can either apply focused crawling on the Web or query existing Web search engines with generated keywords (see chapters 6 and 7). But even with these focused approaches only a small fraction of the retrieved web pages actually contains a textual representation of a semantic relation that the extractor can capture (see also chapter 7). It would thus be highly desirable to be able to filter out irrelevant pages that do not contain relations to speed up the extraction process and only forward relevant pages to the relation extractor.

Our Contribution: We propose a component called relation predictor that predicts whether an extraction service will be able to extract a relation from a web Page. The relation predictor only forwards relevant pages which contain a textual representation of a relation to the actual extraction

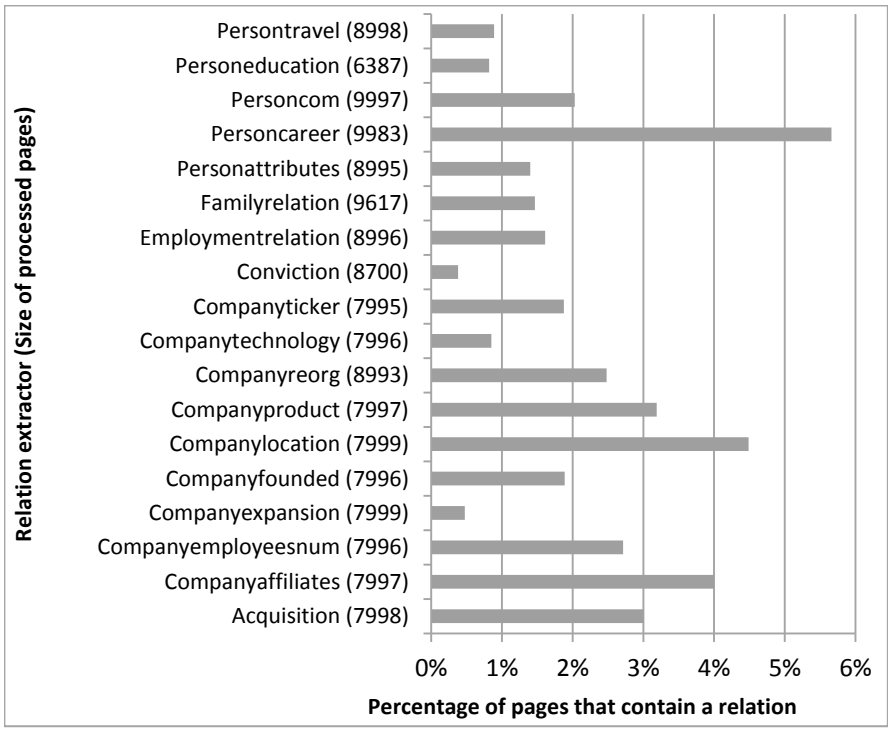


Figure 55: This figure shows the percentage of forwarded pages that contain a relation for 18 different relation extractors. Only a small fraction of processed pages de facto contains a relation.

service. The entire extraction pipeline is shown in Figure 53 and Figure 54. The relation predictor abstracts from extractor specific words and syntactic structures and is generally applicable to multiple extraction services. It avoids computationally intensive natural language processing techniques, such as part-of-speech tagging or deep dependency tree parsing.

9.2 Data Set Description

Relation extraction involves computationally expensive operations for each processed page. Therefore an important problem is forwarding only relevant pages to a relation extractor.

To consistently train and evaluate a predictor, we decided to generate a fixed corpus of annotated web Pages. We choose 18 different relationship

types of the domain ‘company’ and the domain ‘person’ from a public extraction service [75]. For each relationship-type we generated relation-specific keyword queries with a keyword generation strategy (see chapter 6). For relationship type we queried a Web search engine service with generated queries and retrieved the top-1000 pages. Some pages could not be downloaded however, so overall we retrieved about 153.000 pages. Each page is forwarded to a relation extraction service. The resulting set of annotation is seen as the gold standard, since we aim to predict whether a given relation extractor will be able to extract a relation.

Figure 55 shows the distribution of relations for different relationship types. Note, that because of the generated keywords, this sample is already optimally biased towards pages that likely contain a specific relation. Nevertheless we observe a rather sparse distribution of relations in our corpus. Apparently, extraction systems retrieve many irrelevant pages that do not contain any text that represents a relation.

9.3 Relation Prediction Problem

The problem of detecting a relation in a Web page is essentially a binary text classification problem. A classifier generally consists of a feature extraction component that extracts numeric values relevant to the decision and the actual classification algorithm that predicts the label of a document given the features. In this section we describe the general system setup, the linguistic feature extraction process and the actual classification algorithm we use.

9.3.1 Experimental Setup

Figure 3 shows the two different extraction pipelines that we use to train and evaluate the relation predictor. Once Web Pages have been retrieved either by crawling or via a search engine, they are sent through Web Text Filter. This component removes navigational elements, templates, and advertisements (so called boilerplate). It only keeps sentences and paragraphs [93] of a page. Figure 55 overviews this pipeline.

1. Training the Predictor. Most semantic relations are expressed within a single sentence. Therefore we choose to evaluate each Web page on a sentence level rather than treating the whole page as a bag of words. To gen-

erate labeled training instances for the classifier, we use a setup depicted in Figure 55 (upper half). First, we forward training pages to a relation extractor, which labels sentences as either containing a specific relation or not. Next, we transform each sentence to a token representation of shallow text features as described in Section 9.3.2. Finally we encode the occurrence of bigrams of these tokens as binary features x_i to produce l labeled training samples of type (x_i , y_i) $(1 \leq i \leq l)$, where x_i is the feature vector in our n dimensional feature space, and $y_i \in \{-1, +1\}$ denotes the class label.

2. Prediction. At prediction time we segment a page into sentences and transform each sentence into its feature vector representation. We then predict the label for each sentence in the document. If a single sentence in a page is predicted as positive, the entire page is forwarded to the relation extractor. Otherwise, the page is discarded and removed from the relation extraction pipeline. Figure 55 (lower half) illustrates this process.

9.3.2 Linguistic Feature Analysis

Predicting relations for thousands of relation extractors is difficult. For instance, word-based n-gram models can result in tens of thousands of relevant features, which apparently would make the relation predicator susceptible to over fitting to a particular page subset. Therefore authors of [73] utilized part-of-speech tags and predefined patterns for predicting binary relations. Because of the part-of-speech analysis this approach is computational expensive. Furthermore it is only limited to binary relations, in particular relations in which entities are separated by few tokens only. Authors of [94] overcome this shortcoming and recognize the structural locality of generic relations with deep dependency tree parsing.

Relation prediction requires 16 tags only. We transform each sentence into a set of tokens representing shallow text features. Figure 4 gives an overview our set of 16 token tags. In the reminder of this section we illustrate our principles for extracting lexical, syntactic and semantic features of attribute values and relations.

Sentence Transformation. General attribute values as shallow text features. Authors of Web text may ‘invent’ new nouns to express an attribute value. Therefore the number of potential attribute values may become too

large to be held in a hash list. We encode any lowercased tokens, such as verbs and nouns, as (L), tokens starting with an uppercase character, such as proper nouns or acronyms, as (U) and tokens starting with a digit as (D).

Special Case: Closed Classes as Hash Lists. Lower cased nouns may start with determiners, common nouns or pronouns. For these closed classes authors of Web text rarely invent new words. Closed classes help us distinguishing lower cased nouns from other lowercased tokens. We encode English language prepositions (P), determiners (T) and pronouns (R) as hash list.

Special case: Comma. Sequence of noun phrases, proper nouns or digits are a rich source of attribute values. We encode the comma as (,), and encode coordinating conjunctions (C) as hash list, such as 'and', 'or', '&'.

Example: *The following examples showcase our encoding rules for typical phrases that may express an attribute value:*

Complex noun: *the wooden tool → TLL*

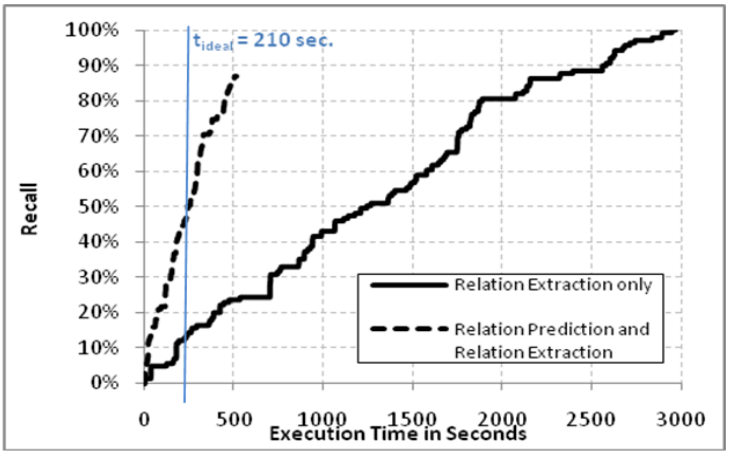
Proper noun: *SAP AG → UU*

Acronym: *IBM → U*

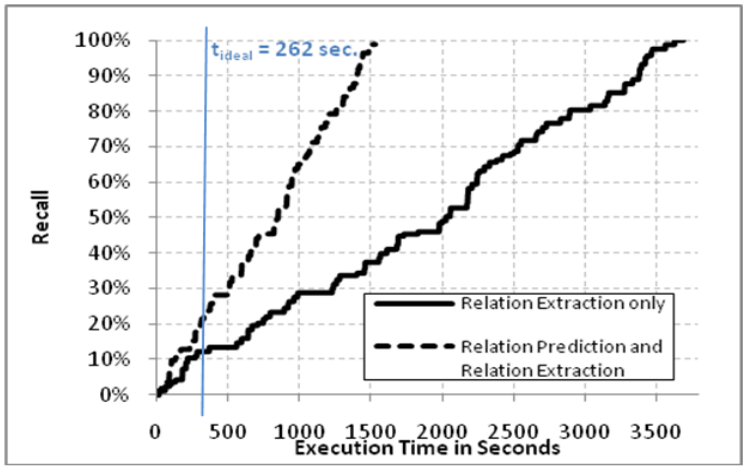
Complex proper noun: *Sony VX 5a → UUD*

Date example: *25 September 1969 → DUD*

Extracting Relations. Verb phrases as hash list: Verb phrases and prepositional verb phrases frequently indicate a relation between two entities in English language Web text [43]. For instance, prepositional verbs resemble phrasal verbs in that both verb forms consist of a verb followed by a preposition. Relation extractors utilize verb dictionaries to determine the semantic type of a relation. We reuse these dictionaries and generalize single and multiword verb phrases in sentences as (V). Thereby, we assume that a relation classifier can access these verb phrase dictionaries. For instance, dictionaries can be obtained from rules of declarative relation extractors [21].



Example results for relationship type *Acqisition*



Example results for relationship type *CompanyReorgaization*

Figure 56: We measure execution time and recall for the relation extraction pipelines acquisition (left) and companyreorganization (right)‘ with and without the predictor on the test data set. The predictor significantly speeds up the relation extraction process by at least a factor of 2 with only modest information loss. (Pages are processed in the same order.)

For obtaining a representative hash list of verbal phrases, we bootstrap the relation extractor with a small sample of pages. Next we observe preposi-

tional verb phrases in returned pages. The method is efficient and requires only a few hundred pages.

Special Case: Appositive and Possessive Relations as Shallow Text Features. We add a tag (‘) for encoding relations that are expressed with a possessive apostrophe. We also add the colon tag (:) for encoding relations that are expressed as appositive phrase.

Example: *The following phrases showcase our encoding rules for typical relationship types. We underline the plain text and the corresponding features that express the relation:*

Relation with verb phrase:

SAP AG recently acquired performance management vendor Cartesis. → *UULVLLLU. ($V_{ACQUISITION}$='acquired')*

Relation Extractor: ACQUISITION (Acquirer, Acquired)

Relation with possessive apostrophe:

said Berlin's politically ambitious mayor, Klaus Wowereit on a press conference yesterday. → *LU'LLL,UUP_PTLLL.*

Relation Extractor: PERSON_POSITION(Person, Position)

Feature Class Hierarchy. Sometimes a token can be mapped to multiple encodings. Part-of-speech taggers disambiguate different part-of speech classes within a single sentence. However, this high precision approach comes at the price of a large feature set that causes longer execution times. The focus of our scenario is different. We can relax the requirement of a precise interpretation for each sentence that would require a part-of-speech tagger. Instead, we introduce a hierarchy among different feature extractors. We consider shallow features as less discriminative than syntactic features and syntactic features as less discriminative than semantic features. If a token or a set of tokens matches for multiple feature class, we select the feature class that is most discriminative. For instance, the token ‘acquired by’ can be encoded as LL, LPP or V. We select the encoding V since it belongs to the most discriminative feature class.

Relation Extractor	Fact Recall after 10% processed pages	
	Extraction Only	Prediction & Extraction
acquisition	16.5%	70.5%
companyafilliates	9.7%	64.8%
companyemployeesnumber	11.2%	43.8%
companyexpansion	6.3%	25.0%
companyfounded	10.1%	60.9%
companylocation	13.9%	42.4%
companyproduct	10.9%	51.7%
companyreorganization	13.5%	84.0%
companytechnology	10.0%	43.3%
companyticker	11.5%	93.4%
conviction	0.0%	42.1%
employmentrelation	3.9%	66.7%
familyrelation	17.2%	17.2%
personattributes	7.3%	41.8%
personcareer	8.9%	12.8%
personcommunication	9.2%	19.5%
personeducation	0.0%	57.1%
persontravel	14.3%	25.7%

Figure 57: The figure shows the recall after 10% of the pages in the test corpus have been processed by the relation extractor both with and without the prediction component in the pipeline. The relation predictor drastically increases recall by effectively filtering out irrelevant pages across various types of relations.

Feature Extraction. After a sentence is transformed into its token representation, we encode the occurrence of bigrams of these tokens as binary features x_i to generate a feature vectors for the classification algorithm.

9.4 Classification Algorithm

The problem of detecting a relation in a Web page is essentially a binary text classification problem. We employ linear Support Vector Machines

(SVMs), which have already been shown to produce superior accuracy on related tasks such as text categorization [96]. SVMs are maximum margin classifiers based on the structural risk minimization principle [96]. SVMs can be trained efficiently as binary classifiers and generalize well on unseen data. Linear support vector machines find a hyper-plane that separates the classes of training examples with maximum margin. This can be formulated as a convex optimization problem. We used the *liblinear* implementation of a L2-loss-L2R soft margin SVM [97], which solves the following quadratic program:

$$\min_{w,\xi_i} \frac{1}{2} ||w||^2 + C \sum_{i}^{l} \xi_i{}^2$$

$$s.t. \qquad y_i(w * x_i + b) \geq 1 - \xi_i \ \ \forall x_i$$

$$\xi_i \geq 0$$

In this equation, C denotes the cost of misclassification and is to be determined through cross validation. The soft-margin SVM allows for some misclassifications of training samples through the slack variable ξ_i. To accommodate for the fact that our training data is heavily unbalanced (>99% of the sentences are negative training samples), we introduce different weights for the misclassification cost c_i for each class $y_i \in \{-1, +1\}$.

9.5 Experimental Evaluation

In this section we report on the robustness, recall, costs, execution time and performance of our relation predictor.

We divided our corpus into two parts: 60% for training and 40% for testing the relation predictor. From these pages we removed boilerplate templates [93] and finally extracted 6.7 million sentences. From the test set we removed non English pages and pages that did not contain any extractable content. Overall our test set that we report our results on contains 42,000 pages. We trained the classifier for each relation predictor in our data set with different settings on the 60% training pages with different parameters. We then evaluated these classifiers on the test set.

We evaluated the predictors against a gold standard. It reflects the maximum number of relations that the evaluated relation extraction service could extract from our corpus, if all pages would have been processed. We ran two different scenarios:

Relation Extraction only. This scenario implements a full scan over the entire set of pages in our test corpus [88]. The scan is executed in a random order. During the scan, each page in the test corpus is forwarded to the relation extractor. Note that this scenario reflects the current access method for public extraction services, such as [75], or the access method of several academic relation extraction systems, such as [21][28][43][51][62][73][75][79].

Relation Prediction + Relation Extraction. This scenario utilizes our relation predictor. Each page in the test set is analyzed by the relation predictor. For each page the predictor decides whether the page is forwarded to the relation extractor or discarded.

9.5.1 Performance Improvements

To demonstrate the performance impact of our predictor, we measure the execution time of the entire relation extraction pipeline both with and without the predictor component on the entire test corpus. Figure 3 (lower part) illustrates this process. Figure 5 shows the recall and execution time for two example relation extraction pipelines *'acquisition'* and *'company reorganization'*.

Our relation predictor is about two orders of magnitude faster than the actual relation extraction service. (The prediction time includes boilerplate removal, sentence segmentation, feature extraction and the actual prediction computation.) For example the predictor takes 27ms per Web page on the relation *'company location'* and 92ms per Web page on the relation *'person career'*.

Imagine a perfect predictor that only forwards relevant pages. The time it would take this predictor to process the entire corpus of Web pages is given by the following equation:

$$t_{ideal} = n_{pos\ documents} \times \left(t_{extraction} + t_{prediction}\right)$$

$$+ n_{neg\ documents} \times t_{prediction}$$

The lower bound for the execution time is represented by the two vertical lines in Figure 56. As an example, we show predictors suited for the ad-hoc query scenario. Here low execution time is the most crucial component, while some missed relations are acceptable.

For both presented relation extraction pipelines there is a substantial speedup in the system, when the predictor is introduced. In the case of *'acquisition'* (left), the system with a predictor is about five times faster than the unfiltered system while still achieving almost 90% recall. Note this is only twice as long as the ideal predictor would take. In the case of *'company reorganization'* (shown on the right) there is still a speedup of factor two, while almost 100% recall is achieved. Analogue, we observed similar results for each of the remaining 16 relation extractors.

9.5.2 Recall Enhancement across Domains

To illustrate the performance improvements for different types of relation extractors, we measured the recall (Percentage of total facts retrieved) after 10% of the pages in the test corpus have been processed by the relation extractor. Results for 18 different relation extractors are shown in Figure 6. The *'prediction + extraction'* pipeline achieves significantly higher recall than the pipeline without the prediction component across almost all different relation extractors. This is a remarkable improvement.

9.5.3 Discussion

Our evaluation results clearly show the usefulness of our relation predictor. The relation predictor robustly predicts relevant pages for 18 different relation extractors.

Robust prediction for a wide range of attributes and relationship types. With our simple and small feature set the predictor recognizes the textual representation of values for very different attribute types, such as date, company, person, location, product, product type, technology, con-

viction charge, ticker symbol, job position, or degree. Moreover, this feature set enables the relation predictor not only to recognize binary relations, but also complex relations with five or more attribute values in Web text.

Robust Prediction for both, rare and frequent relationship types. The relation predictor robustly identifies relevant pages for both, frequent and infrequent, relationship types. As a result, our relation predictor is particularly helpful for 'rare' relationship types. For instance, relations of the type '*company technology'*, *conviction*, '*company expansion',* or '*person education'* appear in less than 2% of the pages. Our relation predictor effectively filters out irrelevant pages for these relationship types.

Slightly higher recall for domain 'Company'. We observe a slightly higher recall for the domain *company* than for the domain *person*. For instance, 6 out of 10 relation extraction pipelines with relation predictor for the domain *company* achieved a Recall@10% forwarded pages of more than 50%. Contrary, we observed comparatively lower measurements for relation extractions that contain attribute values of the type person or location, such as for the relation extractor '*family relation'* and '*person communication'*. We examined missed pages that contain relations for these types and attribute values. For instance, one problem is the recognition of relations between a lower cased person name, such as *lady gaga,* and another person that is expressed with a lower cased pronoun. In our future we will extend our feature set to capture these rare cases as well.

9.6 Related Work

We review work on text filtering, pattern generation and relation extraction optimization.

9.6.1 Text Filtering and Pattern Learning

Text Filtering. Most closely related to our approach is the usage of information extraction systems to perform text-filtering, as discussed in MUC-4 studies [99]. Text filtering components at MUC-4 pre-label sentences with named entities or part-of speech information (or expect pre-labeled docu-

ments). Next, a rule-based pattern or a statistical classifier filters out documents that do not belong to a specific relationship type.

Pattern Learning. Riloff [101] pioneered a learning procedure in which documents are marked as relevant to the relation extraction task. The learning procedure receives a set of POS-tagged and outputs a set of textual patterns that can be used for discovering additional relations. A plethora of different pattern learning system enhanced this approach (see [102] for an overview). For instance, authors of the PROTEUS system [100] suggest a set of regular expressions pattern for learning good-quality patterns automatically from a large, general, un-annotated corpus of documents. Most recently, systems for Open Information Extraction (OIE) [94][73] extract short (up to 5 token) and generic relations between noun phrases in English language Web text. These generic relations are extracted with few syntactic patterns on POS labeled text. Later, the same authors proposed in [93] preliminary ideas to capture also distant relations with the help of deep dependency parsers.

Our relation prediction approach is fundamentally different from previous text filtering and pattern generation approaches. We do not require tags generated through part-of-speech tagging, noun phrase chunking, named entity recognition, co-reference reconciliation or deep parsing. As a result, we predict relations in a page in a few milliseconds, instead of seconds (with part-of-speech tagging) or tens of seconds (with deep parsing). Furthermore our schema is generally applicable to a wide range of relation extractors and not fine-tuned to individual scenarios such as in the MUC-4.

We assume a relation extraction service as a black box. Moreover, and contrary to existing pattern generation approaches, we do not require sending feedback to the relation extractor, making our approach applicable for a broad range of relation extraction scenarios on the Web.

Unfortunately, MUC-4 studies could not verify the usefulness of text filters for improving the performance of a relation extraction system. To our best knowledge, we are the first study that shows the general applicability and usefulness of predicting relations for a wide range of different relation extraction systems. Our results demonstrate a drastic reduction of processed pages while remaining a high recall.

9.6.2 Optimizing Relation Extraction Systems

Optimizing a relation extraction pipeline has been pioneered by authors of [88]. They propose cost models for extraction operations for an offline Web crawl scenario and for an ad-hoc Web search scenario. For instance, they propose a full scan operation that forwards each page in the corpus to a relation extractor. The index scan operator assumes an existing index of crawled pages, such as the index of a Web search engine. The index is queried for pages that likely contain instances of a particular relationship type. Each query returns a list of top-k pages that are forwarded to an extractor. As a result the relation extractor only receives pages that like contain a textual representation of a relation. In Chapter 6 we report an analysis of effective keyword query generation strategies that likely return such pages.

The filtered full scan operator and the indexed filtered scan operator utilize a relation prediction component. Authors of [88] utilize fine-tuned rules for filtering pages. Unfortunately, the study discusses two relation extractors only, such as the disease outbreak and headquarters extractor. For instance, for the disease outbreak extractor, authors of [88] report a maximum recall of 60%. Our evaluation in Section 9.5 is significantly more comprehensive. We compare a full scan operation with a filtered full scan operation for 18 different relation extractors. For all relation extractors we report a high recall and comparable low execution costs.

An orthogonal optimization strategy is caching extracted relations for frequently requested pages, such as Semantic Proxy [75]. The authors of System T [98] optimize the evaluation order of extraction rules. These optimizations are within the relation extractor. Contrary, our approach assumes the relation extractor as black box. Therefore these optimizations are orthogonal to our approach and may enhance the throughput of the relation extraction system further.

9.7 Conclusion

Scaling relation extraction to large document collections, such as the Web, is a challenging problem. For instance, current relation extraction systems are computationally expensive: they might require several seconds to process a single page. Therefore it is too costly to sequentially scan and forward all Web pages to the extractor. However, often such an exhaustive inspection of all pages is not necessary, since only a few relevant pages contain a textual representation of a relation anyways.

We presented a relation predictor, which classifies Web pages as either relevant or irrelevant for a relation extraction task. As a classifier we trained a support vector machine that evaluates pages on a sentence level, where each sentence is transformed into a token representation of shallow text features.

In a broad experimental evaluation on more than hundred thousand pages we demonstrated that our technique robustly predicts relevant pages for 18 different relation extractors with varying attribute types. Most importantly, our relation predictor is two orders of magnitude faster than a relation extractor. The extraction process can be speed up by at least a factor of two.

This tremendous increase in processing time per page allows us to deploy existing relation extraction systems at a large scale and for a wider range of applications than previously possible. For instance, we utilize our relation predictor technique within a highly interactive analytical Web search engine that can be reached at *www.goolap.info* (see also Chapter 10).

Feature	Token	ENC	Example	Role	Explanation
Shallow text	starts with lower case token	L	wood stock, green apple, recently said	indicates attribute value, indicates relationship	indicates lower cased part-of-speech elements, such as adverbs, adjectives, verbs, lower cased nouns, particles, interjections etc.
	starts with upper case token	U	NBA, Berlin, Mercedes Benz, Mark Hurd	indicates attribute value	indicates proper nouns
	starts with digit	N	1997, 25th., 50%, 80legs.com	indicates attribute value	indicates numerical value
	date	D	04.07.1983	indicates attribute value	indicates a date
	starts with punctuation	!	?,!, ., ;	sentence structure	separates clauses in a sentence
Syntactic	articles	A	a, an, the	indicates attribute value	is a noun-modifier, connects noun or noun-phrase to context of sentence
	determiners	D	few, all, most	indicates attribute value	is a noun-modifier, connects noun or noun-phrase to context of sentence
	is comma	,	Angela Merkel, Nicolas Sarkozy	indicates attribute value	represents a list of values
	is dash	-	Flight Paris - London	indicates relationship	represents a closed range relation between values or a contrast between values
	is semicolon	;	Michael went ot; he will be back at 5.	sentence structure	separates clauses in a sentence
	is coordinating conjunction that indicates a sequence	C	and, or, &	indicates attribute value	represents a list of values
	is preposition	P	of, at, to, instead of, prior to, in spite of	indicates relationship	links attribute value to other tokens, such as predicates, verbs and nouns
	is possesive apostrophe	'	IBM's DB2, Inga's dauther Mathilde	indicates relationship	represents possesive relation
	is colon	:	Zeta-Jones Filmography:	indicates relationship	predicate of a relation, substitutes verb phrase
	is personal pronoun	PPN	You, we, I	indicates attribute value	represents initiator of a relation, substitutes for common or propper nouns
	is object pronoun	OPN	it, which, that	indicates attribute value	represents target of a verb that indicates the relation
	possesive pronouns	PN	mine, his, her	indicates attribute value	represents possesive relation
Semantic	is relation specific verb phrase	V	was aqcuired by (relationship 'acquistion')	indicates a relationship	represents a verb/ prepositional verb phrase in a relation
	is entity name	E	VMWare	indicates a relationship	represents a proper noun in a relation

Figure 58: Our tag set contains 16 differentclasses used for predicting relations. Each sentence is tokenized and transformed into a general representation using only these 16 tags. Tags help to identify tokens that represent structure in sentences, tokens that represent potential attribute values or tokens that represent a relation for a specific type.

10 GoOLAP: Interactive Fact Retrieval from Search Engines

GoOLAP provides powerful operators for retrieving, extracting and analyzing factual information from textual representations on the Web. At the core of this service lies FactCrawl, a powerful engine for the retrieval of unseen or missing facts from a Web search engine to a structured fact base. Compared to existing approaches that crawl and maintain very large Web archives, GoOLAP tries to minimize retrieval costs through user triggered fact retrieval. Such an approach poses a lot of technical challenges, such as the design of an iterative fact retrieval process, the interpretation of user interaction and automatic keyword query generation. We will present the opportunities and challenges that we met when building GoOLAP, a system that already enjoys a broad user base and over 6 million objects and facts.

The content and ideas of this chapter are published in [123].

10.1 Introduction

Which companies collaborate with Boeing? Are these organizations also collaborating with Airbus or Fokker? Do employees of these companies have a criminal record? Who published the information?

Each day new pages emerge in the Web that may contain a textual representation of facts for answering these questions. Strategic decision makers may frequently research the Web for questions like these. Often answers to these queries might not be published in textual or structured form by Web 'information aggregators' like Wikipedia.com, Freebase.com, Trueknowledge.com or Yago [27]. Rather, this rare factual information is hidden in unstructured Web text on a few Web pages of news agencies or blogs. Unfortunately, collecting factual answers from these pages with a general Web search engine is still a dreaded process for a user. One option

to populate a fact base is to crawl a large document collection. For instance Google Squared [103] populates its data base with facts from the large corpus of the general Web. The system extracts these facts from tables, lists and from text with open information techniques. However, in Chapter 7 we observed that only a small fraction of a large archive de facto contains factual information. Hence, strategies that might execute a full scan over the entire archive can drastically waste processing and storage resources (see also [88]).

Another option is discovering facts in retrieved pages from ad-hoc keyword search. Unfortunately, this is still a tedious task, since Web search engines do not return aggregated factual information. The heuristic of a search engine user is: type in keywords as queries, 'extract' relevant facts from the top-k documents, filter out relevant facts and compile facts into a structured fact table. Therefore the user typically repeats this process multiple times in order to complement missing attribute values and to enhance the chance to discover unseen facts.

Significance of our approach. We present GoOLAP, a system that aims to automate the fact retrieval process from Web search engines. GoOLAP has three significant aspects: (1) The system provides powerful operators for analytical Web research on textual information, such as augmenting facts for an object, tracing back the textual origin of a fact or comparing factual information for a list of objects. (2) As a natural alternative to crawling a large proportion of the Web, GoOLAP interprets user interactions as input to identify missing facts. These user interactions trigger a fact retrieval process with the goal to populate the GoOLAP fact base from Web-scale indices of existing search engines in an ad-hoc fashion. This process is powered by the FactCrawl engine that leverages sophisticated keyword generation techniques (see Chapter 7) and our page classification techniques (see Chapter 9) to retrieve only pages that likely contain missing and rare factual information. (3) GoOLAP combines these approaches to drastically avoid crawling, indexing and extracting potentially billions of irrelevant pages.

10.2 Exploring GoOLAP's Fact Base

This section describes how a user may explore the GoOLAP fact base, see also the work in [104] for an overview of exploratory search:

Interpret. A query typed into a search field can express different intentions. For example, a user has the intention *Augment facts for Boeing*, so he enters *Boeing* into the search field. While he is typing the query, the system returns multiple interpretations in an auto complete dropdown, such as *company:Boeing*, *person:Boeing* or *product:Boeing*. The user may select an interpretation for Boeing (e.g. company Boeing) to send the interpreted query to the system.

Augment. This operation collects and augments information for a particular object. In addition, the system returns images of the selected object and shows links to objects of the same type, such as other companies. Finally, the system returns a list of ranked facts. This process has two steps. (1) Fact type enumeration. The system takes as input the selected object. Each object corresponds to a particular type for which the system looks up known fact types in a system-wide ontology. Currently this ontology holds over 70 different fact types. For instance, for an object of the type company, the system returns fact types, such as *layoffs*, *competitors* or *products*. (2) Return most interesting facts for each type. For each fact type the system queries the corresponding table for non-empty facts. Next, it ranks facts for each type by the notion of interestingness [9].

Expand. This operation enables a comparative analysis across facts for multiple objects. Consider a user who wants to compare different aircraft vendors, such as Boeing, Fokker or Northrup Grumman. For each object in this user defined selection our system projects a set of facts per type and creates a tabular fact sheet. Objects are displayed as rows, fact types as columns. Each cell lists facts ranked by interestingness. Note that this table represents a non-first normal-form (NF2) of the underlying fact base; therefore multiple values per cell may exist.

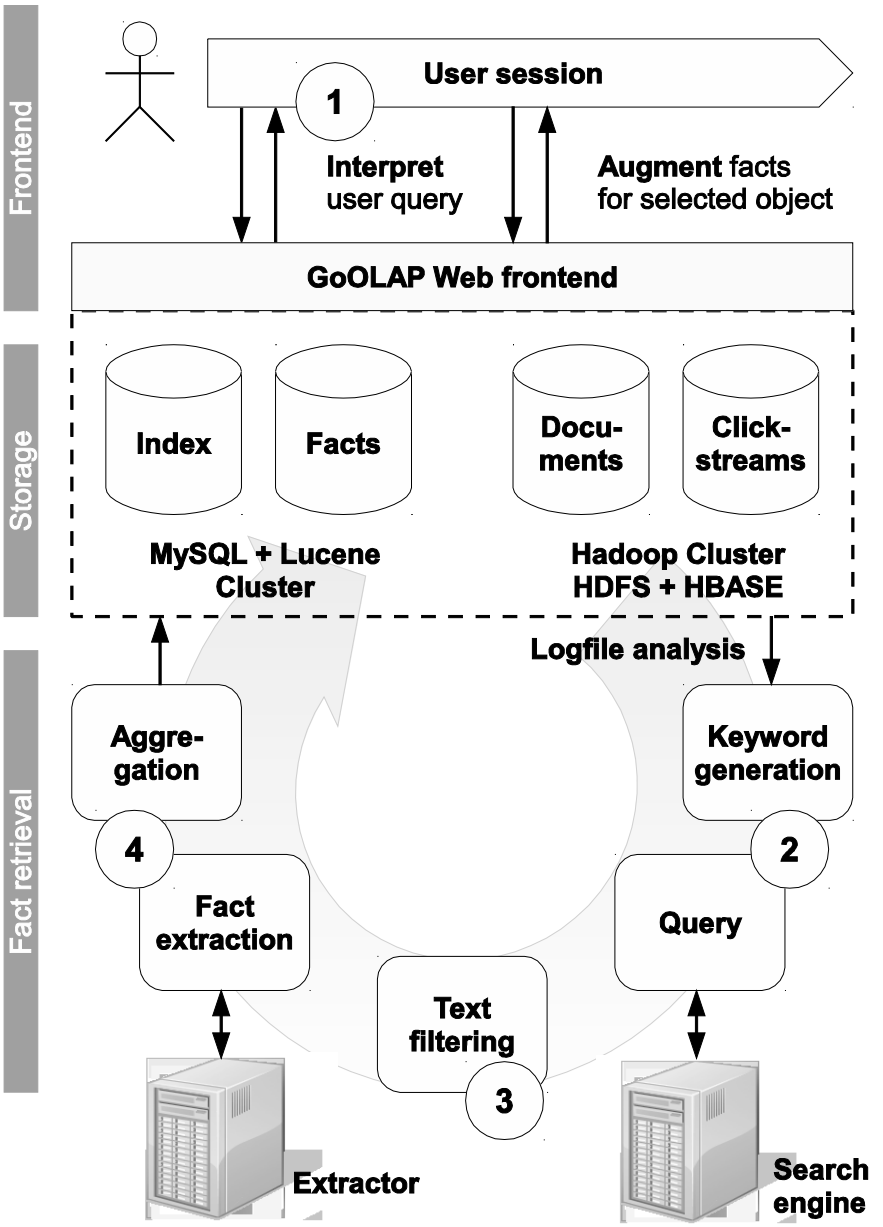


Figure 59: User triggered fact retrieval from Web search engines. The fact retrieval process is interactively triggered by user interaction, here INTERPRET and AUGMENT.

Interaction:	**Intention:**	**Display**	**Retrieval**
The user …	The user ...	The system ...	Fact Crawl …
clicks on a link that points to GoOLAP.info	discovers a GoOLAP result page for *Company:Boeing* in another search engine and clicks on the link	displays the AUGMENT result page for *Company:Boeing*'	retrieves potentially missing facts for *Company:Boeing,* such as *headquarter*, *layoffs*
searches missing object	likes to augment facts for an object *X* that is not yet contained in the GoOLAP fact base	displays a message to inform the user about the empty result and proposes a subscription of *X*	retrieves initial facts for the object *X*
searches and interprets known object	requests to augment facts to the object *Company:Boeing*	augments facts to '*Company:Boeing*' and displays the AUGMENT page for that object	ranks the popularity of *Company:Boeing* and retrieves potentially missing facts, such as *headquarter*, *layoffs*
clicks on AUGMENT	requests an AUGMENT result for the object '*Company:Boeing*'	displays the top rated facts for '*Company:Boeing*' and three arbitrary documents containing information about *Boeing*	ranks the popularity of *Company:Boeing* and retrieves more facts
clicks on TRACE BACK the document	likes to trace back the textual origin of the fact '*EADS competes with Boeing*' 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	ranks the popularity of the fact *EADS competes Boeing* and tries to retrieve more evidences of this fact
clicks on EXPAND list of objects	desires to compare *Competitors of Boeing* and clicks on the EXPAND icon in the column header	expands the list of '*Competitors of Boeing*' and displays an interactive table view for comparison	retrieves potentially missing facts for *Company:Boeing* and for relation 'competitive to' Airbus, EADS, Fokker etc.

Figure 60: GoOLAP collects and interprets more than 30 user interactions that may trigger a fact retrieval process. This Figure displays user interactions and system activities contained in our demonstration scenario.

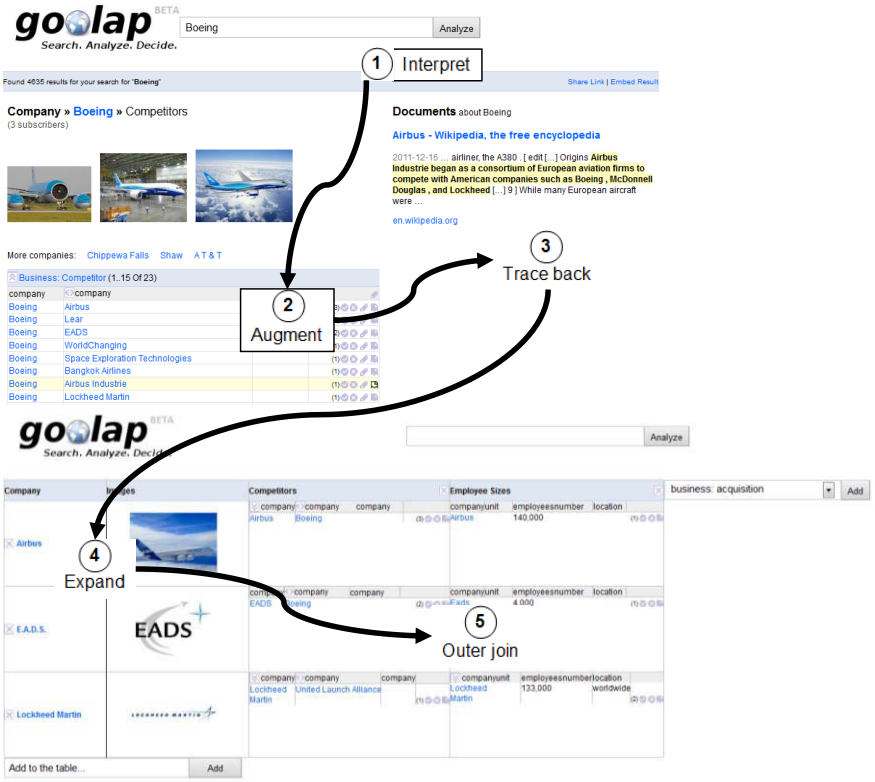


Figure 61: A typical user session with the goal to collect facts about competitors of company Boeing.

Trace back. For each fact, the user can access the original document at any point of the assessment process in order to continue his research.

Subscribe. Each user has the possibility to register a profile on the GoOLAP site. This enables a registered user to give feedback to the results by clicking on agree or disagree buttons or editing factual data in-place. If a user is interested in a particular object, e.g. his own company, he can subscribe to the object by clicking on the star next to the object's name. The system then tries to fetch more data for the subscribed object.

10.3 User-Triggered Fact Retrieval

Figure 59 introduces our general fact retrieval process:

Step 1: Interpret and execute user query. The system collects user interactions, determines the intent of the user and executes a query against the local fact base. While a crawl-based fact retrieval execution guarantees that all documents in an archive are processed, an indexed-based execution might miss some relevant documents. For compensating this disadvantage, GoOLAP observes the *completeness* and *rarity* (see also Section 8) of returned results to identify potentially missing facts. If necessary, GoOLAP schedules a fact retrieval activity (see also Figure 60); for example, if a user requests facts for an object that is not yet contained in the GoOLAP base, the system will trigger the retrieval of pages that contain facts for the missing object.

Step 2: Generate keyword query to retrieve missing facts. GoOLAP forwards the corresponding textual object representation and the semantic type of the missing fact to FactCrawl (see Section 7), a framework that emulates a search engine user's behavior to solve an inverse retrieval problem. This framework queries a Web search engine with automatically generated keywords, re-ranks the resulting list of URLs according to a novel fact score and forwards only promising documents to a fact extractor. FactCrawl generates keywords using structural, syntactic, lexical and semantic information from sample documents. Thereby FactCrawl estimates the fact score of a document by combining the observations of keywords in the document.

Step 3: Filter out irrelevant pages. GoOLAP utilizes a fact predictor (see Section 9); this technique reliably filters out irrelevant pages and only forwards relevant pages to the GoOLAP fact extractors. Most importantly, our fact predictor is two orders of magnitude faster than the fact extractor. The fact predictor is based on a support vector machine that evaluates pages on a sentence level, where each sentence is transformed into a token representation of shallow text features.

Step 4: Fact extraction. GoOLAP extracts factual information and resolves objects with home-grown extractors based on work proposed in [73] and commercial extractors, such as OpenCalais [75]. Overall, GoOLAP provides more than 70 different fact extractors for the domains people, companies, media and entertainment. The system is open to additional fact

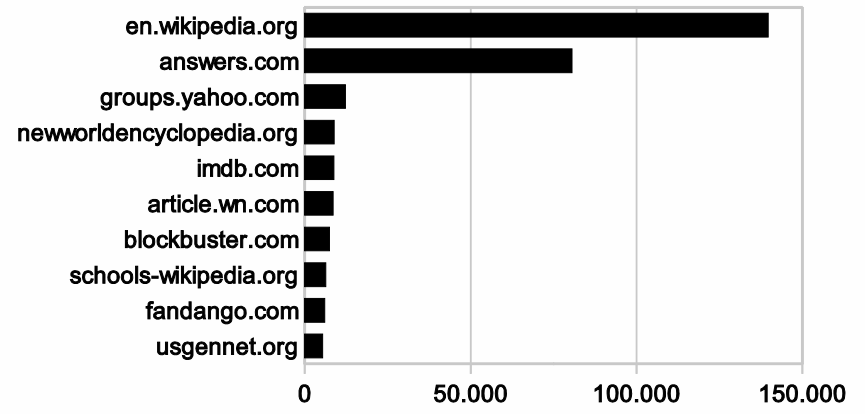


Figure 62: GoOLAP receives facts from more than 40.000 different Web domains. This distribution shows the top-10 domains that provided most facts. The X-axis depicts the number of extracted facts per domain.

extractors such as *extractiv.com* or *alchemy.com*. The retrieval process then aggregates the extracted facts and stores them into the local fact base.

Data storage and parallel execution engine. Factual information is small compared to the raw text documents. Therefore we chose to store the local fact base, the ontology, objects and small document snippets of a few 100 bytes in a horizontally partitioned MySQL cluster database on a raid-based 256 GB solid state disc (SSD) array. The technology allows GoOLAP users to retrieve as well as to update and rate facts. In addition, the current user base may generate interactions that trigger the retrieval of several hundred thousand pages daily. We manage the raw textual documents in a Hadoop cluster (see Section 5 for an introduction and for alternative techniques). The RDBMS references these documents via unique keys. GoOLAP processes the massive amounts of requests for new facts in a parallel fashion on a cluster of 12 nodes, each running with 4 GB RAM on a 2x 2.8 GHz Intel CPU. The parallel query execution engine bases on Hadoop. We chose to abstract the functional programming interface of Hadoop through the declarative query language JAQL (see also Chapter 5) and extend JAQL with first-order functions for keyword generation, Web text filtering and fact extraction.

10.4 Demonstration Scenario

Our interactive online demo shows the interplay of users exploring the GoOLAP fact base and thereby triggering new requests for retrieving missing facts from the index of a Web search engine. Figure 61 shows our demonstration scenario. Our prototype can be reached at www.goolap.info.

Consider an analyst who wants to have an overview over competitors of Boeing. On the GoOLAP start page, she begins to type "Boeing" into the search field (1). The auto complete dropdown returns multiple interpretations of Boeing, picking the company object by default. After sending the query, the following page presents an overview over augmented facts of different types (e.g. business location, products or employment) for the company Boeing and some arbitrary documents that include information about that object. She now opens the table Business:Competitor that shows the top 15 ranked facts about competing companies (2). We suppose she is unsure whether EADS is a competitor of Boeing, thus she clicks on the document symbol on the right of the row to prove evidence for that fact. The next page shows a document from www.defenseprocurementnews.com which explains "Boeing and EADS are in direct competition" (3). She can help ranking the fact by clicking on the agree button or find the sources of other facts in the table the same way. Next, the analyst desires to compare the competitors. She clicks on the button in the column header of the competing companies to expand the list. The view presents a tabular list of competitors of Boeing (4). She can manually remove an incorrect object in the list or add another one that is missing. Suppose the analyst wishes to compare the employee size of the companies. She performs an outer join on the table by adding the column Business: Employee Size from the dropdown menu in order to get a comparable result of facts (5). If there is a wrong result in one cell, she can display alternative values and find a better one. Again, she can trace back the evidence of a fact by clicking the document symbol. She is now happy with her result and saves the table under the name "Aviation Companies". Now, she can continue her research at a later time or share the table to other users, allowing them to make their own changes.

11 Trends and Research Challenges

The unifying theme in this book is the concept of the query intention. In the first part of this book we introduced algorithms for answering navigational and informational lookup queries in a corporate setting. In the second part we presented novel query processing strategies on text that go beyond the simple paradigm of simple informational lookup queries. Finally, in the last chapter we presented GoOLAP, a novel research tool for exeucting a complex querying process, including aggregating, joining and verifying factual information. This research tool involves the creation of new interfaces that move this process beyond predictable fact retrieval.

In the near future, many analytical applications, such as GoOLAP, and a multide of optimizations for novel search tasks may evolve. These applications will pose exiting research questions to the community, such as:

What are fundamental novel search and exploration tasks for which both, the Business Intelligence community and the Search community, should design suitable operators and abstraction mechanisms?

How can we incorporate these techniques into a commercially successful marketplace on which partners can trade data and data entichment algorithms with the goal to design entirely novel, disruptive applications for analytsis and other profesionals?

This chapter concludes this book with our vision about disruptive technologies and a commercialy effective marketplace that provides the nessecary infrastructure for these novel data exploitation tasks.

11.1 Towards Exploratory Search for the Masses

Applications, such as GoOLAP may evolve over time into a comprehensive, effective and valuable information source. For reaching this goal, such applications elegantly utilize user interactions, generated selective keyword queries and apply text filtering techniques to populate an incrementally improving fact base.

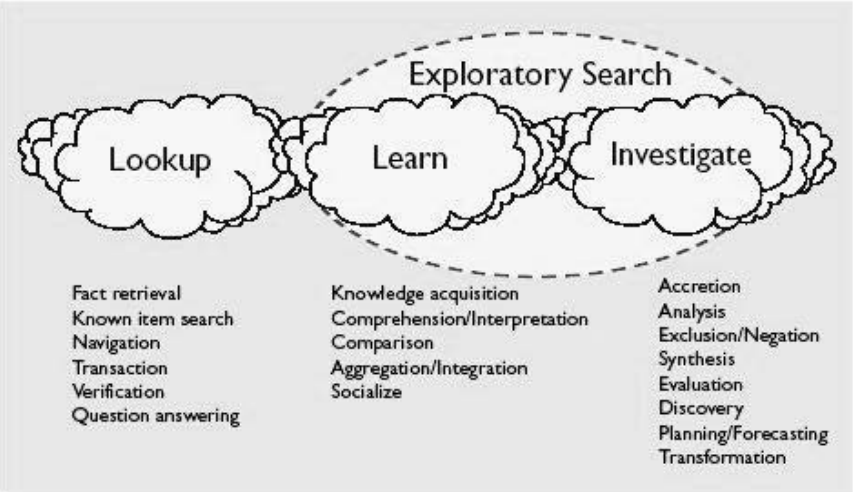


Figure 63: Bloom stated in his taxonomy of educational objectives [115] the following tasks for learning searches: knowledge acquisition, comprehension of concepts or skills, interpretation of ideas, and comparisons or aggregations of data and concepts. Investigative searches aim to achieve Bloom's highest-level objectives, such as analysis, synthesis, and evaluation and require substantial extant knowledge. (Source: [104]).

The search community has started to generalize and structure complex search tasks, like knowledge aggregation, comparision, evaluation and analysis, into the concept space of so called exploratory search [104]. Exploratory (Web) search is a relatively new research area that combines techniques from databse systems, artificial intelligence, information retrieval, text mining and human computer interactions. Authors of [104] list three classes of search activities (see Figure 63): The first class are *lookup searches* which lend themselves to formalized turn-taking where the information seeker poses a query and the system does the retrieval and returns results. Thus, the human and the system take turns in retrieving the best result.

The same authors suggest *learningto search* and *investigative searching* as two new broad classes of search activities, since these activities require strong human participation in a more continuous and exploratory process.

Figure 63 shows common sub tasks for each class. We will now review selected tasks for learning and investigative searches in detail.

11.1.1 Searching to Learn

Learning searches involve multiple iterations and return sets of objects (graphs, or maps, texts, videos) that require cognitive processing and interpretation. These objects often require the information seeker to spend time scanning/viewing, comparing, and making qualitative judgments. Therefore, much of the search time in learning search tasks is devoted to examining and comparing results and reformulating queries to discover the boundaries of meaning for key concepts (see also [104]).

Learning search tasks are best suited to combinations of browsing and analytical strategies, with lookup searches embedded to get one into the correct neighborhood for exploratory browsing, for example:

Learn all about a 'thing'. This class of scenarios is about collecting factual information about a certain thing and integrating this factual information into a common universal relation. It is a common scenario in intelligence applicatons, such as GoOLAP.info.

11.1.2 Investigative Searches

Investigative searches often include explicit evaluative annotation that also becomes part of the search results. Investigative searching may be done to support planning and forecasting, or to transform existing data into new data or knowledge. In addition to finding new information, investigative searches may seek to discover gaps in knowledge (for example, "negative search") so that new research can begin or dead-end alleys can be avoided. Investigative searches also include alerting service profiles that are periodically and automatically executed (see also [104]).

Investigative searching is more concerned with recall (maximizing the number of possibly relevant objects that are retrieved) than precision (minimizing the number of possibly irrelevant objects that are retrieved) and thus not well supported by today's Web search engines that are highly tuned toward precision in the first page of results. This explains why so many specialized search services are emerging to augment general search

engines. Because experts typically know which information resources to use, they can formulate precise analytical queries but require sophisticated browsing services that also provide annotation and result manipulation tools. The following task is a particluary interesting class of examples for investigative search.

Estimate the value of a 'thing', compare the value of 'things': In this scenario experts collect and measure signals from various data sources for estimating key indicators, which describe the value of an immaterial or material good. Common sources include public data sources, such as the Web or 'Open Data' from the UNO, non-public sources, such as commercial data from Reuters and in-house private data, such as data from an ERP or CRM system:

1. ***Ranking influencers on the Social Web.*** *Which top-10 Web forums influence buying decisions of German BMW motorbike customers? What are the most influential users?*
2. ***Value of a Web page for placing advertisements.*** *On which highly ranked and frequently visited Web pages for 'men's underwear' should I buy advertisement space?* [5]
3. ***Value of a starlet or impact of a politician.*** *Order politicians of the German parliament by the number of mentions in major German newspapers. Does extennis star 'Steffi Graf' reach the right customer set for an advertisement campaign for fast food products?*[6]
4. ***Value of a product.*** *What are top-10 journals for 'my' life science publications? A banker will issue question like: Is the product of a loan requesting company really accepted by their customers? What opinions do their customers state about the product?*
5. ***Value of medical treatments for cancer-patients.*** *Consider an alarmed patient that may seek an (adhoc) self- diagnosis: Search for*

---

[5] See also alexa.com for search engine optimization.

[6] A prominent data collector for answering such queries is the page 'Klout.com'

cancer-types on the forum "www.krebsforum-fuer-angehoerige.de". Group cancer types by frequency for female patients between 50 and 60 years. Filter forum contributions by region = "Europe, Germany, Saxony"

6. ***Correlation query of product combinations for grocery shop owner.*** *"How many customers in the forum "BlogSpot.com" combine wine from "Rioja" with a cheese of the quality "semi-curado"? List combinations by quality class of wine and its age. Project results to wines "younger" than 1996.*

In the queries above, 'fact tables' contain measurable key indicators from publicly available Web data, while 'dimension tables' may contain data from commercial vendors or in house private data, such as product lists or location information. Note, that in this scenario the 'schema' of these dimension tables is rather formalized through the schema of these commercial or in house data sources. All six example queries aggregate key indicators from the Web and correlate these key indicators with a monetary transaction, such as decisions on costly marketing measures. Eventually, users only require a partial order of the value of objects. Therefore, users are — to a certain extend — willing to tolerate uncertain data samples, such as sentiment and factual data extracted from textual sources.

Another common class for investigative search is *prior art search* within the process of creating new intellectual property, such as writing patents or publications.

11.2 A Disruptive Change: Marketplaces for Analytics

A grand challenge for both communities, the Search community and the Business Intelligence community, is conducting research on infrastructures that enables the ordinary Web user formulating such exploratory search queries or enables her interacting with results. In order to answer such queries, data sets of potential very large size need to be processed and large IT infrastructures need to be built. Even though, we can reduce the problem for some queries to a data set of a few hundred GB that will fit into main memory of modern hardware, organizations that neeed to answer exploratory queries still need capable staff in their IT department who can maintain and program programms for complex in-memory multi-core infra-

structures [120]. In particular, for small and medium enterprises (SME) the associated risks with handleing such infrastructures and the missing insight of sophisticated programmable abstraction layers are a strong barrier for innovation.

Only recently, vendors of data, providers of data warehouse solutions and algorithm providers started to offer their products as platform-, software-, and data-as-a-service on so called data market places. These data markets supply analysts, business applications, and developers with meaningful data and an eased data access. Moreover, for data producers these marketplaces act as single integration platform. As a result, data marketplaces enable completely new business models with information and analysis tools as electronically tradable goods. The collective storage, analysis, and utilization of data from the Web on a central platform offers many cost savings and high innovation capabilities. Thus, especially for SMEs significant market entry barriers and impediments to innovation are eliminated. The following important questions may arise when it comes to the technical design but also the design of a successful business model:

Benficaries and demands. We envision seven types of beneficiaries that may issue queries against and may profit from the services of a data market. The figure on the next page illustrates these relationships and shows a general schema of a data marketplace for integrating public Web data with other data sources. In analogy to a data warehouse architecture, the schema includes components for data extraction, data transformation and data loading, as well as meta data repositories describing data and algorithms. In additional, the data marketplace offers interfaces for ’uploading’ and methods for optimizing, potentially complementary, black box operators with user-defined-functionality, as well as components for trading and billing these black box operators. In return, the ’vendor’ of the user-defined-function retrieves a monetary consumption (indicated by the euro symbol) from buyers. Moreover, in the case of large data volumes from the Web, the marketplace relies on a scalable infrastructure for processing and indexing data.

Understanding demands, interests and needs of the following seven different beneficiaries is crucial before we can analyze potential technological challenges:

1. Analysts/Professionals. Typical members of this group are domain experts, such as M&A experts, sales executives, product managers, brokers, marketing managers and business analysts. Currently, the most often used data exploration tools for these experts are Web search engines. To a lesser extend this group utilizes 'traditional' OLAP focused business intelligence tools. Moreover, this group utilizes office products, mostly for aggregating and summarizing results from a data exploration into a meaningful report. From a data market perspective, this group tries to benefit from the sheer endless options of combining publicly available (Web) data, commercial data (commercial sources) and private (enterprise) data. To do so, this group issues ad-hoc queries against data sources and combines data in a highly interactive way, thereby formalizing knowledge about data sources and data integration steps through their interactions with a data market place. This group is of high relevance for other participants on the data market, since analysts create a demand for frequently requested data and associated data related services through ad-hoc queries.

2. Application vendors. Often analysts may not be comfortable in expressing their demands in a formalized machine readable representation. Additionally, the task of data exploration requires many repetitive and labor intensive steps. Application vendors formalize common requirements from analysts into applications. These applications ease data access for a broader group of domain experts. Important examples are business analytics applications, mash-up applications**,** customer relationship management applications, and enterprise resource planning applications. These applications formalize common data processing knowledge into a sequence of pre-compiled queries, such as queries for selecting sources or queries for integrating and aggregating data. From a pricing perspective of a data market, pre-compiled query sequences have a high potential to create stable, continuous demands for data and services and often result in subscription based pricing models.

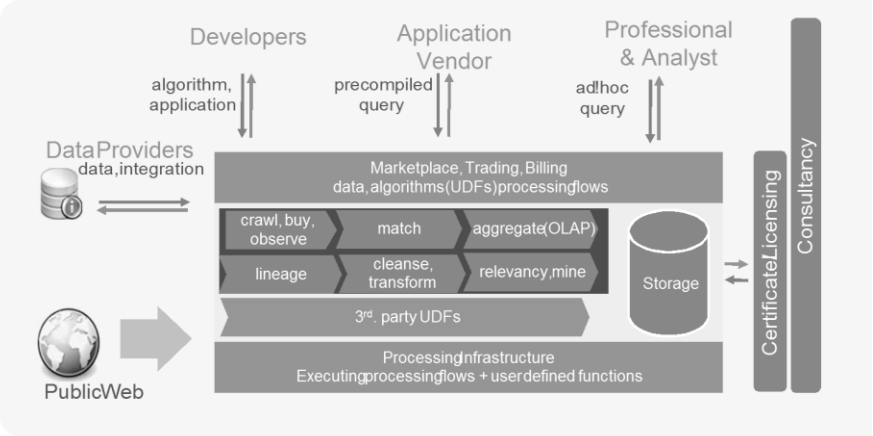


Figure 64: This figure shows a general schema of a data marketplace for integrating public Web data with other data sources. In analogy to a 'data warehouse' architecture, the schema includes components for data extraction, data transformation and data loading, as well as meta data repositories describing data and algorithms. In additional, the data marketplace owners interfaces for 'uploading' and methods for optimizing, potentially complementary, black box operators with user-defined-functionality, as well as components for trading and billing these black box operators. In return, the 'vendor' of the user-defined-function retrieves a monetary consumption from buyers. Moreover, in the case of large data volumes from the Web, the marketplace relies on a scalable infrastructure for processing and indexing data.

3. Developers of data associated algorithms. Both previous beneficiary types, application vendors and analysts, need to integrate data, often from a large and volatile set of unkown sources. Examples are algorithms for data mining, matching, cleansing, relevance computation and lineage tracing. These algorithms vary for different data domains, text data languages, data 'quality' or data 'joinability', and often only fit specific niche domains. Developers may upload these algorithms to a data marketplace as a black box user-defined function, so other participants of the data marketplace may 'try and buy' these algorithms. For running this business, Algorithm developers may buy 'shelf' space and may rent the infrastructure from the owner of the data marketplace.

4. Data providers. We distinguish between commercial and non-commercial providers of Web data, for example Web search engine own-

ers, such as Bing or Google, Web archive owners, providers of linked data from the Web, Web forum owners who seek commercialization of their forum conten, government agencies, like the UNO or the World Bank, which provide statistics free of charge or commercial data providers, such as Reuters or Bloomberg among others, which have a long history in the area of selling financial data and geo-data. Data providers utilize market places for storing and advertising data. Some data providers also started offering data integration algorithms.

5. Consultants. Consultancy services support analysts in tasks like data source selection, integration, evaluating and product development.

6. Licensing and certification entities. On a data market this group assigns a 'branded label' to data, applications and algorithms that may help customers when buying data related products.

7. Data market owner. The owner of a data market place is faced with many technical, ethical, legal and economic challenges. Probably the most important economic challenge is establishing a trusted brand and a large community. An important technical challenge is developing a common plattform for storing, searching and exchanging data and related algorithms, while minimizing algorithm execution time. In particular in Europe, legal aspects may represent another significant market entry barrier for trading data across European countries.

11.3 Challenges for the Business Intelligence Community

Which research challenges may arise from the combination of data and data associated services in data markets? In this section we present attractive research opportunities for the Business Intelligence community.

11.3.1 Authentic Information from Text Data Sources

The first trend is a growing number of data providers, such as forum owners that seek commercializing user-generated-content. Currently, these parties usually sell advertisement space. In the future, these parties will also sell raw and homogenized content or even enrich and aggregate content, thereby allowing for analytical insights. Governmental offices, such as data.gov.uk or www-genesis.destatis.de are another interesting example data

provider. In the past, these organizations offered publicly available reports, often as printouts. Only recently, these organizations have started to offer online analytics. Finally, publicly funded initiatives, such as multimillion projects triggered by the European Union or financed by venture capital, will collect and aggregate data and will seek options for commercializing data. Examples are a www.mia-marktplatz.de, an online market place for trading data of the German Web or datamarket.com, a company that mainly collects, integrates, aggregates, and visualizes public available data from governmental institutions.

However, not all information is authentic or trustworthy. In particular on the Web, certain users may exert adversarial behavior by spreading rumors or misinformation either deliberately or unconsciously, which makes traceability of information as well as cross-validation and certification a crucial challenge for any exploratory search application. Another challenge is the typical *Zipf* effect. For certain events information is plentiful. However, for others there is little or no information available until a specific deadline. Thus, the available information is based on rumors, deliberately placed information or misinformation and may be repeated in various slightly modified versions by multiple sources.

Challenge 1: Create information extraction systems that return structured, semantically meaningful and authentic data from zipfian distributed and noisy text data on the Web. Enable an analyst to conduct OLAP operations on such data with the same simplicity as a Web search.

11.3.2 Data Markets offer the entire Stack for Niche Domains

Recent technological advances, such as sophisticated implementations of the map/reduce principle (e.g., PACT/Nephele [83]), distributed storage and querying architectures (such as Big Table/HBase [47]), or high level batch processing languages (like Pig-Latin [58]) drastically simplify the access to even large clusters of data storage and processing nodes.

The availability of this technology in the open source community 'potentially' enables each marketplace owner to host the growing number of available data sources. Therefore, competition and diversification among data markets will rise, which we consider as the second important trend.

Hence, data markets will provide not only data, but will soon start to offer data associated algorithms and data visualizations. Reaching different beneficiaries groups is another option for diversification. One example are data market places, such as infochimps.com or factual.com, which mainly focus on the group of data developers and which provide subscriptions for application programmer interfaces for this group. Past efforts have shown that building applications for niche domains requires tools for creating mash-ups and data supply chains (like DAMIA [121]). We assume that data market places will soon also serve domain specific applications to reach additional groups of analysts in areas like finance, legal& ligitation analysis, health, tax and accounting, pensions/benefit & compensation analysis, intellectual property& science or (social) media.

Challenge 2: Enable analysts to create domain specific data processing flows with little costs. Enable and optimize black-box user defined functions in these flows.

11.3.3 Fresh Data from Monitoring the Web

Brand monitoring describes the task of monitoring the appearance of labels of consumer goods and also of institutions, such as of universities, regularly on the Web. This is done towards different ends. On goal is to analyze how a brand is perceived by customers. Another goal is to react to negative comments in order to reduce the harm. In particular, the last example falls into the category of publish-subscribe patterns. Besides tradtionald OLAP queries and queries over text and structured data, companies are likely to demand such services in near realtime in the near future.

Challenge 3: Build systems that can reliably answer brand monitoring queries to indexes, with heavy read and write access, sticking to ACID constraints on a scalable infrastructure in order to enable near realtime brand monitoring.

11.3.4 Customers will demand and use Product Substitutes

Current data markets do not recognize that some products can be substituted by others. That is particulary true for data associated algorithms, such as text mining libraries for extracting factual and sentiment information from user-generated content. Particulary in these application domains, al-

gorithms are often trainied on the same corpora or implement mechanics of the same recent research paper. Hence, even though, different vendors sell these mechancis under different umbrellas, key indictators, such as precision, recall and execution time, suggest that base text mining methods are comparable effective and efficient.

Because of the different unmbrellas and marketing efforts of each vendor it remains difficult for application developers to identify appropriate data algorithms, in particular for niche data sets. Worse, analysts and developers cannot determine how a particular algorithm is different from competing products. Ideally, customers of data and associated algorithms could try out algorithms on a selected data set before buying. However, such an undergoing has only been partialy realized, and only by the research comunity. A standardization of data processing mash-ups (see also Challenge 2) would enable product substitution, which in turn would enable competition between suppliers.

Challenge 4 (example substitution): Given a list of entities and their properties, identify a set of mining algorithms that have been optimized to a very high degree for exactly this data set. Provide a user with an execution sample and recommend an algorithm with regard to execution time, precision and recall.

11.3.5 Disruptive Data Markets enforce Price Transparency

On the consumer side the main motivating factor for using data market places is to buy products or services at the right price and in the right quality from a single source. The datamarket could act as intermediary instance which provides price transparency to users. This would force data suppliers and algorithm developers to optimize their data sets and algorithms. This may also lead to an increase in customers on the market place which again is the most attractive factor for suppliers. Moreover, some technology vendors would seek to tune their products niche domains markets, where the (expected) competition is low. Therefore, price transparency can lead to a positive development of the entire market place usage and may increase the revenue for the data market operator. However, for an individual supplier this transparency would eventually cause a drop on sales,

since customers of this supplier will eventually substitute the more expensive product with a lower priced product of another supplier.

As a result, the data market could offer a wider diversity of products and technologies. Because of these obvious benefits, it is likely the owner of the data market that may establish initiatives, such as benchmark data sets, standard interfaces, or search engines for specific data mining products.

Challenge 5: Develop disruptive models that encourage data owners and algorithm providers to offer transparent prices on a marketplace.

11.3.6 Learn from your Customers

On a data market place, information about data, algorithms and customer behavior is available on a single platform. This transactional information enables a data marketplace to track and derive customer preferences, which are valuable for suppliers. Each transaction between two parties on a tradable good creates fresh and valuable pricing information. Common examples are Thomson Reuters transactions on 400 worldwide stock exchanges (including the LIBOR), but also transactions on Amazon.com, Ebay.com or Google.com.

The market place operator could sell this 'secondary', transactional data which would provide another revenue stream. On the economic side, making this information available will bring a data market place closer to a perfect market (i.e., a market on which all information is transparent and available to everyone; one market close to this is the stock market). Moving closer to a perfect market will optimize processes and prices and thus optimize the overall profits and welfare. More than anything else, a broad user base can attract suppliers who offer their data and algorithms on a data marketplace.

Challenge 6: Collect transactional data from customers. Leverage this data for solving Challenges 1-5.

12 References

[1] A. Broder: A taxonomy of web search. In SIGIR Forum 2002.

[2] N. Craswell, D. Hawking, and S. Robertson: Effective site finding using link anchor information. In SIGIR 2001.

[3] A. Dillon and B. A. Gushrowski: Genres and the WEB: Is the personal home page the first uniquely digital genre? In Journal of the American Society of Information Science, 51(2):202–205, 2000.

[4] N. Eiron and K. S. McCurley: Analysis of anchor text for web search. In SIGIR 2003.

[5] R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson: Searching the workplace web. In WWW 2003.

[6] M. Fontoura, E. J. Shekita, J. Y. Zien, S. Rajagopalan, and A. Neumann: High performance index build algorithms for intranet search engines. In VLDB 2004.

[7] D. Hawking: Challenges in enterprise search. In 15th. Australasian Database Conference, 2004.

[8] Java regular expressions. http://java.sun.com/docs/books/tutorial/essential/regex/ (Last visit 01/03/2011).

[9] I.-H. Kang: Transactional query identification in Web search. In Asian Information Retrieval Symposium, 2005.

[10] I.-H. Kang and G. Kim: Query type classification for web document retrieval. In SIGIR 2003.

[11] I.-H. Kang and G. C. Kim: Integration of multiple evidences based on a query type for web search. In Information Processing Management, 40(3):459–478, 2004.

[12] B. Kessler, G. Numberg, and H. Schütze: Automatic detection of text genre. In ACL 1997.

[13] W. Kraaij, T. Westerveld, and D. Hiemstra: The importance of prior probabilities for entry page search. In SIGIR 2002.

[14] U. Lee, Z. Liu, and J. Cho: Automatic identification of user goals in web search. In WWW 2005.

[15] Y. Li, R. Krishnamurthy, S. Vaithyanathan, and H.V. Jagadish: Getting work done on the web: Supporting transactional queries. In SIGIR 2006.

[16] D. E. Rose and D. Levinson: Understanding user goals in web search. In WWW 2004.

[17] A. Schwartz and M. Hearst: A simple algorithm for identifying abbreviation definitions in biomedical texts. In Pacific Symposium on BioComputing, 2003.

[18] T. Upstill, N. Craswell, and D. Hawking: Query-independent evidence in home page finding. ACM Trans. Inf. Syst., 21(3):286–313, 2003.

[19] N. Craswell, D. Hawking, and S. Robertson: Effective site finding using link anchor information. In SIGIR 2001.

[20] H. Cunningham: Information extraction -a user guide. Technical Report CS-97-02, University of Sheffield, 1997.

[21] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu: Avatar semantic search: a database approach to information retrieval. In SIGMOD 2006.

[22] I.-H. Kang and G. C. Kim: Integration of multiple evidences based on a query type for web search. In Information Processing Management, 40(3):459–478, 2004.

[23] W. Kraaij, T. Westerveld, and D. Hiemstra: The importance of prior probabilities for entry page search. In SIGIR 2002.

[24] J. F. McCarthy and W. G. Lehnert: Using decision trees for coreference resolution. In IJCAI 1995.

[25] K. Nanda: Combining lexical, syntactic and semantic features with maximum entropy models for extracting relations. In ACL 2004.

[26] A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic: Searching the web: the public and their queries. J. Am. Soc. Inf. Sci. Technol., 52(3):226–234, 2001.

[27] F. M. Suchanek, G. Kasneci, and G. Weikum: Yago: a core of semantic knowledge. In WWW 2007.

[28] T.S.Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu: Avatar information extraction system. IEEE Data Engineering Bulletin, May 2006.

[29] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne: Finding high quality content in social media, with an application to community-based question answering. In WSDM 2008.

[30] R. Baeza-Yates and B. Ribeiro-Neto: Modern Information Retrieval. Addison Wesley, May 1999.

[31] A. Z. Broder: Identifying and filtering near-duplicate documents. In COM 2000.

[32] M. S. Charikar: Similarity estimation techniques from rounding algorithms. In STOC 2002.

[33] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe: Collection statistics for fast duplicate document detection. ACM Trans. Inf. Syst., 20(2):171–191, 2002.

[34] X. Z. Fern and W. Lin: Cluster ensemble selection. In SDM, pages 787–797. SIAM, 2008.

[35] A. Gionis, P. Indyk, and R. Motwani: Similarity search in high dimensions via hashing. In VLDB 1999.

[36] M. Henzinger: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In SIGIR 2006.

[37] U. Manber: Finding similar files in a large file system. In USENIX 1994.

[38] R. Ramakrishnan and A. Tomkins: Towards a people web. In IEEE Computer, 40(8):63–72, 2007.

[39] M. Theobald, J. Siddharth, and A. Paepcke: Spotsigs: robust and efficient near duplicate detection in large web collections. In SIGIR 2008.

[40] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and D. Zhuang: Simfusion: measuring similarity using unified relationship matrix. In SIGIR 2005

[41] S. Ye, R. Song, J.-R. Wen, and W.-Y. Ma: A query-dependent duplicate detection approach for large scale search engines. In APWeb, pages 48–58, 2004.

[42] M. Kantardzic: Data Mining: Concepts, Models, Methods, and Algorithms Wiley Interscience 2003.

[43] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, O. Etzioni: Open Information Extraction from the Web. In IJCAI 2007

[44] D. Burdick, P. M. Deshpande, T.S. Jayram, R. Ramakrishnan, S. Vaithyanathan: OLAP Over Uncertain and Imprecise Data. In VLDB Journal, Volume 16, No. 1, January 2007

[45] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, J. Zhou: SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. In VLDB 2008

[46] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, R. Yerneni: PNUTS: Yahoo!'s Hosted Data Serving Platform. In VLDB 2008

[47] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber: Bigtable: A Distributed Storage System for Structured Data. In OSDI 2006

[48] M. Cafarella, D. Suciu, O. Etzioni: Navigating Extracted Data with Schema Discovery. In WebDB 2007

[49] T. Cheng, X. Yan, K. Chen-Chuan Chang: EntityRank: Searching Entities Directly and Holistically. In VLDB 2007: 387-398

[50] J. Dean, S. Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. In OSDI 2004

[51] P. DeRose, W. Shen, F. Chen, A. Doan, R. Ramakrishnan: Building Structured Web Community Portals: A Top-Down, Compositional, and Incremental Approach. In VLDB 2007

[52] D. Ferrucci, A. Lally: UIMA: an architectural approach to unstructured information processing in the corporate research environment. In Natural Language Engineering Volume 10, Issue 3-4, September 2004

[53] Gartner Executive Programs CIO Survey, January 10, 2008

[54] S. Ghemawat, H. Gobioff, S. Leung: The Google file system. In SOSP 2003

[55] T. Götz, O. Suhre: Design and implementation of the UIMA Common Analysis System. In IBM Systems Journal 2004, Vol. 43, No. 3

[56] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly: Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In EuroSys 2007

[57] G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum: NAGA: Searching and Ranking Knowledge. In ICDE 2008

[58] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins: Pig Latin: A Not-So-Foreign Language for Data Processing. In SIGMOD 2008

[59] J.M. Pérez, R. B. Llavori, M. J. Aramburu, T. B. Pedersen: In Integrating Data Warehouses with Web Data: A Survey. In IEEE Trans. Knowl. Data Eng. 20(7): 940-955 (2008)

[60] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, S. Vaithyanathan: An Algebraic Approach to Rule-Based Information Extraction. In ICDE 2008

[61] Y. Sismanis, P. Brown, P. J. Haas, B. Reinwald: GORDIAN: Efficient and Scalable Discovery of Composite Keys. In VLDB 2006.

[62] W. Shen, A. Doan, J. F. Naughton, R. Ramakrishnan: Declarative information extraction using datalog with embedded extraction predicates. VLDB 2007

[63] M. Weis, F. Naumann, U. Jehle, J. Lufter, H. Schuster: Industry-Scale Duplicate Detection. In VLDB 2008

[64] W. Wu, B. Reinwald, Y. Sismanis, R. Manjrekar: Discovering topical structures of databases. In SIGMOD 2008

[65] H. Yang, A. Dasdan, R. Hsiao, D. Parker: Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters. In SIGMOD 2007

[66] G. Kasneci, M. Ramanath, F.M. Suchanek, G. Weikum: The YAGO-NAGA approach to knowledge discovery. SIGMOD Row 37(4): 41-47 (2008)

[67] A. Jain, A. Doan, L. Gravano: Optimizing SQL Queries over Text Databases. In ICDE 2008.

[68] A. Jain. D. Srivastava: Exploring a Few Good Tuples from Text Databases. In ICDE 2009.

[69] A. Jain, P.G. Ipeirotis, A. Doan, L. Gravano: Join Optimization of Information Extraction Output: Quality Matters! ICDE 2009.

[70] W. Shen, P. DeRose, R. McCann, A. Doan, R. Ramakrishnan: Toward best-effort information extraction. In SIGMOD 2008.

[71] E. Agichtein, L. Gravano: QXtract: a building block for efficient information extraction from Web page collections. In SIGMOD 2003.

[72] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. Saita: Declarative Data Cleaning: Language, Model, and Algorithms. Very Large Data Bases. Morgan Kaufmann Publishers, Rome, CA, 371-380.

[73] O. Etzioni, M. Banko, S. Soderland, D.S. Weld: Open information extraction from the Web. Commun. ACM 51(12): 68-74 (2008)

[74] YahooBoss service. http://developer.yahoo.com/search/boss/fees.html (Last visited 01/06/10)

[75] OpenCalais. http://www.opencalais.com/comfaq (Last visited 01/06/10)

[76] J. Liu, X. Dong, A.Y. Halevy: Answering Structured Queries on Unstructured Data. In WebDB 2006

[77] HSQLDB. http://hsqldb.org/ (Last visited 01/06/10)

[78] G. Fung, J. Yu, H. Lu: Discriminative Category Matching: Efficient Text Classification for Huge Document Collections. In ICDM 2002

[79] R. Feldman, Y. Regev, M. Gorodetsky: A modular information extraction system. Intell. Data Anal. 12(1): 51-71 (2008)

[80] Fortune 500. http://money.cnn.com/magazines/fortune/fortune500/2008/full_list/ (Last visited 01/06/10)

[81] W.B. Croft, D. Metzler, T. Strohman: Search Engines, Information Retrieval in Practice. Addison Wesley 2010: 313-315

[82] X. Dong, A. Y. Halevy, J. Madhavan: Reference Reconciliation in Complex Information Spaces. In SIGMOD 2005

[83] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, D. Warneke: Nephele/PACTs: a programming model and execution framework for Web-scale analytical processing. In SoCC 2010

[84] A. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava: Answering Queries Using Views. In PODS 1995: 95-104

[85] R. Avnur, J.M. Hellerstein: Eddies: Continuously Adaptive Query Processing. In SIGMOD 2000: 261-272

[86] V. Markl, V. Raman, D.E. Simmen, G.M. Lohman, M. Pirahesh: Robust Query Processing through Progressive Optimization. SIGMOD Conference 2004: 659-670

[87] T. Mitchell: Machine Learning. McGraw-Hill, 1997.

[88] P.G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano:. To search or to crawl?: towards a query optimizer for text-centric tasks. In SIGMOD 2006

[89] F. Naumann, F: Quality-Driven Query Answering for Integrated Information Systems. Springer 2002

[90] I. Ilyas, G. Beskales, and M.A. Soliman: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40(4): (2008)

[91] C. Bizer, T. Heath, T. Berners-Lee: Linked Data - The Story So Far. Int. J. Semantic Web Inf. Syst. 5(3): 1-22 (2009)

[92] Freebase. www.freebase.com (Last visited 01/01/11)

[93] J. Christensen, Mausam, S. Soderland, O. Etzioni: Semantic Role Labeling for Open Information Extraction. NAACL HLT 2010.

[94] C. Kohlschütter, P. Fankhauser, W. Nejdl: Boilerplate detection using shallow text features. WSDM 2010: 441-450

[95] F. Wu, D.S. Weld.: Open Information Extraction using Wikipedia. ACL 2010

[96] T. Joachims: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. ECML 1998.

[97] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R.Wang, C.-J.Lin: LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9(2008), 1871-1874.

[98] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan: SystemT: an Algebraic Approach to Declarative Information Extraction. ACL 2010

[99] D. Lewis and R.M. Tong: Text filtering in MUC-3 and MUC-4. MUC 1992

[100] R. Grishman, S. Huttunen, R. Yangarber: Information extraction for enhanced access to disease outbreak reports. Journal of Biomedical Informatics 35(4): 236-246 (2002)

[101] E. Riloff: Automatically generating extraction patterns from untagged text. AAAI, 1996: 1044-1049

[102] S. Chakrabarti, S. Sarawagi, S. Sudarshan: Enhancing Search with Structure. IEEE Data Eng. Bull. 33(1): 3-24 (2010)

[103] D. Crow. Google Squared: Web scale, open domain information extraction and presentation. ECIR 2010

[104] G. Marchionini. Exploratory search: from searching to understanding. Communications of the ACM, 49:41-46, Apr. 2006.

[105] T. Lin, O. Etzioni, and J. Fogarty. Identifying interesting assertions from the web. In 18th CIKM Conference, 2009.

[106] Reuters Corpus Volume I (RCV1) available at http://trec.nist.gov/data/reuters/reuters.html (Last visit 01/03/2011)

[107] T. Dunning. Accurate methods for the statistics of surprise and coincidence. Computational Linguistics. 19:61–74, March 1993.

[108] Alias-i. Lingpipe 4.0.1. http://alias-i.com/lingpipe. (Last visited 30/09/11).

[109] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008

[110] Apache Lucene. http://lucene.apache.org (Last visited 30/09/11).

[111] M. Zhou, T. Cheng, and K. C.-C. Chang. Docqs: a prototype system for supporting data-oriented content query. In *SIGMOD*, 2010.

[112] S. E. Robertson. On term selection for query expansion. *J. Doc.*, 46:359–364, January 1991.

[113] W. W. Cohen. Fast Effective Rule Induction. In *ICML*, 1995.

[114] P. Bohannon, S. Merugu, C. Yu, V. Agarwal, P. DeRose, A. Iyer, A. Jain, V. Kakade, M. Muralidharan, R. Ramakrishnan, and W. Shen. Purple sox extraction management system. SIGMOD Rec., 37:21–27, March 2009.

[115] B.S. Bloom, M.D. Engelhart, E.J. Furst, W. Hill, and D.R. Krathwohl: Taxonomy of educational objectives: the classification of educational goals; Handbook I: Cognitive Domain New York, Longmans, Green, 1956.

[116] J. Nivre: Algorithms for Deterministic Incremental Dependency Parsing. Computational Linguistics 34(4), 513-553.

[117] J. Nivre, L. Rimell, R. McDonald and C. Gómez Rodríguez: Evaluation of Dependency Parsers on Unbounded Dependencies. Coling 2010, 833-841.

[118] G. Attardi, F. Dell'Orletta, M. Simi, A. Chanev and M. Ciaramita. 2007. Multilingual Dependency Parsing and Domain Adaptation using DeSR. In EMNLP-CoNLL 2007.

[119] Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In LREC 2006.

[120] Antony Rowstron, Dushyanth Narayanan, Austin Donnelly, Greg OShea, and Andrew Douglas. Nobody ever got fired for using hadoop on a cluster. In HotCDP 2012 - 1st International Workshop on Hot Topics in Cloud Data Processing, 2012.

[121] David E. Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: data mashups for intranet applications. In SIGMOD Conference, 2008.

[122] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis: Dremel: interactive analysis of web-scale datasets. Commun. ACM 54(6): 114-123 (2011)

13 About the Author

Dr.-Ing. Alexander Löser joined the Technische Universität Berlin as senior scientific staff member in 2008. He lectures a data mining and business intelligence class. Before, Alexander conducted research on the IBM search engine project AVATAR for emails and the IBM intranet as a Postdoc at the IBM Almaden Research Center. This project was later transferred into System-T and Lotus Notes. In 2008, Alexander joined the research division of SAP AG for two years as senior research scientist and project manager. He investigated techniques for searching and analyzing user generated content for the SAP Software Developer Network for more than one million SAP customers.

Alexander holds a PhD from Technische Universität Berlin in the area of routing strategies of peer-to-peer networks and a Master Degree on Business Computer Science. Alexander has more than 14 years experience in industrial and academic research. He published more than 25 papers on peer-reviewed conferences, international journals and holds 4 patent applications. In 2000 his work for the interactive, large scale image search engine 'Virtual Design' received the Special Award of Multimedia Design. Later, his work on the Avatar Semantic Search System with IBM Research was acknowledged in the Computer World Horizon Awards 2006. In 2011 he received an award from the Federal Minister of Economics and Transportation, Rainer Brüderle, for his project idea "MIA – A market place for Information and Analytics for Data from the German Language Web".

Since 2013, Alexander is faculty member of the Beuth-Hochschule für Technik Berlin. His current research focuses on Web-scale Analytics, in particular over unstructured content and social media, in highly parallel execution environments. Moreover, he investigates technical and economic research questions of the paradigm 'data-a-as-service'. Learn more about Alexander's recent work at www.GoOLAP.info or www.datexis.com .

13.1 Relevant Publications of the Author

[123] Alan Akbik and Alexander Löser: **N**-Ary Facts in Open Information Extraction. AKBC-WEKEX, ACL 2012

[124] Alexander Löser, Sebastian Arnold, Tillmann Fiehn: The GoOLAP Fact Retrieval Framework. LNBIP 96/2012

[125] Alexander Löser, Christoph Nagel, Stephan Pieper, Christoph Boden: Beyond Search: Retrieving Complete Tuples from a Text-Database. Accepted for Information System Frontiers Journal (journal impact factor 1,5)

[126] Christoph Boden, Alexander Löser, Christoph Nagel, Stephan Pieper: FactCrawl: A Fact-Retrieval Framework for Full Text Indices. WebDB 2011 (acceptance rate 27%, 47 Submissions)

[127] Alexander Löser, Christoph Nagel, Stephan Pieper, Christoph Boden: Self-Supervised Web Search for Any-k Complete Tuples BeWeb@EDBT/ICDT

[128] Christoph Boden, Thomas Häfele, Alexander Löser: Classification Algorithms for Relation Prediction. DaLi@ICDE 2011

[129] Alexander Löser, Christoph Nagel, Stephan Pieper: Augmenting Tables by Self-Supervised Web Search. BIRTE@VLDB 2010

[130] Alexander Löser, Steffen Lutter, Patrick Düssel, Volker Markl: Ad-hoc queries over document collections - a case study. BIRTE@VLDB 2009

[131] Klemens Muthmann, Alexander Löser: Detecting Near-Duplicate Relations in User Generated Forum Content. OTM Workshops 2010: 698-707 2009

[132] Klemens Muthmann, Wojciech M. Barczynski, Falk Brauer, Alexander Löser: Near-duplicate detection for web-forums. IDEAS 2009: 142-151 (acceptance rate 24%, 210 Submissions)

[133] Alexander Löser: Beyond Search: Web-Scale Business Analytics. WISE 2009: 5 2008 (Keynote)

[134] Falk Brauer, Marcus Schramm, Wojciech M. Barczynski, Alexander Löser, Hong Hai Do: Robust recognition of complex entities in text exploiting enterprise data and NLP-techniques. ICDIM 2008: 551-558

[135] Alexander Löser, Wojciech M. Barczynski, Falk Brauer: What's the Intention Behind Your Query? A few Observations From a Large Developer Community. IRSW 2008

[136] Alexander Löser, Gregor Hackenbroich, Hong-Hai Do, Henrike Berthold: Web 2.0 Business Analytics. Datenbank Spektrum, 25/2008 (Invited Paper)

[137] Alexander Löser, Fabian Hueske, Volker Markl: Situational Business Intelligence. BIRTE@VLDB 2008: 1-11 (Keynote)

[138] Falk Brauer, Alexander Löser, Hong Hai Do: Mapping enterprise entities to text segments. PIKM 2008: 85-88 2007

[139] Karen Walzer, Jürgen Anke, Alexander Löser, Huaigu Wu: A Concept for Flexible Event-Driven Invocation of Distributed Service Compositions. ICDCS Workshops 2007: 64

[140] Karen Walzer, Alexander Schill, Alexander Löser: Temporal constraints for rule-based event processing. PIKM 2007: 93-100

[141] Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan, Alexander Löser: Navigating the intranet with high precision. WWW 2007: 491-500 (acceptance rate 11.6%, 612 Submissions)

13.2 Relevant Talks and Tutorials of the Author

[142] Alexander Löser: Unstructured information management. Hasso Plattner Institut Berlin 2007 (Talk)

[143] Alexander Löser: Business Intelligence over Text in the Cloud. Freie Universität Berlin 2009 (Talk)

[144] Alexander Löser: Scalable Ad-Hoc Reporting from Text Data. Humboldt Universität Berlin 2009 (Talk)

[145] Alexander Löser: Ad-hoc Reporting from Web-based Text Data. SAP / BOBJ Paris 2009 (Talk)

[146] Alexander Löser: Ad-hoc Analyse unstrukturierter Web-2.0-Daten“. 13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW), 2009 (Tutorial)

[147] Alexander Löser: BI-over-Text: Analyzing user-generated data using cloud computing techniques. Dagstuhl Seminar on Social Web Communities 2009.

[148] Alexander Löser, Volker Markl: Cloud Computing: Technologien, Anwendungen und Geschäftsmodelle. Deutsche Informatik Akademie 2009. (Talk)
[149] Alexander Löser: Web 2.0 Business Analytics: Qualität durch Datenmenge? Informatiktage 2010 in Bonn. (Keynote)
[150] Alexander Löser, Volker Markl: Cloud Computing: Technologien, Anwendungen und Geschäftsmodelle. Deutsche Informatik Akademie 2010. (Talk)
[151] Alexander Löser: Information Management in der Cloud. Tutorial auf der 14. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW), 2011
[152] Alexander Löser: Beyond Search: Interactive Web Analytics. Dagstuhl Seminar on Challenges in Document Mining 2011.
[153] Alexander Löser: Web Scale Business Analytics. 1st. European Business Intelligence Summer School (eBISS) 2011. (Tutorial)
[154] Alexander Löser: Marketplaces for Interactive Web Analytics. IBM DB2 Community Days 2011. (Talk)

13.3 Relevant Patents and Applications of the Author

[155] Falk Brauer, Hong-Hai Do, Wojciech Barczynski, Alexander Löser, Marcus Schramm: Graph-based Re-Composition of Document Fragments For Named Entity Recognition Under Exploration Of Enterprise Databases. Granted 24.7.2012 as US8229883 B2
[156] Alexander Löser, Wojtiech Barzcyinski, Falk Brauer: Systems and methods for modular information extraction. Granted 26.7.2011 as US7987416 B2
[157] Huiyu Zhu, Alexander Löser, Sriram Raghavan and Shiv. Vaithyanathan: Searching navigational pages in an intranet. DATE: 03-12-2009.
[158] Alexander Löser, Sriram Raghavan, Shivakumar Vaithyanathan: Method and system for automatically generating regular expressions for relaxed matching of text patterns. Granted 9/7/2013 as US 8484238 B2.