Distributed Deep Web Search

Kien Tjin-Kam-Jet

ISBN: 978-90-365-3564-9

Distributed Deep Web Search

Kien Tjin-Kam-Jet

You are invited to attend the public defence of my thesis

Distributed Deep Web Search

preceded by an introduction on

December 19th 2013
at 16:30 in
Waaier, room 4

University of Twente

Distributed Deep Web Search

Kien Tjin-Kam-Jet

Samenstelling van de promotiecommissie:

Prof. dr. P.M.G. Apers, promotor
Prof. dr. F.M.G. de Jong, promotor
Dr. ir. D. Hiemstra
Dr. ir. R.B. Trieschnigg
Prof. dr. W. Meng, Binghamton University, New York
Prof. dr. G.J.M. van Noord, Rijksuniversiteit Groningen
Prof. dr. T.W.C. Huibers
Prof. dr. V. Evers

CTIT Ph.D. Thesis Series No. 13-273
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, The Netherlands.

SIKS Dissertation Series No. 2013-34
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 978-90-365-3564-9
ISSN: 1381-3617 (CTIT Ph.D. Thesis Series no. 13-273)
DOI: 10.3990./1.9789036535649
http://dx.doi.org/10.3990/1.9789036535649

Cover design: Kien Tjin-Kam-Jet
Printed by: Gildeprint

DISTRIBUTED DEEP WEB SEARCH

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 19 december 2013 om 16:45 uur

door

Kien-Tsoi Theodorus Egbert Tjin-Kam-Jet

geboren op 5 augustus 1982
te Oranjestad, Aruba

Dit proefschrift is goedgekeurd door:

prof. dr. P.M.G. Apers (promotor)
prof. dr. F.M.G. de Jong (promotor)
dr. ir. D. Hiemstra (assistent-promotor)

OneBox to structure them all,

OneBox to find them.

OneBox to query them all,

and in transparency bind them.

For A-Jien, Romeo, and Kam-Fong

Preface

It is not customary for me to receive phone calls in a supermarket, but in 2009, I received a call telling me that I got the job as a PhD researcher. Looking back, I still wonder how I managed to keep all groceries in the basket as I literally jumped up in all excitement.

The project I was going to work on focused on distributed search and in particular on including deep web data in the search process. The project was full of challenges and I am grateful for the opportunity for doing this research. I have long been fascinated by the quickness and effectiveness with which all sorts of information can be retrieved from a digital data store. This fascination is perhaps due to the many success stories of information retrieval, from relational databases empowering corporate companies around the world, to inverted-indices driving millions of our daily information needs on the web. Yet web search, which can be regarded as the biggest success story, is far from a finished product. This thesis shows some of the shortcomings of current web search, but more importantly, it shows promising directions for dealing with these shortcomings.

The web is full of (not-so-)stylish websites, (ir)relevant information, and (very) complex web forms, and, unless someone comes along and tells us otherwise, we simply take the hassle of using those complex web forms for granted. That people crave for easier ways of searching through complex web forms became evident after launching the "Treinplanner", which received a lot of media attention through Twitter, Facebook, local radio and even national television. I guess that this is one of the things which makes doing research so worthwhile, the understanding that it can make the life of people easier.

I hope those who read this thesis will get a better understanding of the issues of and opportunities for improving deep web search.

Acknowledgements

Finishing a PhD thesis is not something that you achieve on your own. That's why I want to grasp this opportunity to thank everyone who supported me during these past four years and I would like to mention some of them in particular.

First off all, my thanks go to my daily supervisors, Djoerd Hiemstra and Dolf Trieschnigg. This thesis would not have been written without the valuable input of you both, which definitely raised this end-product to a higher level. Djoerd, thank you for your trust in me when you offered me this PhD position. It was a very valuable experience. Your knowledge of, practical approach to, and dedication to research in information retrieval certainly stimulate many researchers and I am grateful that I was in the position to learn from that. Dolf, your interest and enthusiasm worked very inspiring and you continuously stimulated me to get to the bottom of things. You always had (or otherwise made) time for a good critical discussion, and yes, we had many of these over the years. They were without a doubt very helpful. I admire your vigorousness and the way you can look at things from different perspectives.

Next, I would like to thank my promotors and committee members. Peter, thank you for steering the work in the right direction and in particular for firing up the engine. Franciska, thank you for your valuable feedback on my thesis and for helping me with the finishing touch. I feel honored that Weiyi Meng, Gert-Jan van Noord, Theo Huibers, and Vanessa Evers all agreed to take place in my dissertation committee. Thank you very much.

Also, I would like to thank all those who participated in my user-studies. Without their help I would not have been able to collect so much data. Special thanks to Han Mooiman for his valuable support in bringing about the Treinplanner demo. I really appreciate the willingness of people to make a contribution to scientific studies.

Speaking of support, this research would not have been possible without the funding of the *Nederlandse Organisatie voor Wetenschappelijk Onderzoek* (NWO).

Next, a word of thanks to all my colleagues from the database group: without you those four years would not have been so nice. You created a good working environment and a friendly atmosphere with room for serious conversations as well as for fun. I really enjoyed the lunches we had together, the Friday after-

noon jokes, and the occasional times we played a game of Atlantis. Especially how we warned all newcomers to watch out for the “oops” moments and then everyone asked “what are those exactly?” and then... “oops”. I have also seen several roommates come and go during those past four years, but I appreciated the company of each of you. Between all the hard work we had some nice conversations and discussions. Of course, there were also the nice conferences, and I have seen plenty of empirical evidence at these scientific gatherings that science, creativity, and beer can go well together. A special thanks should go to Ida, and to Suse, who did make life easier. You always stood by to assist with the flight and hotel reservations (even for my girlfriend when she was traveling with me). I appreciate your support throughout the years.

Then last, but certainly not least, I would like to thank my family, family-in-law, and friends for the interest they showed in my work and the support they offered me. There are some people who I would like to mention in particular. Mom and dad, thanks for your relentless support and love. You always stood beside me. Though my dad is no longer with us, I know he would have been proud. Thank you for always believing in me and for giving me the opportunity to study in the Netherlands. I would also like to express my gratitude to Kam-Fong and Arnout, for the nice weekends filled with relaxation and eating, and to Wolter, for simply being a very inspiring person. I am also grateful for the love and support from my father- and mother-in-law, Charles and Reini. Thank you, for your constant interest in my work, the fruitful discussions, and for being there for me.

Now, I would like to thank the one dearest to me. Honestly, I can’t imagine what it would have been like without the unconditional love and full support of my girlfriend. Liseth, I cannot thank you enough.

Enschede, November 2013

Contents

Chapter 1

Introduction

"The only thing that is constant is change -"
– *Heraclitus*

This thesis introduces a new method for searching dynamic and structured content across multiple websites and domains. This chapter provides an outline of current web search practices, identifies problems of current web search technologies, and presents the research questions that will be answered throughout the remaining chapters.

1.1 The web search landscape

The World Wide Web, also referred to as the web[1], has radically changed the way in which we produce and consume information. Notably, it contains billions of documents which makes it likely that *some* document will contain the answer or content you are searching for. The web has been growing at a tremendous rate and is in a constant state of flux: some documents change over time, some just disappear completely, and yet others are newly created. To give an idea of how the web has grown over the last decade: in 1999, it was estimated that the web consisted of 800 million web pages and that no web search engine indexed more than 16% of the web (Lawrence and Giles, 2000); in 2005, the web was estimated at 11.5 billion pages (Gulli and Signorini, 2005); in 2008, Google announced[2] the discovery of one trillion (1,000,000,000,000) unique URLs on the web; and in 2013, Google updated this number to 30 trillion[3]. The immense size of the web, its continuous growth, and its highly dynamic nature, make it challenging to build a web search engine that is effective, fast, and that can scale up to web proportions (Baeza-Yates et al., 2007). It is difficult to assess to what extent major search engines like Bing and Google actually keep up with the growth of

[1] `http://en.wikipedia.org/wiki/Www` (April 16^{th} 2013)
[2] `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html` (May 1^{st} 2013)
[3] `http://www.google.com/insidesearch/howsearchworks/thestory/` (May 14^{th} 2013)

the web; however, they often do manage to return satisfying results within just a few milliseconds.

The driving force behind a web search engine is its inverted index, which works much like the index in the back of a book: for each word, it contains a list of pages in which the word occurs[4]. Before a search engine can build its index, it first downloads and analyzes many web pages using a program called a *crawler*. Essentially, a crawler downloads web pages by following hyperlinks. That is, it first downloads a predefined set of (popular) web pages. Then, it scans the downloaded pages for new hyperlinks to other pages and downloads the other pages, and so on. While inspecting the crawled web pages, the search engine keeps track of which words appear on which pages and builds up its index. However, not all web pages can be crawled or found by following hyperlinks. For example, if no web site links to a particular page, then that particular page cannot be found by crawling. Traditionally, the part of the web that can be crawled is referred to as the *visible web* or *surface web*, and the part that cannot be crawled is referred to as the *invisible web* or *hidden web* (Bergman, 2001; Florescu et al., 1998). A further distinction is often made in the hidden web: those pages that are retrieved by submitting a web form are referred to as the *deep web* (Bergman, 2001; Madhavan et al., 2008; Chang et al., 2004a).

We wish to point out that the terms *invisible web*, *hidden web*, and *deep web* are sometimes used synonymously, denoting either the part of the web that cannot be crawled, or the pages that are accessed via web forms (Bergman, 2001; Calì and Martinenghi, 2010; Raghavan and Garcia-Molina, 2001). Also, note that the definition "*if something can be crawled, it is part of the surface web; otherwise, it is part of the deep web*" bares an ill-definition. As crawler technology advances, what is now considered as part of the "deep web" might not be considered "deep" anymore at some point in the future. So, even if the content stays the same, it could go from being "deep" to being "surface" content. The real issue is that some content can exhibit certain properties that are problematic when it comes to crawling and indexing the content.

1.1.1 Problems with crawling and indexing

Crawlers automatically gather web content to be indexed. However, as we will explain below, some content cannot be crawled, and some content is not suited to be indexed. Furthermore, since the contents of a page can change over time, all indexed pages must regularly be re-crawled and re-indexed to keep the index up-to-date. We say that the content has changed when a request yields different contents upon re-issuing the same request[5]. We refer to content that rarely or never changes as *static content*; content that changes often as *dynamic content*[6]; and content that changes very frequently as *highly dynamic content*.

[4]The word may also have other relations with the document, e.g., it may also occur in anchor texts of hyperlinks pointing to the page.

[5]The content as intended here refers to the main content or information on a page, and disregards generated data such as advertisements, timestamps, or session IDs.

[6]This does not refer to multimedia content such as videos, unless they are regularly updated.

A URL (universal resource locator) is necessary to retrieve web content. URLs may contain optional parameters, which can be specified via a web form. However, web forms can either use a HTTP GET request method, or a HTTP POST request method. In short, the parameters are included as part of the URL in the case of HTTP GET, but not in the case of HTTP POST. This means that, if some pages can only be retrieved via a form that uses HTTP POST, then no one will be able to publish a URL to link to those pages (since the URL does not include all parameters); hence, those pages cannot be crawled in the conventional way of following hyperlinks. In general, content that must be retrieved via a web form bears two kinds of problems. First, the content is hard to crawl, because:

a) crawlers generally cannot fill out the necessary web forms; and,

b) if forms use HTTP POST, then there may be no URL to link to these pages.

This means that, unless additional measures have been taken, such as using *search engine friendly*[7] URLs, content behind web forms cannot be crawled by a typical search engine. Second, even if the content could be crawled (either by trying to fill out a web form or by following hyperlinks), the content itself may not be suited for indexing, because:

a) the content is highly dynamic. For example, in booking sites or shopping sites, the availability or number of items in stock may change rapidly. Also, new products are repeatedly added and old ones removed. If such content would be indexed, it should then also be frequently re-indexed;

b) the content is seemingly unlimited. For example, certain web forms like that of a web calculator or ones that convert values from one unit into another, produce results for every input. Since the input possibilities are endless, the output is also endless. Note that even without a web form, a website can still produce endless output. For example, on a website that shows a monthly calendar with a link to next month's calendar, there may be no end to the number of times you can click the link to next month's calendar. The point is that a very large or even infinite amount of data is being generated by some function. Instead of indexing the generated data, developers should rather acquire the function that generated the data, but the functions may be proprietary and not publicly available; and,

c) the content resides in a structured database and must be accessed by means of a *structured query*. A structured query is a way to represent an information need by specifying restrictions on one or more attributes of an item. For example, if an end-user is searching for an laptop that costs no more than 450 dollars, the end-user could specify the minimum price attribute and the maximum price attribute as shown in Figure 1.1. The results for

[7] `http://en.wikipedia.org/wiki/Search_engine_friendly_URLs` (April 18^{th} 2013)
Search engine friendly URLs contain all parameters as short informative texts instead of seemingly random characters. For example, instead of a URL like `http://example.com/blog.php?ext=id%3D1`, a more descriptive URL might be `http://example.com/blog/the_deep_web`.

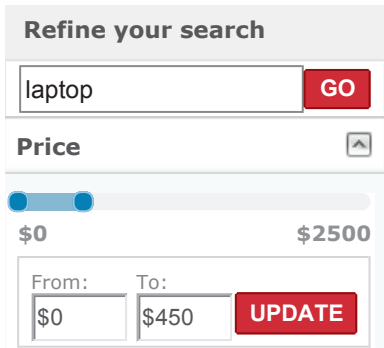


Figure 1.1: A structured query interface. In this particular example, an end-user has entered a search request for laptops with a price between 0 and 450 dollars.

> this query should only contain laptops (so no mobile phones, game consoles, or desktop computers) which are less than 450 dollars (so no popular laptop for 699 dollars). Though this example might seem obvious, a keyword based retrieval system, e.g., like a general web search engine, might return erroneous results, such as mobile phones or desktops. This is because a keyword index does not keep track of attributes, it simply returns pages that contain the "words" *laptop*, *0*, or *450*.

Why then bother indexing this kind of content if it is such a hassle? One reason is that this content generally informs about a service that is offered by a company, e.g., a rental service, a travel planning service, or an online shopping service. These services can be *highly relevant* to the end-user. Another reason is that it would be nice to have a *single-point-of-entry* to these pages. Today, if you need to compare products from different websites, you would have to re-enter the query in each website's form. This is not only tiresome, it is also easier to make a mistake. Every form is different so you must first spend some effort in analyzing and understanding the form, meanwhile, you may forget to specify one attribute (in which case you must start all over again).

Summarizing, there are two different kinds of problems when it comes to indexing web content: first, the process of getting the content can be a problem; and second, the content itself can be a problem (which can be divided in three subproblems: highly dynamic, seemingly unlimited, and result of a structured query). In the rest of this thesis, the term *deep web* will be used to refer to web pages that share one or more of these (sub)problems.

1.1.2 Query types: keyword, structured, and free-text

Web search engines generally retrieve documents containing as many of the keywords in the query as possible. Though it could matter whether a keyword occurs in the title, in the introduction, or near the document's end, users generally cannot influence these structural aspects. Also, it does not matter whether or not

the keyword order as specified in the query corresponds with the order of the keywords found in the document. Therefore, *keyword queries* are also said to be *unstructured queries*, and are often regarded as a set, or bag, of words. However, when it is possible to specify in what structural part or for what attribute a keyword must occur, the query is regarded as a *structured query* consisting of a collection of key-value pairs. Web forms with multiple input fields, like the one in Figure 1.1, are often used to enter structured queries consisting of key-value pairs. The key identifies the attribute or structural part, and the value basically denotes a restriction on that attribute or part. The notion of a structured query appears in other research areas as well, and should not be confused with, for instance, an SQL query (Codd, 1970) or an XQuery[8]. Throughout this thesis, unless stated otherwise, the term *structured query* refers to a set of key-value pairs. In the next section, we motivate a different means of entering a structured query. Rather than a multi-field web form, end-users can textually describe the structured query and enter the description in a single text field. We will use the term *free-text query* to distinguish such a textual description of a structured query from an ordinary keyword query.

1.2 A distributed deep web search approach

Ideally, we envision that end-users can search all web content using just one search engine: they will have a *single-point-of-entry* to both the deep web and the surface web. Generally however, deep content can only be accessed by submitting a structured query via a web form that has multiple input fields. It is impossible to aggregate all web forms into one large form in which one can enter all different kinds of structured queries. Besides, even if such a form could be created, it would have unworkably many fields and would become too complex and practically unusable. It is possible though to aggregate web forms per domain, using a method which is commonly referred to as *virtual integration* (see Chapter 2). For example, all travel-related web forms or sports-related forms could be aggregated. The aggregated forms would have reasonably many fields and would still be usable, but this approach would not lead to a single-point-of-entry.

Rather than creating forms with many fields, the approach that we propose in this thesis is to use a single text field and *translate* the end-user's free-text query into a structured query. The system that translates a free-text query will be referred to as a *free-text search system*, and the text field or interface in which a free-text query can be entered will be referred to as a *free-text interface* (to distinguish it from the traditional keyword-based web search interface). We further describe our approach and its related paradigm in Section 2.5.

Creating a single-point-of-entry to the deep web would be easier if deep web content could be accessed via free-text interfaces instead of via multi-field web forms. For instance, consider a search broker that mediates between end-users and all web sites or sources that offer deep web content (and that these sources

[8] `http://www.w3.org/TR/xquery/` (September 16^{th} 2013)

have a free-text interface). This broker serves as the single-point-of-entry, so end-users only need to interact with the broker. When a user submits a free-text query to the broker, the latter simply forwards the query to the most relevant sources. Whether or not a source is selected should depend on the query, e.g., if the query is about travel planning, it should not be forwarded to a shopping site. The sources would return their results to the broker which, in turn, would combine these results and return a single ranked list of results to the end-user.

1.3 Research questions

Our proposed solution for distributed deep web search assumes that a free-text query can be translated into a structured query either at the broker or at the deep web source. So, a necessary step is to ensure that a free-text query can be effectively translated into a structured query. We distinguish between two cases. In the first case, the free-text search system has full knowledge of the values that can be entered in a web form and how these values are typically used in a free-text query. In the second case, the free-text search system has only partial knowledge of the values that can be entered in a web form and how these values are typically used in a free-text query.

The effective translation of a free-text query into a structured query is an important aspect. Yet another important aspect is to ensure that a free-text query can be translated within a few milliseconds, and thus that the query translation process is efficient. However, a higher effectiveness may come at the price of a lower efficiency.

Our first set of research questions relates to the effectiveness and efficiency of translating a free-text query into a structured query:

RQ1: What is an effective approach to translate free-text queries into structured queries, when the free-text search system:

- **a**) fully knows what values can be entered in a single form and how these values are typically used in a free-text query?
- **b**) partially knows what values can be entered in a single form and how these values are typically used in a free-text query?
- **c**) fully knows what values can be entered in multiple forms and how these values are typically used in a free-text query?

RQ2: What is the trade-off between efficiency and effectiveness in translating a free-text query into a structured query?

The most common way to submit a structured query is via a web form that has multiple input fields. Even though in theory it may be possible to submit a structured query using a free-text interface, the question remains whether or not the free-text interface would be of practical use. That is, would end-users actually use this new way of searching to search for structured content. Therefore, our second set of research questions is user-centric and concerns the interaction between end-users and the prototype system in our experiments:

RQ3: Do end-users prefer to use a free-text interface rather than a complex web form for submitting structured queries?

RQ4: How do end-users phrase free-text queries when they intend to describe structured queries?

RQ5: What are the most frequent mistakes, if any, that should be taken into account in future free-text systems?

1.4 Thesis overview

Chapter 2 sketches the big picture surrounding deep web search. It introduces the classical *surfacing* and *virtual integration* paradigms, and explains why this two-sided classification scheme is too simplistic for a proper classification of existing deep web search systems. It then proposes a 7-point classification scheme and elaborates on a third paradigm, *virtual surfacing*, which is our vision of future web search systems.

Chapter 3 focuses on the scenario where the free-text search system knows all values that can be entered in a web form. It introduces a rule-based approach for translating free-text queries and describes a user study as a validation mechanism. The results of the user study serve to answer Research Question RQ1a, RQ3, and RQ4.

Chapter 4 focuses on how end-users interact with the free-text search system that was introduced in Chapter 3. Over 30,000 queries from almost 12,000 users were collected in an online experiment. This chapter summarizes the various ways in which end-users formulate their queries, and it evaluates the accuracy of the free-text search system. Further, 116 end-users participated in an online questionnaire, which compared the free-text interface with its multi-field counterpart. The results of this study will serve to answer Research Question RQ1a, RQ3, RQ4, and RQ5.

Chapter 5 focuses on the scenario where the free-text search system does not know all values that can be entered in a web form. It extends the rule-based approach of Chapter 3 by introducing three segmentation models, but re-ranks the results based on probabilistic Hidden Markov models. The results of this experiment will serve to answer Research Question RQ1b.

Chapter 6 focuses on the scenario where multiple web forms can be searched simultaneously. It introduces a stack decoding implementation of the probabilistic approach of Chapter 5, and uses heuristics to further increase the efficiency of the decoding process. The system is evaluated using data from an online experiment, and the results will serve to answer Research Question RQ1c and RQ2.

Chapter 7 concludes this thesis by revisiting the research questions of this chapter, and putting the conclusions from Chapters 3 to 6 in perspective. It discusses the limitations of our approach and gives suggestions for future work.

Chapter 2

Deep web search paradigms

"If fifty million people say a foolish thing, it is still a foolish thing -"
– *Anatole France*

In this chapter, we review prominent aspects for designing deep web search systems *and introduce the classic paradigms:* surfacing *and* virtual integration*. We observe that there is much variation between existing deep web search systems and that it would be better to describe these systems in terms of seven key aspects that we have identified. Finally, we motivate and introduce a third paradigm,* virtual surfacing*, which, in our vision, is a better way of searching the web.*

Parts of this chapter have been published in Tjin-Kam-Jet (2010); Tjin-Kam-Jet et al. (2011c).

2.1 Introduction

Web surfers are used to finding information on the web with search engines such as Bing, Google, and Yahoo. The speed at which these search engines return results can be largely attributed to their use of a centralized, inverted index. However, as discussed in Chapter 1, a centralized index also has several drawbacks. First, the index only contains a snapshot of the web page. If for instance, the actual page changes, the old content is still mirrored in the index instead of the new content. Thus, to keep the index up-to-date, a search engine must repeatedly re-crawl and re-index its pages. Second, much information on the web cannot be easily indexed because it is either difficult to crawl or it is inherently difficult to index: such content is referred to as *deep web content*.

Deep web content is usually accessed via web forms that have multiple fields. We refer to these web forms as *deep web interfaces* or as *complex web forms*. There are many possibilities for filling out a complex web form. However, some ways of filling out a form may not make sense and will either result in an error or in no response. Therefore, it is important to understand the interface and to know, for example, what kind of values make sense to enter in which fields in

order to gain access to the content behind the form. Next, we describe why it is a problem for crawlers to fill out web forms automatically, and describe approaches for accessing the content behind deep web interfaces.

2.1.1 Deep web interfaces: a problem for crawlers

When it comes to designing deep web interfaces, there is no standard for interface design[1]. Web developers are free to place or not to place labels that explain the kind of data that should be entered in each input field. They may use radio buttons, checkboxes, and selection menus; there is no pre-defined meaning for these control elements. Even worse, some web forms alter the typical behavior of certain control elements using small programs (e.g., JavaScript).

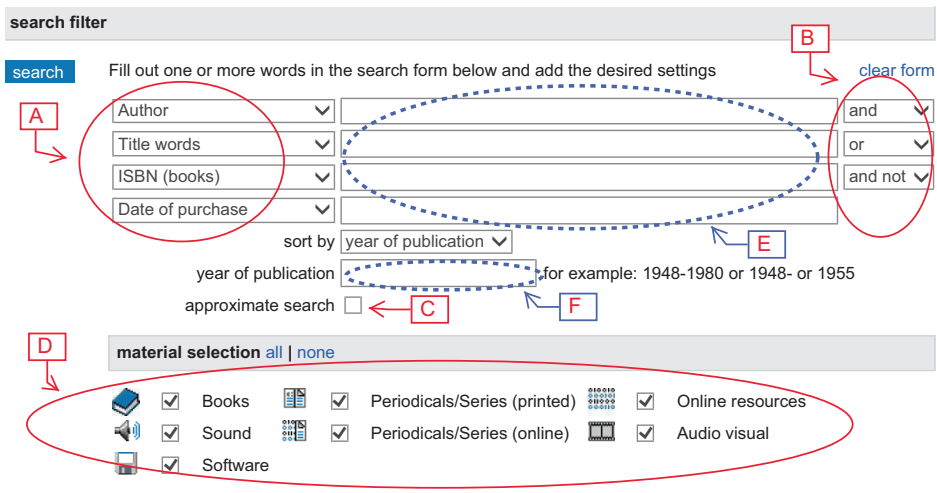


Figure 2.1: The advanced library catalogue search form of the University of Twente[3]. Certain fields affect how the values of other fields must be interpreted.

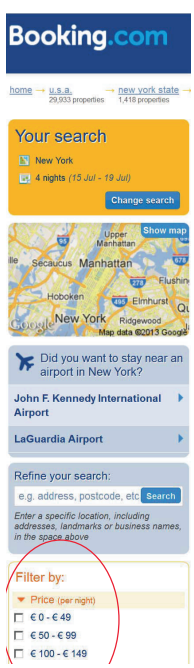
Consider the complex web form of the University of Twente library catalog depicted in Figure 2.1. The checkbox denoted by C affects whether the year that is entered in F must match exactly or approximately. Likewise, the options in group A and in group B affect how the values that can be entered in group E must be interpreted. Note that checkbox C has a label to its left ("approximate search"), while the fields in group E have no labels, instead, they have select menus (group A) that serve as labels in this case. This example shows that there is no pre-defined meaning of control elements and that the meaning of an element is to be determined in context of the other elements. Further, it is customary

[1]The W3C *has* standardized[2] what control elements can be used and how these should be rendered by the client browser. In other words, they have standardized the building blocks that developers can use.

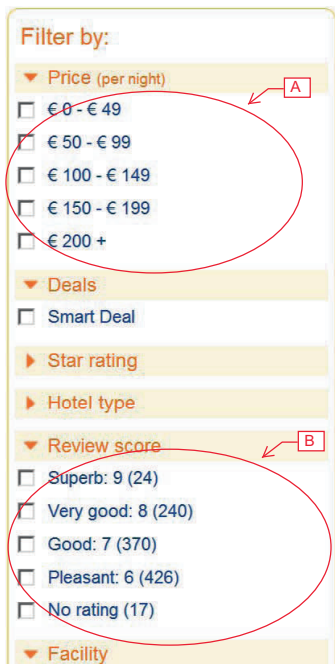
[2]`http://www.w3.org/TR/html401/interact/forms.html` (April 16^{th} 2013)

[3]`http://opc4.utsp.utwente.nl/DB=1/ADVANCED_SEARCHFILTER` (September 16^{th} 2013)

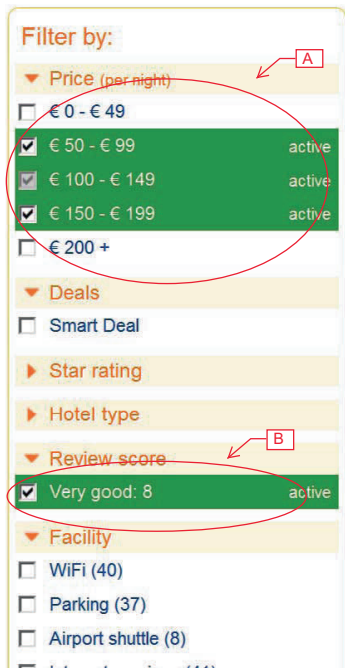
that multiple options can be chosen from a group of related checkboxes, while only one option can be chosen from a group of related radio buttons. In this example, the checkboxes in group D adhere to this custom as they can all be selected. However, Figure 2.2 (group B) shows an example where this typical behavior is altered so that only one option can be selected.



Search interface of Booking.com.



Part of the interface, before selecting some options.



Part of the interface, after selecting some options.

Figure 2.2: A complex, faceted search form in which the typical behavior of a checkbox is altered. Normally, from a group of checkboxes, you can select any number of options, as is the case in group A. However, in group B, as soon as one option is selected, the other options disappear.

We thus rely on common sense of the web developer to design, and the end-user to understand the web form and interact in a fruitful way. However, human end-users sometimes find it difficult to understand a deep web interface; an automated approach to understand and fill out a deep web interface, as needed by crawlers, is even more difficult to come up with.

As an aside, there is a technical reason why crawlers may refrain from interacting with web forms. As mentioned in Chapter 1, a form may use either a HTTP POST, or a HTTP GET request method. By convention, HTTP POST is used when a request affects the internal state of the web server, such as purchasing a product. A search request would typically not affect the internal state of a server, so a search form would typically use a HTTP GET request. However, there

can be (badly designed) web forms that use HTTP GET when in fact they should use HTTP POST, and vice versa. Therefore, crawlers generally refrain from interacting with arbitrary web forms to avoid any side effects such as deleting an item, or creating an account on behalf of someone else.

2.1.2 Understanding the deep web interface

The purpose of *interface understanding* is to enable a program to automatically submit queries and receive responses. Understanding a response is a related but different issue and discussed in the next section. In the simplest case concerning a single web form, interface understanding just means knowing the fields that can be filled out. A program could then fill out random values in these fields. Though not strictly necessary, it would be handy to also know what kind of values to enter in which fields, and how the values are related (e.g., knowing that a particular country has particular cities, or that a minimum value should be less than or equal to a maximum value). In more complex cases involving multiple web forms, interface understanding means knowing which fields share a similar semantic concept, thus enabling the program to (simultaneously) enter the same query in several related web forms.

Raghavan and Garcia-Molina (2001) adopt a task-specific, human-assisted approach to crawl deep web content. Their HiWE (Hidden Web Exposer) crawler issues structured queries that are relevant to a particular task, with some human assistance. For instance, by providing initial sets of "products" that are of interest, the crawler will know what to fill in if it encounters a form with a "product" field. They apply fuzzy matching to determine what values, if any, to fill in a field. They note: "The main challenge in form analysis is the accurate extraction of the labels and domains of form elements. Label extraction is a hard problem, since the nesting relationship between forms and labels in the HTML markup language is not fixed." By computing the layout for only that part of the page that contains the form, they aim to extract the labels that are visually adjacent to the fields. Álvarez et al. (2007) developed a crawler called DeepBot that is somewhat similar to HiWE. It also extracts labels that are visually adjacent to fields, and it uses domain-specific definitions. However, it fully supports JavaScript, and it is more flexible, e.g., it can detect if a field has more than one "label", which can result in better accuracy.

Zhang et al. (2004) hypothesize the existence of a hidden syntax that guides the creation of interfaces. This hypothesis effectively states that interfaces are utterances of a language with a non-prescribed grammar. Therefore, interface understanding is reduced to a parsing problem, for which they devised a 2P grammar and a best-effort parser. The 2P grammar specifies "patterns" and their "precedences" (hence the name 2P), as well as their relative positions from each other (e.g., left, right). A pattern is a production rule for a part of the interface. For example, in the patterns P1 and P2 below, pattern P1 states that a query interface, QI, consists of one or more "rows" of HQI. Pattern P2 states that each HQI consists of horizontally aligned patterns CP. Such patterns will eventually boil down to the actual fields and labels in the form. Generally,

the parser must capture all conventional patterns, which means that the parser should contain a large pattern database. However, having many patterns may lead to conflicts due to parsing ambiguity, in which case the precedence of the patterns should resolve the conflict.

$$\text{QI} \leftarrow \text{HQI} \mid \text{above}(\text{QI}, \text{HQI}) \tag{P1}$$
$$\text{HQI} \leftarrow \text{CP} \mid \text{left}(\text{HQI}, \text{CP}) \tag{P2}$$

In terms of identifying related interfaces, one approach is to hypothesize that "homogeneous" sources share a similar hidden generative model for their schemas (He et al., 2004a). Then, clusters should be chosen such that the statistical heterogeneity among the clusters is maximized. Another approach is to cluster the interfaces using fuzzy set theory (Zhao et al., 2008). Wu et al. (2004) rather aim for accurate, richer and more complex matching, that involves manual interaction to resolve uncertain mappings. Adequate interface clustering can in turn benefit query interface integration, for instance, He et al. (2004b, 2005) explore methods for integrating interfaces of a similar domain. A survey by Khare et al. (2010) is as a good starting point for further reading on query interface understanding. A recently published book titled *Deep Web Query Interface Understanding and Integration* by Dragut et al. (2012) gives a comprehensive overview of the approaches that have been developed over the last decade. The book is written from a virtual integration perspective. We introduce virtual integration in Section 2.2.2.

2.1.3 Web scraping: understanding the results page

It is natural to expect the results of a query to contain one or more (links to) relevant answers. However, the page that contains the relevant answer will usually also contain a lot of irrelevant information such as advertisements, navigation links, and information about other items. Therefore, the actual meaningful pieces of information must be extracted from the web page. This is referred to as *web scraping*, and programs used for scraping are often called web *wrappers* or web *scrapers*. Initially, wrappers were built by hand, which was a tedious job for developers. Wrappers were often site-specific, so if a new site was added, a new wrapper had to be built. Therefore, research shifted towards "generic" information extraction techniques, and automatic wrapper generation techniques, called *wrapper induction*. Wrappers based on generic techniques were less accurate than domain-specific wrappers, but they were also less susceptible to changes in the result page. There is much research on wrapper induction (Kushmerick et al., 1997; Kushmerick, 2000; Wang and Lochovsky, 2003; Zhao et al., 2005; Zheng et al., 2007; Chuang et al., 2007; Senellart et al., 2008; Liu et al., 2010; Weninger et al., 2010; Dalvi et al., 2011). For further reading, we recommend early surveys on wrapper induction by Laender et al. (2002); Flesca et al. (2004), and more recent articles by Dalvi et al. (2011); Trieschnigg et al. (2012).

It is not our intention to further explain how wrappers work, so for further reading, we refer to the material listed above on wrapper induction and scraping

techniques. We do wish to emphasize what can be concluded from this research: it is possible to automate the extraction of *structured* content (i.e., to extract the values of a particular aspect or attribute from a page), although the extraction process may sometimes yield wrong results.

2.1.4 Machine readable results

One reason why scraping is needed is because web sites are designed for people, e.g., the content is formatted in HTML pages intended for human reading and browsing (Kushmerick et al., 1997). To make "scraping" easier, websites can enrich the HTML structure with a standard meta-data markup scheme[4] (this falls between scraping and being machine readable). However, the content could also be made available in a standard machine readable format (e.g., XML, JSON, RDF, Atom, RSS), or accessed directly through an API (application programming interface). If a web site provides a feed of frequently updated content, it may choose to publish this in a *syndication format*, like Atom and RSS. Clients can then subscribe to this feed, and can check for updated content. Additionally, if a site also provides a search service, it can specify how to use its search interface by describing it in a document according to the OpenSearch standard[5]. If deep web content would be made available in a *standardized* machine readable format, it would mean a step forward for deep web search systems. However, if all web companies would provide *customized* APIs, then separate code would have to written for each API. For now however, web companies have no incentive to provide separate interfaces to their data besides their public web interfaces. When companies do provide a separate API, studies have shown that the results retrieved via the API do not always correspond to those retrieved from the web interface (Alba et al., 2008; McCown and Nelson, 2007).

2.2 Surfacing versus virtual integration

In this section, we describe how the approaches of the previous section, to automatically fill out forms and extract content, are used to build deep web search systems. In literature, two paradigms for deep web search systems are distinguished: *surfacing*, and *virtual integration*. We now introduce these paradigms and point out their main strengths and weaknesses.

2.2.1 Surfacing

In surfacing, web forms are automatically submitted with "guessed" field values and the resulting pages are indexed like a surface web page[6] (Raghavan and Garcia-Molina, 2001; Álvarez et al., 2007; Barbosa and Freire, 2007; Wu et al., 2006; Madhavan et al., 2008). This approach has several disadvantages: since

[4] `http://www.schema.org/` (May 11^{th} 2013)
[5] `http://www.opensearch.org` (May 11^{th} 2013)
[6] A web page which can be crawled by following hyperlinks.

there are many deep web sources, the crawler cannot afford to linger on each source, but it is challenging to efficiently guess how to fill out the web forms (Cafarella et al., 2008b); the goal is to index as much content of the source as possible without causing too much traffic, but maximizing content coverage while minimizing query traffic is challenging (Wu et al., 2006; Callan and Connell, 2001; Madhavan et al., 2008); the surfaced content is often the result of a structured query and is then stored in a keyword index, which may result in a loss of semantics. Therefore, it may not be possible to retrieve the content from the keyword index using a (structured) query (Madhavan et al., 2009); lastly, surfacing is effective only for certain kinds of deep content, e.g., not for highly-dynamic content, since this would require the system to frequently re-crawl the contents. Yet the biggest advantage is that this approach can be coalesced in the existing infrastructure of a search engine. Therefore, it can scale to web proportions and serve as a single-point-of-entry to both traditional search results as well as deep web search results.

We illustrate the surfacing approach and its offline and online processes in Figure 2.3. The offline process (depicted in the gray background) probes the deep sites by repeatedly submitting web forms with guessed values for the input fields. The deep sites respond with web pages that possibly contain search results. These pages are then indexed by the search engine. The online process accepts and matches the user query against the search engine's local index and returns results from this index.

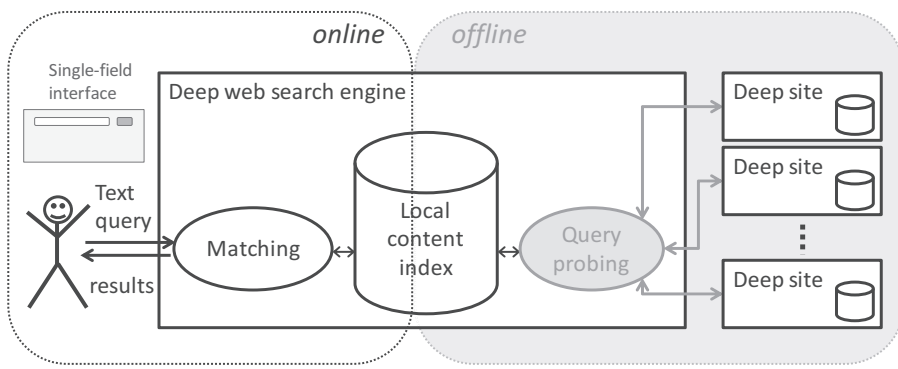


Figure 2.3: Schematic overview of surfacing.

2.2.2 Virtual integration

In virtual integration, related deep sources are integrated in a larger, virtual system by merging their interfaces (Dragut et al., 2009b; Madhavan et al., 2009; Calì and Martinenghi, 2010; He et al., 2005; Chang et al., 2004b; Halevy et al., 2006b). It is important to understand how the interfaces relate to each other, and which fields share similar semantics. A unified multi-field interface (MFI)

must be created such that each field of the MFI links to the related field(s) in each deep source's web form. Effectively, when an end-user fills out the unified MFI, the user is filling out multiple web forms at the same time. This approach has several disadvantages: creating a unified MFI is challenging because the deep sources may have different query capabilities, e.g., one source can search for a particular attribute, whereas the other cannot; the field mapping is not always straightforward. In one form, an information need can be specified by entering one value in one field, whereas in another form, multiple fields may be required (e.g., one form may have a field for "person name", and another may have fields for "first name" and "last name"); and finally, despite the efforts in automatic interface extraction and schema mapping, this approach does not scale to web proportions. However, if there are not too many sources and they can be well managed, the biggest advantages are that it supports structured queries, that it fully covers the contents of the underlying sources, and that it can effectively be used for any kind of deep web content.

We illustrate the virtual integration approach and its offline and online processes in Figure 2.4. The offline process (depicted in the gray background) detects the schemas of (the query interfaces of) the deep sites and stores them in a database. It then uses the schemas to build a unified MFI (i.e., it knows exactly how each field of the unified interface maps to some field of a deep site). The online process forwards the user query to the deep sites, downloads and merges the content of the disparate sources, and presents the results to the user.

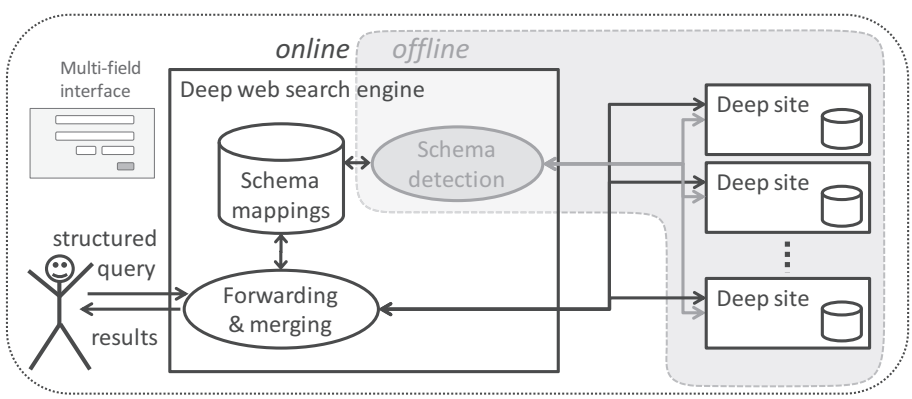


Figure 2.4: Schematic overview of virtual integration.

2.2.3 Summary

To recapitulate, surfacing means submitting forms and putting the contents in a centralized keyword index, end-users can then use keyword queries to search; virtual integration means building a unified interface and leaving the contents at the deep web sources, end-users can then use structured queries to search. For surfacing deep web content, we can get away with "not really understanding"

the deep web interface, and simply submitting guessed values. For virtual integration however, understanding the interface is important because fields that are semantically related must be linked. Arguably, the virtual system must also interpret the results of deep web sources. For example, if a query requires that all results be sorted by price in ascending order, then the virtual system must understand how the results can be compared in order to sort them.

2.3 Seven aspects of deep web search systems

The traditional distinction between, and the definition of, surfacing and virtual integration seems to take only two aspects into account: index location and query handling. But what if a system has some aspects that are typical of both approaches? It would be better to describe deep web search solutions by their key aspects as it is more specific. It is also more insightful to explain the advantages and disadvantages of each aspect individually. Based on our observation of the differences amongst deep web search systems, e.g., their query handling and design choices, we propose seven aspects in which these systems can be categorized. These aspects do not cover implementation details, but up to some extent, this can be inferred. For example, to support keyword query handling, it is likely that the data will be stored in an inverted index. To support structured query handling, it is likely that the data will be stored as structured records in a relational database. We now describe the seven aspects:

1. **Index location.** We distinguish between a *local index* and a *remote index*. A search engine can build a local index of deep content and serve results from this index. Alternatively, it can forward the query to the remote deep site, and thereby "use the remote index" to show results. A local index has the advantage that it usually has faster response times compared to a remote index, as it does not have to wait for other systems to respond. However, a disadvantage is that it can get out-of-date, whereas the remote index is by definition[7] up-to-date. Furthermore, the choice of index location has impact on the effort needed to keep the index up-to-date, and on what kind of content can be effectively retrieved.

2. **Content dynamics.** In Chapter 1, we introduced the terms *static*, *dynamic*, and *highly dynamic* to refer to content that rarely or never changes, that changes often, or that changes very frequently, respectively.

   Mostly static and dynamic content can be effectively served from a local index. A local index could in theory contain up-to-date highly-dynamic content, if the content was crawled and indexed just before the query was issued. However, it is not a reliable way to show highly-dynamic content: forwarding the query to the deep source and displaying those results is the safest way to show highly-dynamic content. Also, if some content has the

[7]The content in a remote index is used to generate deep web pages. Therefore, the contents of the generated deep web pages and the remote index will always be in sync.

seemingly unlimited property (see Chapter 1), only part of this content can be indexed and thus retrieved. In other words, if the content is highly dynamic, or if it has the seemingly unlimited property, then a local index clearly has a disadvantage over a remote index.

3. **Query handling.** We distinguish between (also) supporting *structured queries* or only supporting *keyword queries*. An advantage of a structured query is that it enables end-users to specify restrictions on one or more attributes of an item so that they can expect focused results. For example, the search results for a laptop costing less than 450 dollars, should only contain laptops (so no mobile phones, game consoles, or desktop computers) which are less than 450 dollars (so no popular laptop for 699 dollars). Though the example might seem trivial, a keyword based retrieval system might actually produce erroneous results, such as mobile phones or desktops. A disadvantage of structured queries is that it may be more complex to maintain the data.

4. **Query interface.** We distinguish between a *single-field interface* (SFI), or a *multi-field interface* (MFI). Both interfaces could in theory be used for entering structured queries and/or keyword queries, but in practice, the SFI is often used for keyword queries, and the MFI is often used for structured queries. We now consider a search system that supports structured queries. If the system has an MFI, then each field of the MFI will link to one or more fields of the underlying interfaces of the deep sources. If the system has an SFI, then it must first translate a free-text query to a structured query for each of the deep sources. Compared to the SFI, the MFI has the advantage that it significantly reduces the processing steps required at query-time. However, automatically creating and maintaining the MFI in the first place is challenging. Furthermore, MFIs have the disadvantage that separate interfaces must be maintained per domain since each domain has its own set of "generic fields". SFIs, on the other hand, do not have this disadvantage.

5. **Content acquisition.** We distinguish between *crawling links*, *crawling forms*, *scraping*, and *obtaining data via machine readable results*. Crawling links refers to the conventional way of following hyperlinks to download content, which is relatively easy. Crawling forms refers to submitting web forms to download deep content, which is relatively hard, as we explained in the beginning of this chapter. The advantage of crawling is that, since the content is acquired through the standard web interface (through which end-users access the content), the acquired content is consistent with the content that end-users see. Scraping refers to extracting structured information from crawled web pages. The ability to extract structured information is an advantage. However, scraping may sometimes yield wrong results. Finally, obtaining data via machine readable results refers to data that is acquired through APIs, or in standardized formats like XML. The

Table 2.1: Most important differences between surfacing and virtual integration.

	Surfacing	Virtual integration
1. Index location	local	remote
2. Content dynamics	static, dynamic, partial coverage	static, dynamic, highly-dynamic, full coverage
3. Query handling	keyword query	structured query
4. Query interface	single-field interface	multi-field interface
5. Content acquisition	*crawling forms*	*scraping or API*
6. Domains & sources	sources of any domain	sources of same domain
7. Results interface	*local-static*	*local-interactive*

advantage is that this allows complex, structured information to be described to a computer. In practice however, companies sometimes provide machine readable content that is inconsistent with the content that end-users see (see Section 2.1.4).

6. **Domains and sources.** We distinguish between *single* or *multiple* topical domains (e.g., travel planning, hotel booking, or car rental), and *single* or *multiple* sources. A deep web search system serves results from one or more sources which can be from the same domain, or they could be from multiple domains. The ability to query over multiple domains has the advantage that it would provide a single-point-of-entry to all sorts of systems. Querying over a single domain has the advantage that the user interface can be tailored to the domain, potentially providing better guidance for end-users to formulate their queries.

7. **Results interface.** We distinguish between a *remote results interface*, a *local-static interface*, or a *local-interactive interface*. If, after a query has been submitted, a search system immediately redirects the user to the deep site containing the most likely result, then the system uses a remote results interface. If the system displays a list of results so that the user can select which results to view, then the system uses a local-static interface. If the system provides additional faceted search capabilities, like sorting and filtering on specific attributes (e.g., size, color, or price), then the system uses a local-interactive interface. A remote results interface has the advantage that it removes some cognitive load from end-users, as they do not have to inspect the result list before they can decide what result to click. A local-static interface has the advantage that it gives end-users the freedom to make their own selection of possibly relevant results, but at a higher cognitive load. A local-interactive interface does not only offer the freedom to select your own results, but also offers added functionality to further refine your query and slice-and-dice the results.

In terms of these aspects, we can quickly summarize the differences between surfacing and virtual integration, as shown in Table 2.1. The aspects *content acquisition* (no. 5) and *results interface* (no. 7) are not explicitly reported in

literature, that is why we used italics to indicate what we believe to be the most likely choices for the aspects in each paradigm. As a final remark, note that one can search content from multiple domains and sources in the surfacing paradigm, whereas one can only search content from a single domain in the virtual integration paradigm. As a consequence, surfacing is better suited for being a single-point-of-entry to the entire web.

2.4 A classification of deep web search systems

Deep web search systems come in more variations than just surfacing or virtual integration. We illustrate this in Table 2.2, where we tabulate several systems according to the seven aspects of the previous section. We do not claim that this list of systems is exhaustive; however, it includes enough different systems to give an overview of current solutions to deep web search. We briefly describe each system in Table 2.2.

Table 2.2: A classification of deep web search systems. Items between parenthesis are hypothetical and are not explicitly reported in the original paper(s).

		Aspect						
	System	Index location	Retrievable content	Search functionality	Query interface	Content acquisition	Domains and sources	Results interface
Surfacing	HiWE	L	(2)	-	-	2,4	B,2	-
	DeepBot	L	(2)	-	-	2	B,2	-
	WebTables	L	1,2	(1)	(1)	4	B,2	(2)
	Google surfacing	L	1,2	2	1	2	B,2	2
	"Item search"	L	1,2	(1,2)	1	1,4	B,2	3
Virtual integration	Flight planners	R	2,3	1	2	4	A,2	3
	MetaQuerier	R	2,3	1	2	4	A,1	(2)
	WISE-integrator	R	2,3	1	2	4	A,2	(2)
	VisQI	R	2,3	1	2	4	A,2	(2)
	FTI-3	R	2,3	1	1	4	A,1	1,2
	FTI-6	R	2,3	1	1	4	B,2	1,2

The first two systems, HiWE (Raghavan and Garcia-Molina, 2001) and DeepBot (Álvarez et al., 2007), are actually deep web crawlers and are not complete search systems. However, the kind of results we could expect with a hypothetical search system on top of the crawled data would be dynamic in nature; this hypothetical finding is indicated with parenthesis in the table.

The WebTables project (Cafarella et al., 2008a; Cafarella, 2009), extracts structured content from tables (i.e., indicated with the `<table>` HTML tag) residing in web pages found in the Google index. It is not entirely clear what kind of structured queries are supported, and what kind of query and results interfaces

Table 2.3: Legend explaining the aspects and values in Table 2.2.

Aspect		Short description
Index location		
R	Remote	Search engine shows results from remote data source(s)
L	Local	Search engine shows results from local index
Content dynamics		
1	Static	Content is not likely to change over time
2	Dynamic	Content is very likely to (repeatedly) change over time
3	Structured	Content is the result of a (proprietary) web application
Query handling		
1	Structured	Search engine supports structured (key-value) queries
2	Non-structured	Search engine supports basic keyword queries
Query interface		
1	Single field	Search interface consists of single text field
2	Multiple fields	Search interface has multiple input fields
Content acquisition		
1	Crawling links	Search engine downloads web pages by following hyperlinks
2	Crawling forms	Search engine surfaces web pages by submitting web forms
3	Scraping	Search engine extracts structured records from web pages
4	Machine readable results	Web pages either contain meta-data markup which aids scraping, or web site and search engine transfer structured data via custom APIs
Domains and sources		
1	Single	Search engine can only return answers from one source
2	Multiple	Search engine can return answers from multiple source
A	Single	All sources are from the same domain
B	Multiple	Sources are or can be from different domains
Results interface		
1	Remote	Results are accessed and displayed from the original source
2	Local-static	Result summaries are simply displayed at the search engine
3	Local-interactive	Results can be filtered or sorted on different attributes

are used. According to the authors, queries may contain `spatial` operators (e.g, `samecol` and `samerow`, which only return results if the search terms appear in cells in the same column or row of the table) and query-appropriate visualization methods are used to render the results.

Google also surfaces deep content (Madhavan et al., 2008; Cafarella et al., 2008b), but, to our knowledge, the system does not support structured (key-value) queries; instead, the standard keyword index is used.

"Item search"[8] refers to structured (key-value) search that is enabled because: *i*) the search engine supports structured queries; and *ii*) the (surface or

[8]Examples of search engines that support structured queries over items that are specified in a machine readable format:
`http://www.bing.com/shopping/search?q=example` (August 19^{th} 2013),
`http://www.google.com/prdhp?hl=en&tab=pf` (August 19^{th} 2013),
`http://www.bing.com/recipe/search?q=chocolate` (August 19^{th} 2013),
`http://www.pricerunner.co.uk` (August 19^{th} 2013),
`http://shopping.yahoo.com` (August 19^{th} 2013).

deep web) data is published in an open standard, or is delivered via an API. Examples of item search systems include Bing Product Search, Yahoo Shopping, and PriceRunner. These systems use a central index since some data may be crawled from the surface web. Also, they support structured search by means of *facets*. For example, in PriceRunner, the search results for the keywords "digital camera" can be narrowed down further by facets like manufacturer, effective pixels, and optical zoom. Structured queries can *only* be specified by making use of the facets, e.g., typing "digital camera, manufacturer: Sony" in the search field does not yield the same results as typing "digital camera" and choosing "Sony" for the manufacturer facet. Note that the structured queries are only possible *after* issuing a keyword query, because the facets are only shown in the results interface. The initial query interface only shows a keyword input field.

A flight planner[9] brokers over many airline sites and shows highly-dynamic flight results. A multi-field search interface allows users to enter structured queries. The available flights are shown in a local interactive interface allowing the user to refine the results and easily compare results from different sources.

MetaQuerier (He et al., 2005) translates, on-the-fly, a query expressed in one interface to a set of queries in a target interface. Translation can take place without specifically prepared translation knowledge; so it should be applicable over various domains as long as both source and target interfaces are from the same domain. We do not know what kind of results interface is used by MetaQuerier.

WISE-integrator (He et al., 2004b) automatically creates a unified interface for a group of web forms of the same domain. Based on the visual layout of a web form, it extracts attributes which are used to match and integrate multiple interfaces into a unified interface. The unified interface consists of multiple fields, so users can pose structured queries. We do not know what kind of results interface is used by WISE-integrator.

VisQI (Dragut et al., 2009a,b; Kabisch et al., 2010) also automatically creates unified interfaces for groups of web forms of the same domain. It adopts a hierarchical representation of query interfaces, and it outperforms previous approaches (on extracting query interfaces) with about 6.5%. We do not know what kind of results interface is used by VisQI.

FTI-3 and FTI-6 are free-text search systems that are described in more detail in Chapters 3 and 6, respectively. In terms of their functionality, FTI-3 only searches a single source in a single domain, whereas FTI-6 searches multiple sources in multiple domains.

2.5 Virtual surfacing: the third paradigm

As introduced in Chapter 1, we envision a system where end-users can search both the deep web and the surface web using just one single-field interface. It would not

[9] Examples of search brokers that support structured queries and search multiple airlines:
`http://www.kayak.com/flights` (August 19^{th} 2013),
`http://www.travelocity.co.uk/site/travel/flights` (August 19^{th} 2013),
`http://www.cheapoair.com` (August 19^{th} 2013).

only serve as a single-point-of-entry to the complete web, it would also offer both unstructured (keyword) and structured (key-value) search capabilities. However, neither within the bounds of the surfacing paradigm, nor within the bounds of the virtual integration paradigm, can we create such a system. Therefore, we need a new paradigm. As can be seen in Table 2.2, FTI-6 contains aspects that are normally found in surfacing approaches (e.g., a single-field search interface to all data, searching of many sources and domains), but in essence it is a virtual integration approach (e.g., remote indices, dynamic and highly-dynamic results); as such, it tries to combine the best of both worlds. In a way, it represents a new paradigm as it offers: multi-domain keyword search capabilities and multi-domain structured search capabilities through a uniform, single-field search interface. We refer to this as *virtual surfacing*.

2.5.1 Scientific motivation

The scientific motivation for virtual surfacing comes both from a *dataspace* perspective (Franklin et al., 2005; Halevy et al., 2006a), as well as a *distributed information retrieval* perspective (Callan, 2000; Si and Callan, 2005).

After observing that deep web sites are autonomous, independent, and offer site-specific (structured) search capabilities on a specific data set, we can regard deep web sites as participants in a dataspace. In a dataspace, there is no central coordination, all participants simply co-exist, and one requirement is that each participant provides support for at least basic text querying. The dataspace roadmap identifies six research challenges, one of which concerns the ability to pose basic queries to any participant in such a heterogeneous environment. This is where our free-text query translation plays a key role and can be part of the solution, as it enables each deep site to provide support for text querying.

The motivation from a distributed information retrieval perspective comes from the fact that the (structured) search functionality of certain deep sites can be much more advanced than that of other sites in the same domain. Rather than providing only the general search functionality that is available across all sites of a given domain, it would be beneficial to also provide the more advanced, site-specific, search functionality. This requires a query to be issued at the deep site itself, which can be accomplished by having a search broker that forwards the end-user's query to relevant participants. Having a broker that mediates between end-users and (deep) web search engines is essentially a federated search architecture. One of the research challenges in distributed or federated search concerns the broker's ability to select the most relevant search engine, or resource, for the given query. Existing resource selection methods either treat each resource as a very large document and apply bag-of-words retrieval methods to select the best resource, or simply select the most popular resources. In this respect, the contribution of our virtual surfacing approach is that, rather than modeling the contents of a resource, we model the queries that can be accepted by the resource. In turn, this information could lead to better resource selection methods.

2.5.2 Two federated search architectures

We distinguish two federated architectures in which virtual surfacing can be realized. In the first architecture, shown in Figure 2.5, a free-text search system is placed as an abstraction layer on top of a deep web site so that free-text queries can be translated to structured queries. This in turn simplifies federated search in two ways. First, there is no need for a unified schema per domain, because all sources of all domains have a single field, free-text interface. Second, no query adaptation is needed at the search broker: the free-text query can be forwarded as is. In this architecture, the broker provides (structured) deep web search through the free-text search capabilities of deep sites, and it provides surface web search through the keyword search capabilities of general surface web search engines. Offline, the broker creates a description of each search engine, this is indicated in gray in Figure 2.5. Online, at query time, the broker uses these descriptions to select search engines that are most likely to return relevant results for the given query, and forwards the query to those search engines. This incurs additional network latency as, once the query has been forwarded, the broker must wait for the responses of the remote engines before it can merge their results and show the results to the user. Despite the additional network latency, the benefit of this architecture is that the results are always up-to-date, that the search broker does not need to maintain the free-text configurations of the deep web sites, and that it can simply forward the free-text query to remote search engines. In Chapter 3, we will introduce a rule-based free-text search system which can be configured to translate free-text queries into structured queries for a particular deep site.

In Federated Search Architecture 2, shown in Figure 2.6, the search broker keeps a free-text-configuration for each deep web site in a *deep web entry-point index*. This index does not contain actual deep web content. Instead, it is used to

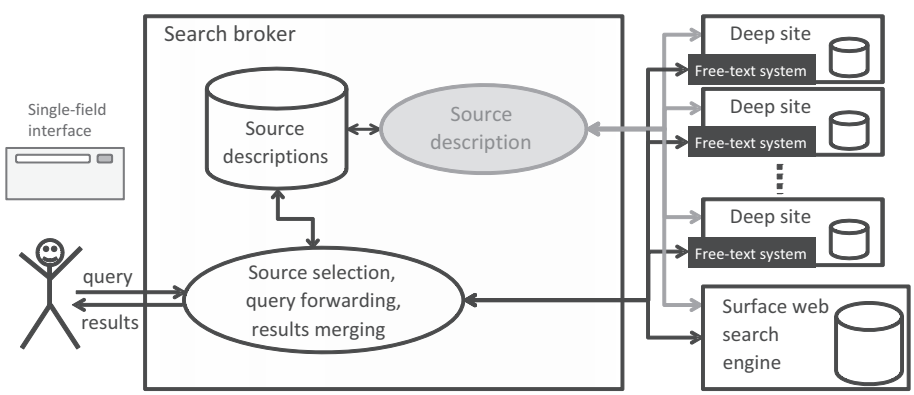


Figure 2.5: Federated Search Architecture 1. Offline, depicted in gray, the search broker creates a description of each search engine. Online, the broker forwards the query to one or more search engines, merges their results, and presents the results to the end-user.

generate 'entry-points' that provide access to deep data. In this case, an entry-point is a filled-out web form: by submitting the web form, end-users can obtain the actual deep web results. In this architecture, the broker provides (structured) deep web search by translating free-text queries into structured queries. The broker knows how to interpret and translate a free-text query for each deep site because it keeps a free-text-configuration for each deep site. Consequently, the broker can generate deep web search results without having to consult the remote search engines, and therefore, without the incurred network latency of consulting the remote engines. The broker has two alternatives for providing surface web search results (these alternatives are not depicted in the figure). The first alternative is to "outsource" surface web search and thus to forward the query to existing web search engines. Like in the first architecture, this introduces additional network latency. The second alternative is to let the search broker itself crawl and index the surface web, and maintain a local keyword index. A query can then be matched against both local indexes (i.e., the deep web entry-point index and the keyword index). This alternative has two advantages. First, there is no additional network latency which is otherwise introduced by forwarding the query and waiting for the response. Second, since the broker has access to the result scores, it can lead to better final rankings of the combined deep web and surface web results. Overall, the benefits of this second architecture are that it minimizes network latency, and that it can potentially deliver better combined rankings since it has more information about the type of queries that can be answered by each deep site. However, unlike the first architecture, the broker cannot simply forward a free-text query to remote search engines. Instead, the query translation task falls to the broker. In Chapter 6, we will introduce a free-

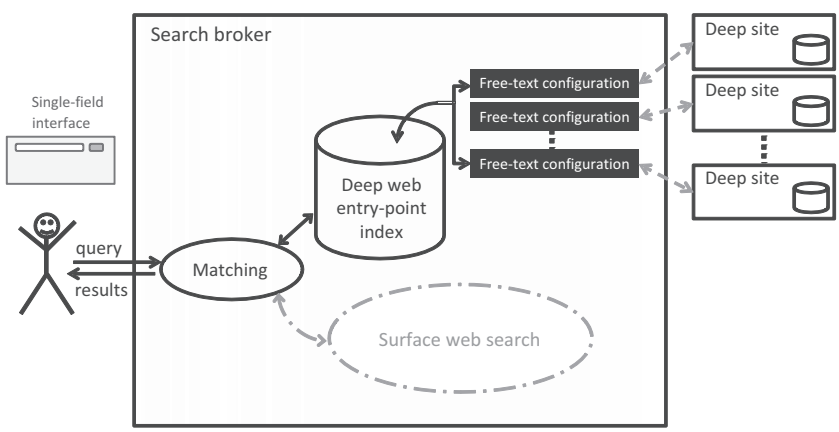


Figure 2.6: Federated Search Architecture 2. Offline, the broker obtains free-text-configurations from each deep site. Online, the broker translates the free-text query to produce deep web search results; and, it obtains surface web search results either from other search engines, or from its own keyword index.

text search system which can be configured to translate free-text queries into structured queries for multiple deep sites.

2.6 Summary

In this chapter, we presented an overview of the problems, solutions, and paradigms for deep web search. We introduced the two traditional search paradigms, *surfacing* and *virtual integration*, and observed that existing deep web search systems sometimes contained aspects of both paradigms. We therefore proposed to describe deep web search systems by seven key aspects: *i*) index location; *ii*) Content dynamics; *iii*) query handling; *iv*) query interface; *v*) content acquisition; *vi*) domains and sources; and, *vii*) results interface. It is not only more specific to describe search systems by these aspects, it is also more insightful for explaining the advantages and disadvantages of each individual aspect. Furthermore, each paradigm has different problems and benefits. In the *surfacing* paradigm, the main problems are that it is hard for crawlers to automatically fill out and submit complex web forms to access deep web content, and that it is not possible to perform structured search. The main benefit is that it is possible to search in multiple domains using a simple search interface. In the *virtual integration* paradigm, the main problems are that it is hard to unify the search interfaces of deep sites, and that a separate virtual search system must be created for each domain. The main benefits are that it is possible to perform structured search and that the results are up-to-date. Finally, we motivated and introduced a third paradigm, *virtual surfacing*. Virtual surfacing offers multi-domain keyword search capabilities and multi-domain structured search capabilities through a uniform, single-field search interface. Its main challenge is the translation of free-text query to structured queries. Virtual surfacing contains aspects that are normally found in surfacing (e.g., a single text search interface to all data, searching of many sources and domains), as well as aspects that are normally found in virtual integration (e.g., remote indices, structured queries, dynamic and highly-dynamic results); as such, it tries to combine the best of both worlds.

Chapter 3

A rule-based approach to translate free-text queries into structured queries

"They are ill discoverers that think there is no land,
when they can see nothing but sea -"
– *Francis Bacon*

In this chapter, we investigate the problem of using free-text queries as an alternative means for searching 'behind' web forms. First, we describe a rule-based system for translating the free-text queries into filled out web forms, and a specification language for specifying free-text interfaces. Second, we validate the approach with a user study in a laboratory experiment.

Parts of this chapter have been published in Tjin-Kam-Jet et al. (2011b,a).

3.1 Introduction

Many web pages can only be accessed by submitting complex web forms. Journey planners, real estate websites, online auction and shopping websites commonly require the end-user to fill out a form consisting of a number of fields in a graphical interface. The end-user must first interpret the form and then translate his or her information need to the appropriate fields. Filling out these forms can be slow because the user typically switches between using the mouse to select an input field and using the keyboard to enter a value. A *free-text interface* (FTI) alleviates these problems by allowing the end-user to enter an information need in a single textual statement. Rather than navigating between and entering information in the components of the web form, the end-user can focus on formulating his or her information need in an intuitive way.

An FTI should offer simple textual access to content behind a complex web form that consists of multiple input fields and options. It should accept an end-user's free-text query as input, which is a description of a structured information need, and return a ranked list of plausible interpretations (which are ways of filling out the complex web form) as output. A study about the usability of natural language search interfaces showed that end-users often do not know what capabilities a system has and that they should be guided during the query formulation process (Kaufmann and Bernstein, 2010). An FTI should therefore offer dynamic query suggestions as a means to guide users in formulating their queries. Lastly, it should be easy for developers to configure or specify the query capabilities of an FTI. Examples of a complex web form for a travel planning website and an FTI to the same form, are given in Figs. 3.1 and 3.2, respectively.

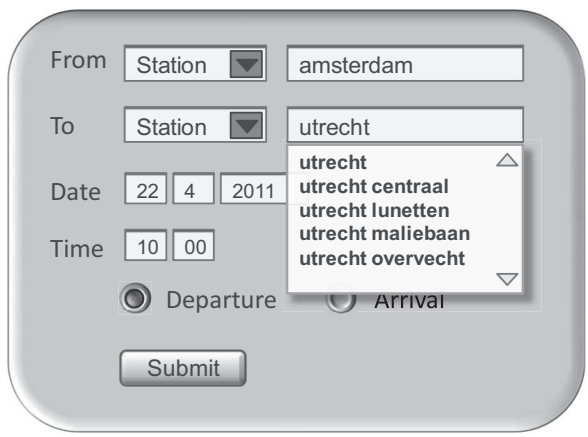


Figure 3.1: A complex web form that offers interactive query suggestions, based on the Dutch Railways site.

Figure 3.2: A *free-text interface* that offers interactive query suggestions, tailored to the complex web form.

Throughout this thesis, we postulate that an FTI would be more user-friendly

and that end-users would rather use an FTI instead of a complex web form. In this chapter, we demonstrate that a rule-based approach can be used to extract the underlying structured information need from a free-text query and we provide answers to the following questions:

1. Can a rule-based approach effectively be used to translate free-text queries into structured queries for a single web form?
2. Do end-users prefer to use a single-field, free-text interface over a multi-field, complex web form to enter structured queries?
3. How much variation exists in the query formulations of end-users?
4. Are end-users consistent in their query formulations?
5. What are positive and negative aspects of the free-text interface?
6. Is searching by means of a free-text interface faster than searching by means of a complex web form?

Next, we describe a general, rule-based framework for translating free-text queries into structured queries and describe how it can be configured for a particular form. We then describe a user study experiment that provides answers to the questions listed above, and conclude this chapter.

3.2 The FTI framework

Let us start with the assumption that an end-user is familiar with a particular complex web form and that, instead of filling out this form, he or she can enter the query as a textual description. Also assume that this description contains the exact values that would otherwise have been used to fill out the web form.

From the system's perspective, the free-text query of the end-user is just a sequence of characters. In order to distill the intended fields and values, it must perform three tasks: *i*) it must decide how to split the input into smaller segments; *ii*) for each segment, it must decide whether the segment denotes a value that should be entered in one of the form's input fields; and, *iii*) if that is the case, the system should also decide in which field to put the value. This process of finding the intended fields and their corresponding values is referred to as *query interpretation*. After query interpretation, the system will either generate *query suggestions* or generate *result snippets*. The following subsections explain these processes in more detail.

3.2.1 Query interpretation

To simplify the interpretation process, we require that all values that can possibly be entered in the complex form must be specified to the system (e.g., put in a dictionary or described by a regular expression). Let us define the following: a *segment* is a substring of the user's input and consists just of a sequence of

characters. A *token* is a textual unit that can be assigned to a field, consists of one or more words, and belongs to one type. A *type* denotes a set of tokens. A *pattern* is a regular expression that is used to identify known (combinations of) tokens. A *label* is assigned to a segment to denote that the segment contains some (part of a) pattern. Finally, an *interpretation* is a set of labeled segments. The query interpretation process consists of five steps:

1. *Identifying segments:* the free-text query is scanned for known patterns from left to right. At each word (words are delineated by a white space), all matching patterns starting from that word are stored. Pattern matching is greedy, meaning that a pattern will match as much of the query as possible. This process yields a set of possibly *overlapping* segments;

2. *Generating non-overlapping sets of segments:* because the found segments may overlap, the set Γ of all combinations of segments must be generated such that: *i*) each combination contains no overlapping segments. And, *ii*) each combination must have maximal coverage of the input, i.e., adding another segment would cause one or more segments to overlap;

3. *Generating interpretations:* each segment can have multiple labels since it can be matched by multiple patterns, e.g., a segment containing the characters '2000' could be labeled as a year, an amount of money, or a car model. Therefore, for each combination of non-overlapping segments $\gamma \in \Gamma$, all combinations of labels must be generated such that each segment corresponds to one label. This yields the set of possible interpretations;

4. *Filtering interpretations:* first, interpretations are 'cleaned' by removing extraneous labels. For example, if a label that denotes the start or prefix of a pattern is not followed by a label that denotes the body of a pattern, it is removed. Also, labels that exceed the number of times they can appear in the underlying web form are removed. Second, interpretations that are completely subsumed by other interpretations are removed; and,

5. *Ranking interpretations:* the interpretations are first ranked by the number of labels they contain, then by the order in which the patterns are specified in the configuration file, and finally by the order in which the non-overlapping sets of segments were generated.

Figure 3.3 shows a step-by-step example for a travel-related query "Wycombe to shopping paradise Bicester North Camp". For sake of simplicity, assume that the system has two simple patterns p_1 = (*from* `station`), and p_2 = (*to* `station`). Here, *from* and *to* are optional prefix tokens (which are labeled as pr_1 and pr_2, respectively), and `station` is a type consisting of only three tokens: 'Wycombe', 'Bicester North', and 'North Camp'. Lastly, a pattern may occur at most once.

Step 1 underlines the segments of the input that are matched by one or more patterns. The first segment s_1 is matched by both patterns p_1 and p_2. Segment s_2 contains the prefix of pattern p_2, labeled as pr_2. The overlapping segments s_3 and s_4 are split in step 2, yielding the non-overlapping sets of segments γ_1 and

1. *Identifying segments:*

$$\underbrace{\underbrace{\text{Wycombe}}_{s_1(p_1,p_2)}\quad\underbrace{\text{to}}_{s_2(pr_2)}\quad\text{shopping paradise}\quad\underbrace{\underbrace{\text{Bicester}\quad\text{North}}_{s_3(p_1,p_2)}\quad\text{Camp}}_{s_4(p_1,p_2)}}$$

2. *Generating non-overlapping sets of segments:*

$$\Gamma = \{\ \ \gamma_1 = \{s_1, s_2, s_3\}\ \ ,\ \ \gamma_2 = \{s_1, s_2, s_4\}\ \ \}$$

3. *Generating interpretations:*

$$\begin{array}{lll} \gamma_1 \mapsto & i_1 = \{p_1, pr_2, p_1\} & \gamma_2 \mapsto \quad i_5 = \{p_1, pr_2, p_1\} \\ & i_2 = \{p_1, pr_2, p_2\} & \quad\quad\ \ i_6 = \{p_1, pr_2, p_2\} \\ & i_3 = \{p_2, pr_2, p_1\} & \quad\quad\ \ i_7 = \{p_2, pr_2, p_1\} \\ & i_4 = \{p_2, pr_2, p_2\} & \quad\quad\ \ i_8 = \{p_2, pr_2, p_2\} \end{array}$$

4. *Filtering interpretations:*

i)
$$\begin{array}{ll} i_1 = \{p_1, \cancel{pr_2, p_1}\} & i_5 = \{p_1, \cancel{pr_2, p_1}\} \\ i_2 = \{p_1, pr_2, p_2\} & i_6 = \{p_1, pr_2, p_2\} \\ i_3 = \{p_2, \cancel{pr_2}, p_1\} & i_7 = \{p_2, \cancel{pr_2}, p_1\} \\ i_4 = \{p_2, \cancel{pr_2, p_2}\} & i_8 = \{p_2, \cancel{pr_2, p_2}\} \end{array}$$

ii)
$$\begin{array}{ll} \cancel{i_1 = \{p_1\}} & \cancel{i_5 = \{p_1\}} \\ i_2 = \{p_1, pr_2, p_2\} & i_6 = \{p_1, pr_2, p_2\} \\ \cancel{i_3 = \{p_2, p_1\}} & \cancel{i_7 = \{p_2, p_1\}} \\ \cancel{i_4 = \{p_2\}} & \cancel{i_8 = \{p_2\}} \end{array}$$

5. *Ranking interpretations:*

$$\text{Ranked list} = [i_2,\ i_6]$$

Figure 3.3: An illustration of the query interpretation process.

γ_2. For each set $\gamma \in \Gamma$, step 3 generates all possible label combinations. Such a combination is in fact an interpretation, thus step 3 yields the interpretations $i_1 \ldots i_8$. Step 4-i removes the erroneous labels in each interpretation, in this case, it removes the prefix pr_2 when it is not followed by the pattern p_2, and it removes a label if it already occurred. In step 4-ii, an entire interpretation is removed if it is a subset of any other interpretation. At the end of step four, we are left with two interpretations i_2 and i_6, denoting "from Wycombe to Bicester North", and "from Wycombe to North Camp", respectively. Since the two interpretations contain an equal number of labels in the same order, the interpretations are ranked by the order in which the non-overlapping sets were generated. So, in step 5, i_2 is ranked higher than i_6.

3.2.2 Generating suggestions

The query interpretation process can be extended to generate suggestions which are interactive query expansions (White and Marchionini, 2007). The suggestions are generated based on the last segment in the interpretation, and filtered based on the entire interpretation. Three types of query suggestions can be generated and are listed next, along with the conditions that trigger them:

Token expansions. If the last segment contains just the first few characters of known tokens, then up to 10 possible token completions are shown in alphabetical order.

Pattern expansions. If the last segment denotes a complete prefix of a pattern, then this triggers token suggestions of the expected type, but only if the type was defined by a list of tokens. If the expected type was defined by a regular expression, no suggestions can be shown.

If the last segment denotes the body of some pattern and if the set of postfix strings of this pattern is non-empty, then the default (longest) postfix is shown.

Relation expansions. If the last segment denotes a token (like a car brand) for which there are related tokens (like the corresponding car models), then those tokens are shown.

3.2.3 Generating result snippets

The result generation process is another extension of the query interpretation process, and is triggered when the end-user submits the query. Each interpretation is post-processed in three steps:

1. For all fields that are not in the interpretation and for which default values are known, the system adds those fields and values to the interpretation. For example, the current time could be added to the example query given earlier as a default value.

2. The interpretation is discarded if it does not satisfy all constraints in the FTI's configuration (see Section 3.3.2), otherwise:

3. The field values of the interpretation are normalized if needed, and a result snippet is generated according to the FTI's configured rules (see Section 3.3.4).

3.3 Configuring the framework

The FTI can be configured by specifying the following items: *i*) the types and tokens, i.e., the web form's lexicon; *ii*) the constraints, *iii*) the patterns; and *iv*) the result generation rules. Together, these items are used for: identifying and limiting the set of valid queries, ranking query interpretations, and generating query suggestions.

3.3.1 Lexicon

The lexicon contains all known values that can be entered in the complex web form: it consists of a list of tokens, regular expressions, and a mapping from external strings to internal values. The lexicon closely corresponds to the syntactic constraints that are imposed by input fields. For example, a field can limit the number of characters, or only accept a token from a pre-defined list of tokens; hence the need for a list of tokens. Some fields pose other kinds of syntactic restrictions by using regular expressions, such as, allowing only numbers or zip-codes; hence the need for regular expressions. Finally, some fields (e.g., drop-down menus, radio buttons, and check boxes) can map external strings to internal form values; hence the need for mapping external to internal values.

3.3.2 Constraints

There are two types of constraints that can be used to determine if an interpretation is valid or not: *mandatory field constraints* indicate that certain fields should be present in an interpretation, regardless of the value in these fields; and, *value comparison constraints* indicate that if the interpretation contains values of two related fields, the values should satisfy some boolean function. Many web forms have constraints. For example, in the car-sales domain, a web form may require the end-user to enter at least a value for a car make (which is a mandatory field constraint). Further, if the end-user also wishes to enter a car model, it should correspond with the specified car make (which is a value constraint).

3.3.3 Patterns

A pattern consists of three parts: a prefix (context), a token (value), and a postfix (context) part. The prefix and postfix each denote a finite, possibly empty, list of tokens. Patterns take field-specific context words into account, so a pattern should be specified for each input field. This enforces that if a token is labeled as a prefix for (the pattern of) some field F, then the token that follows must always be labeled as a value token for (the pattern of) field F.

Consider the query “find me a trip to Amsterdam from Paris”. Here, the values are ‘Amsterdam’ and ‘Paris’, and the contexts are ‘to’ and ‘from’, respectively. A naive system that only considers values without their context could return two possible interpretations: either “from Amsterdam, to Paris”, or “from Paris, to Amsterdam”. However, such a response would not be intuitive. By using patterns, we can produce responses that are more intuitive.

Two patterns can be combined into a range pattern. A range pattern is useful for disambiguation. For example, the input ‘1000 - 2000 euros’ would be interpreted as ‘minimal 1000 euros and maximal 2000 euros’. Without range patterns, the system would encounter ‘1000’ (which could be a car model, a minimum price, or a maximum price) and ‘2000 euros’ (it could be minimum price or maximum price), which would have to be further processed in order to remove erroneous interpretations.

3.3.4 Result generation rules

The goal is to give short textual descriptions of a query interpretation, as shown in Figure 3.4. Since an interpretation can have too many fields and values to fit in a short description, a decision must be made of which ones to show to the user. Therefore, the title and the description are generated based on an ordered set of *field templates*. A field template specifies how a field's value should be displayed. A threshold M further specifies the maximum number of fields that can be shown in the short description, so up to M fields are displayed in the specified order. The snippet also contains the necessary parameters to retrieve the content. This includes the the web form's (action) URL, its http-request method (i.e., HTTP GET or HTTP POST), and the field-value pairs that are interpreted from the free-text query.

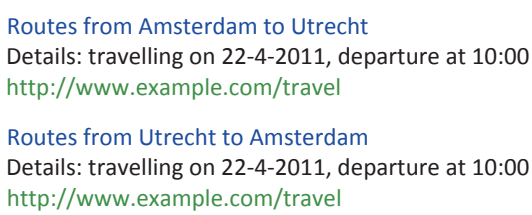


Figure 3.4: Query interpretations shown as result snippets.

3.3.5 An example configuration

Figure 3.5 depicts an example configuration file and shows how the lexicon, the constraints, the patterns, and the result generation rules, are specified.

The `tokens` element contains token instances. Each `instance` belongs to a specific *type*, has one internal value and a list of external values (treated as synonyms by the system). Multiple instances can belong to a single type.

The `pattern` element's `id` attribute contains the name of the input field to which the captured value should be assigned. A `capture` element specifies the *type* to be captured. The `prefix` and `postfix` elements specify a finite list of strings. This list may be specified by fully writing out all possibilities, or by a Kleene star-free regular expression, which will be automatically expanded to the list of possible strings. A pattern's `option` element relates a particular `prefix` with a particular `postfix`. The use of options is portrayed in the following example. Consider an input field to enter some minimum mileage, and three prefix-capture-postfix combinations: "minimum ... kilometers", "minimum number of kilometers ...", and "minimum number of kilometers ... kilometers". The latter of these combinations is peculiar and it would be parsed if we specified just one pattern option consisting of: "<prefix>minimum(number of kilometers)?</prefix>" and "<postfix>kilometers</postfix>". Moreover, the system would also generate the postfix suggestion "kilometers" if it parsed "minimum number of kilometers ...". To prevent this behavior, we could

specify two options, one containing just the prefix "<prefix>minimum number of kilometers</prefix>", and the other containing both "<prefix>minimum </prefix>" and "<postfix>kilometers</postfix>".

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<root>
  <tokens>
    <instance type='station' internal='1'>
      <external>amsterdam amstel</external>
      <external>amstel</external>
    </instance>
    <instance type='station' internal='2'>
      ...
  </tokens>
  <patterns>
    <pattern id='fromloc'>
      <option>
        <prefix>((depart(ing|ure)? )?from)?</prefix>
        <capture>station</capture>
      </option>
    </pattern>
    <pattern id='toloc'>
      ...
  </patterns>
  <constraints>
    <mandatory_fields>
      <fieldset>
        <field>fromloc</field>
        <field>toloc</field>
      </fieldset>
    </mandatory_fields>
    < field_field  not_equal='fromloc' to='toloc' />
  </constraints>
  <results>
    <url method='get'>http://www.example.com/search.html?loc1={fromloc}&amp;amp;...</url>
    <title  max='3' starttext='Example.com: search results for '>
      <fieldtemplate id='fromloc'  prefix='from '  postfix=' ' />
      <fieldtemplate id='toloc'  prefix='to '  postfix=' ' />
      ...
    </title>
    <defaults>
      <field  id='arrivalTime' external='arriving on '  internal='true'/>
    </defaults>
  </results>
</root>
```

Figure 3.5: An example configuration file.

The `constraints` element may contain: *i*) a list of mandatory field combinations; and *ii*) a list of field value comparisons, e.g., comparing the value of one field to the value of another or to some constant value. An interpretation is valid if it satisfies at least one of the mandatory field combinations, and all value comparisons.

Lastly, the `results` element specifies what the interpretation's title, description, and URL should look like. Here, a developer could also specify default (internal) values (with corresponding external values) for input fields. The `url` element specifies both the action-URL and http-request method. The `title` element (just like the omitted `description` element) contains an ordered list of

`field templates`. Each template corresponds to exactly one of the form's input fields, indicated by the `id` attribute. The title (as well as the description) is generated by listing the contents of its `start text` attribute and concatenating the contents of the active field templates, up to the specified `max` number of templates. A template is active if the value of the field it refers to is not empty.

3.4 Laboratory experiment

We evaluated this framework on an existing travel-planner site. Figure 3.1 (on page 28) illustrates the site's web form. Six information items can be specified in the form: a departure location, an arrival location, an optional via location, the time, the date, and a flag indicating whether the date and time are for arrival or departure. We configured the framework so it can interpret queries for this travel-planner site, and the resulting FTI is depicted in Figure 3.2 (also on page 28).

3.4.1 Experimental procedure

The experiment consisted of an offline part, an online part, and a questionnaire. We first prepared a randomly generated set of search tasks, or artificial information needs. These tasks were shown either graphically as a route on a map, or described textually as a sequence of (two or three) station names, a date, and a time. Dates were either relative, such as "next week Wednesday", or absolute, such as "1-2-2011". Times were described either textually, such as "half past ten", or numerically, such as "17.30".

During the offline part, the participants provided background information (e.g., age, gender, highest education), and wrote down their most recent travel question, at least, if they could remember it. Next, an information need was shown as a route on a map, along with a desired date and time. The participants were asked to fill out the complex web form on paper based on this information need. Likewise, they were shown a different information need and were asked to fill out the FTI on paper. Finally, the participants were shown a filled out complex web form, and they reformulated that into a question suitable for the FTI. We aimed to collect query formulations with as little bias to the question as possible. That is why we asked the participants to formulate a query from memory, and to formulate a query based on graphical instead of textual descriptions of the information need.

During the online part, the participant had to look up the departure and arrival times of 10 specific train routes. Each route or task was described textually, with a different order of the information items (i.e., the date, time, and locations), and with different wordings (e.g., *ten past one*, or *13:10*). The tasks were split into 5 tasks per system, and were handed to the participants on a piece of paper. Participants first familiarized themselves with an online system before performing 5 search tasks, then, they familiarized themselves with the

other system before performing the remaining 5 search tasks. We recorded the total search time for each set of tasks.

Regarding the questionnaire, we used a five-point Likert scale by which participants could indicate whether or not they thought that: the FTI was easy to use; they could find results faster using the FTI than using the complex form; the results of the FTI were correct; the FTI was nicer and/or better than the complex form; they preferred the FTI over the complex form. We also asked the participants to indicate the most negative and the most positive aspects of the system, and explained why they thought so.

3.4.2 Analysis

Using a Paired Samples T-Test (Kutner et al., 2005), we tested whether the task completion times of the FTI differed significantly ($p < 0.05$) from those of the complex web form, and whether the five-point Likert scale values differed significantly from neutral (i.e., the number '3'), also using the T-Test. We further evaluated the query formulation consistency by looking at the order of the information items. Each item was first replaced by a symbol: we replaced the 'from' (location) with A, 'to' with B, 'via' with V, 'date' with D, and the 'time' with T. For example, the input "from Amsterdam via Haarlem to The Hague, tomorrow at 10am." was represented as AVBDT. Using Kendall's τ (Kendall, 1975), we measured how the item order in the participant's formulation of the task description correlated with the item order in the original task description. Also, for each participant, we measured the average Kendall's τ over the combinations of that participant's own formulations.

3.5 Results

A total of 17 participants (11 male, 6 female) participated in the study. The age of the participants ranged from 21 to 66 (median: 27, mean: 32). Most participants were between the age of 21 and 33. Their educational background ranged from (under)graduate students in various studies to people working in healthcare, consultancy, and IT-software development. On average, the time to complete the experiment (including the questionnaire) took around 30 minutes per participant.

3.5.1 Opinions about the FTI

Comparing the free-text interface (FTI) against the complex web form, the participants indicated on a five-point Likert scale whether the FTI was: *faster*, *nicer*, *better*, and *preferred*. The results are given in Table 3.1, where '1' indicates full agreement, and '5' denotes the opposite. All results were in favor of the FTI and differed significantly from neutral ($p < 0.05$), except for the third aspect (i.e., whether the FTI is "better" than the complex form). On average, the participants felt they could search faster using the FTI than using the complex web

form, which was supported by the times measured for the web form and the FTI, shown in Table 3.2. The participants were significantly ($p = 0.032$) faster, by about 9%, when using the FTI instead of the complex form.

Table 3.1: Average results of the questionnaire, comparing the FTI to the complex web form on a five point Likert-Scale from 1 (full agreement) to 5 (full disagreement). Results in bold are significant ($p < 0.05$).

Question	Score
Faster	**2.4**
Nicer	**1.8**
Better	2.5
Preferred	**2.0**

Table 3.2: Average time in minutes to complete all five search tasks for each interface. The results differ significantly ($p = 0.032$).

Interface	Average time in minutes
Free-text interface	6.7
Complex web form	7.3

3.5.2 Speed and success rate

We counted the number of incorrect routes reported by the participants in the online experiment. Out of the 170 answers, 14 were wrong: 6 errors were made using the FTI, and 8 using the complex form. The most likely explanation for the errors is that the participants misread the task, and entered a wrong time or station name. Out of the 17 participants, 10 participants made zero errors, 2 participants made one error, 3 participants made two errors, and 2 participants made three errors. However, since we did not measure the time per query, we cannot omit the times for the failed queries for comparing the two systems. Yet if we use only the data from the 10 participants who made no errors, there is still a 9% difference in time, in favor of the FTI.

3.5.3 Pros and cons

The participants listed the most negative and most positive aspects of the FTI. The following **negative** aspects were mentioned: 24% of the participants indicated that there was no example or short manual (forcing the participants to 'just type in something, and it worked'); 18% indicated that the interface was too simple, e.g., it lacked pictures; and 12% disliked that they had to explicitly click a result snippet to view the travel plan, even when only a single result snippet was returned. The following **positive** aspects were mentioned: 41% of the participants liked how the system 'understood' dates like tomorrow and Tuesday, and written time like 'ten past nine'; 41% liked that you only had to type (without clicking on menus); 35% mentioned the query-suggestions as a useful feature; and 18% appreciated the fact that the input order of information items (e.g., time, date, places) did not matter.

3.5.4 Formulation consistency

When considering only the order of the information items[1] in a query, there were 17 different query formulations. As can be seen in Figure 3.6, the five most frequent online query formulations were: ABDT 41%, ABVDT 15%, and, tied at third place with 6%, were ABTD, DTABV, and TABVD.

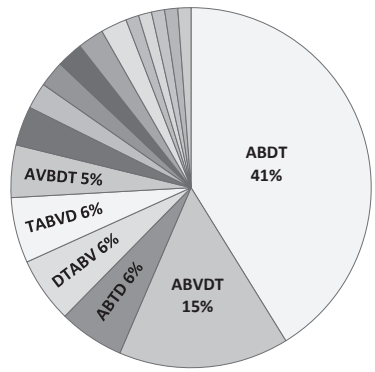


Figure 3.6: Distribution of the most frequent online query formulations

When inspecting whether or not the participants formulated their queries with the same order of information items as that of the online task descriptions, the mean Kendall's τ was **0.42**. Each of the five tasks of the FTI has a different information order. The task with the highest average τ (0.96) was sequenced ABDT, the other four tasks were BADT (0.67), TABVD (0.39), DTABV (0.09), and TBAD (-0.02). Two participants always followed the same information order of the task descriptions and had an average τ of 1.0 (though they used different wordings). Three participants had an average τ between 0.6 and 1.0, and the remaining twelve participants had an average less than or equal to 0.3.

The mean Kendall's τ for the (within participants) online query formulations was **0.64**. Six participants always formulated their questions in the same order, regardless of the task description, and had an average τ of 1; six other participants averaged between 0.7 and 0.9; and, the remaining five participants had an average τ less than 0.2.

Overall, the participants were highly consistent in their query formulations individually; however, there was considerable query variation between participants. Further, the task descriptions had little effect on the participants' query formulations; the moderate correlation (0.42) is most probably an artifact caused by participants consistently formulating their queries as ABDT. This explains the high correlations between the query formulations and the two tasks ABDT and BADT.

[1] i.e., the 'date' (D), 'time' (T), and the 'from' (A), 'to' (B), and 'via' (V) locations.

3.6 Discussion

3.6.1 Methodology and results

Query variation

We tried to prevent the participants from mindless copying of the task descriptions by presenting the tasks on paper instead of on screen. Nevertheless, the large number of different query formulations we collected was surprising, since: *i*) the participants could have just retyped the task descriptions; *ii*) there were only 17 participants; and *iii*) the travel-planner web form was relatively simple. With so much query variation in this limited scenario (in both the order of information items and wordings used), even higher variation might be expected in a more complex scenario.

Time difference

The paper&pencil-approach demanded manual time measurement. We measured the total time to complete all 5 search tasks, as it would be more difficult to obtain accurate measurements for individual tasks. Consequently, we could not determine whether the time per task decreased or not. Even though we noticed several times that participants were clearly experimenting with the free-text interface during the tests (as they were talking out loud, saying 'what if I typed...'), the average time of the FTI is still significantly lower than that of the complex web form.

3.6.2 Specialized features

In some cases, it could be handy to invoke a suitable function with the detected values as arguments. For instance, to extract the actual 'dd-mm-yyyy' time format from the input 'next week Friday', some function similar to 'getDate()' should be called to obtain the current date in order to calculate the intended date. The framework contains several pre-defined functions (e.g., for extracting dates and times) which can be invoked simply by specifying the field(s) that accept(s) a function value. Future versions of the framework will allow developers to add new functions.

3.6.3 Practicality of the framework

For end-users

Our work could add to the solution of the deep web problem. Given a free-text query, we can generate "real-time deep web results" (i.e., result snippets with valid query-URLs as data-entry points). Deep web search ultimately enhances our search experience by allowing end-users to search more content and to specify attribute or facet restrictions, besides merely a list of key words. Our work may benefit other search environments as well, particularly when there is some sort of

semi-structured search involved, examples could be desktop search and intranet search.

For providers

We believe that companies that offer deep web content will be encouraged to create their own configuration files for three reasons: *i*) we showed that end-users prefer such an interface; *ii*) (we claim that) it is easy to write a configuration file; and *iii*) visibility and user-friendliness are crucial for web companies.

Evidence for the last point can be found in a study by Alba et al. (2008), where they observed that: (1) the revenues of websites depend on the data that users see on the site's web pages; (2) websites are extremely motivated to ensure correctness, accuracy, and consistency on the web pages shown to the end-user; and (3) websites do not accord the same level of significance to the data delivered by the APIs. Alba et al. (2008) also show that web companies care greatly for their 'public image', since: *i*) selling products or services is difficult if users do not know about you; and *ii*) online users are more inclined to make a purchase if they feel positive about the website. Furthermore, the large number of articles on the web about search engine optimization strongly indicates that web companies make serious investments to increase their visibility to the users.

Our free-text interface for searching over web forms has the potential to both increase the visibility of a web site (i.e., deep web search, enabling search over otherwise uncrawlable data) and to provide more user-friendly search interfaces for websites that implement this interface.

3.7 Related work

The problem of filling out a complex web form for a given text query was tackled by Meng (1999). Meng used various statistical disambiguation techniques to infer the underlying meaning of a query. However, statistical approaches require large amounts of (training) data that are often difficult to obtain (e.g., they require domain-specific queries which often involve manual labeling). Instead of statistical disambiguation, the free-text interface described in this chapter adopts a rule-based approach to search for valid pattern combinations and returns a ranked list of alternative interpretations to the end-user.

Though the free-text interface is not a natural language interface, it is possible to enter queries that may resemble natural language statements. We now briefly consider certain aspects of natural language systems. One of the earliest natural language systems is Eliza (Weizenbaum, 1966). Eliza parses its input text using decomposition rules that are triggered by keywords. It generates responses based on reassembly rules pertaining to the decomposition rule. All rules are stored in a script which can be easily modified. During a session, a re-triggered decomposition rule may generate a different response. Unlike Eliza, the free-text interface described in this chapter generates responses based on a set of detected patterns rather than based on a single decomposition rule. Other natural language inter-

faces have been developed that were based on grammars (Burton, 1976; Hendrix et al., 1978; Carbonell et al., 1983; Carbonell and Hayes, 1981); however, the majority of these systems were application-specific which made it difficult to port the systems to different applications (Androutsopoulos et al., 1995). The difficulty of porting a system from one application (domain) to another is also apparent in information extraction systems, i.e., systems that extract all entities from large bodies of texts. To overcome the difficulty of porting, Appelt and Onyshkevych (1998) propose the Common Pattern Specification Language (CPSL). At the heart of the CPSL grammar are the *rules*. Each rule has a priority, a pattern and an action. Input matched by the pattern part can be operated on by the action part of the rule. Ambiguity arises when multiple rules match at a given input location, and is resolved as follows: the rule that matches the largest part of the input is preferred, and if two rules match the same portion of the input, the rule with the highest priority is preferred. In case of equal priorities of matching rules, the rule declared earlier in the specification file is preferred. Like Appelt and Onyshkevych, we propose a pattern specification language, and the patterns are used to scan the input text. However, we generate interactive query suggestions and we produce a ranked list of interpretations instead of a single interpretation.

Besides natural language systems, there are also keyword-based retrieval systems to search structured data. From this group of systems, one of the earliest systems was DataSpot (Dar et al., 1998). DataSpot used free-form queries and navigations to explore a *hyperbase* (a graph of associated elements) for publishing content of a database on the web. Recent systems (Demidova et al., 2010; Tran et al., 2007; Zhou et al., 2007; Tata and Lohman, 2008; Kandogan et al., 2006) generate a ranked list of structured queries or query interpretations, such that the user can select the right interpretation. These systems use a probabilistic or heuristic approach to rank the interpretations. However, most of these systems model the query as a bag of terms, disregarding the context of the extracted values, whereas the free-text interface described in this chapter uses patterns to capture the context of the extracted values.

All systems described so far process textual input using either grammar-based approaches, statistical approaches, or heuristic approaches. However, if we also consider spoken input, then we must consider the OVIS project which provides access to public transport information in the Netherlands through spoken input (Nederhof et al., 1997). An evaluation of the two natural language processing approaches within the OVIS project shows that the grammar-based approach performs much better than the data-oriented one (van Zanten et al., 1999). The grammar-based approach in the OVIS project uses a detailed grammar for Dutch, as well as acoustic evidence and n-gram statistics. Conceptually, the approach described in this chapter and the grammar-based approach in OVIS have two things in common. First, there is no need for grammatically well-formed input, as long as the system can extract the right pieces of information. Second, the interpretations are ranked by simple heuristics, the most important one being that the interpretation should account for as much of the input as possible.

The main difference between the two approaches is that one approach is based on grammatical processing whereas the other (ours) is not. Though grammars can be used to parse complex expressions which cannot be parsed using simple heuristics, grammars can be difficult to port to different languages or application domains.

3.8 Conclusion

In this chapter, we described a general, rule-based framework for interpreting an end-user's free-text query and extracting a structured query which represents a filled out web form. The framework can be used for both generating query suggestions on the fly and generating ranked query interpretations. Since the framework is configurable, we also described a novel specification language for describing a free-text interface (FTI) to a particular web form. Finally, we validated the framework by means of a user study, providing answers to the following research questions:

i) *Can a rule-based approach effectively be used to translate free-text queries into structured queries for a single web form?* Participants could successfully complete their search tasks using our free-text search system, which uses a rule-based approach to translate the free-text query. Therefore, we can conclude that a rule-based approach can effectively be used to translate free-text queries into structured queries for a single web form.

ii) *Do end-users prefer to use a single-field, free-text interface over a multi-field, complex web form to enter structured queries?* Yes, there was a statistically significant number of participants that preferred the free-text interface over the complex web form.

iii) *How much variation exists in the query formulations of end-users?* If we only consider the order of information items in a query (see Section 3.5.4), then the 85 free-text queries can be reduced to 17 unique query formulations. As can be expected, certain query formulations are more common than others, for instance, the top five query formulations make up almost three quarters of the free-text queries. However, the number of query formulations is considerable if we take into account that there were only 17 participants and that they were given the same 5 artificial tasks. This experiment suggests that we can expect a so called "long tail", or in other words, a large number of query formulations that do not occur very often.

iv) *Are end-users consistent in their query formulations?* Yes, the within-participants analysis showed a high correlation of 0.64 for the participants' query formulations.

v) *What are positive and negative aspects of the free-text interface?* The negative aspects are that participants do not know how they should formulate a query, and that the free-text interface lacks some short manual or example. The positive aspects are that the free-text interface "understands" written dates and times, and that it is possible to just type a query, without switching between or selecting different fields.

vi) *Is searching by means of a free-text interface faster than searching by means of a complex web form?* Yes, both subjective and objective measurements showed that participants finished their search tasks in less time if they used the free-text interface rather than the original complex web form. The time difference statistically significant, and on average, participants were 9% faster.

Overall, we can conclude that the participants could find results faster using the free-text interface rather than using the complex form, that they preferred the free-text interface over the complex form, and that they were highly consistent in their query formulations. Furthermore, we can conclude that there was considerable query variation between participants, even in this relatively simple scenario where we studied free-text queries containing only six information items.

Chapter 4

Free-text query log analysis

"Big Brother is Watching You -"
– *George Orwell*

In this chapter, we analyze how end-users interact with an experimental free-text search system. The system was publicly available and we logged over 30,000 queries from almost 12,000 users. An analysis of the log shows that there is great variety in query formulation, as over 400 query templates were found at least 4 times, and that over 92% of the queries were correctly interpreted. We also conducted a usability study and found that a significant number of users prefer the free-text interface over the complex web form for searching.

Parts of this chapter have been published in Tjin-Kam-Jet et al. (2012a).

4.1 Introduction

A free-text search system provides end-users with a simple means of accessing deep web content. Rather than entering structured queries in a complex, multi-field web form, end-users can enter their queries in a single-field, free-text interface. In the previous chapter, we focused on the nuts and bolts of the free-text search system, and described a rule-based approach to translate free-text queries into structured queries. In this chapter, we focus on the usability of the free-text search system: how do end-users interact with the system and is it well adapted to their needs? After all, the aim is to make it easier for end-users to search the web. To this end, we use a methodology that is referred as *search log analysis*, *query log analysis*, or *transaction log analysis*. The name given to the methodology varies depending on the research subject. For instance, since web search engines maintain log files that contain the end-user's search query, the shorter names *search log* or *query log* are often used instead of the more generic name *transaction log*, to refer to the type of log that is being analyzed. Logs are an unobtrusive way of collecting and analyzing significant amounts of data on the search behavior of a large number of users; query log analysis enables one to

"examine the characteristics of searching episodes in order to isolate trends and identify typical interactions between searchers and the system" (Jansen, 2006). Many studies have already applied query log analysis within various domains with the aim of improving, for example, retrieval functions (Joachims, 2002), spelling corrections (Ahmad and Kondrak, 2005), and query segmentation (Li et al., 2011; Hagen et al., 2011). In our case, our aim is of an exploratory nature: we use query log analysis to investigate how end-users phrase free-text queries. To start with, we measure the free-text query length and the search session length to find out whether these statistics differ from those of general web queries. While there are multiple definitions of search sessions, the definition we use, and the definitions used in the related work at the end of in this chapter, conform to the definition given by Gayo-Avello (2009): a search session consists of one or more successive queries related to a single information need or goal during, at most, one day. We also measure more specific aspects, such as, how often users reformulate their query, and how much query variation can be found in the query log. To quantify the amount of query variation, we count the different query *templates* that can be found in the query log. Recall that a free-text query describes values to fill out a form, each value is intended for a specific field. A query template is basically just the sequence of intended fields in a query.

In addition to the query log analysis, we analyzed Twitter messages for opinions about the system, and we used a questionnaire to evaluate the usability of both our free-text search system and of the complex web form.

In this chapter, we answer the following research questions:

1. Do end-users prefer to use our single-field, free-text interface over a multi-field, complex interface to enter structured queries?

2. How much variation does exist in the query formulations of end-users?

3. Are end-users consistent in their query formulations?

4. Do end-users make use of query suggestions?

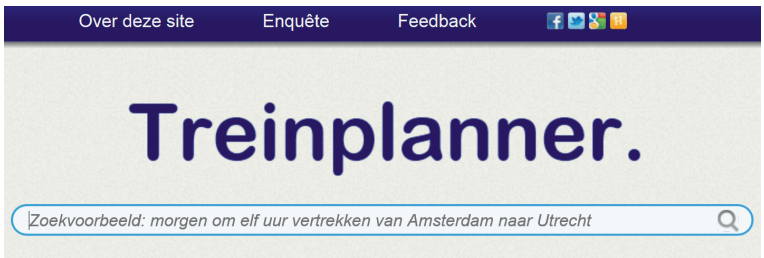
5. If the free-text search system returns unsatisfactory results, what was the cause?

6. What is the free-text search system's accuracy in correctly interpreting free-text queries?

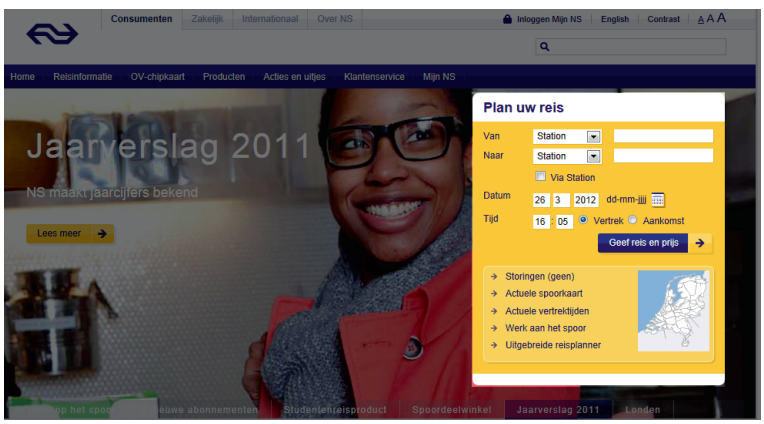
The remainder of this chapter is organized as follows. In Section 4.2, we describe our online experiment to gather query log data and questionnaire data for further analysis. In Section 4.3, we analyze a different aspect in each subsection and first describe how we analyzed the data, and then describe the results. In Section 4.4, we compare related query logs to ours, and we further discuss our work in Section 4.5. Finally, we conclude this chapter in Section 4.6.

4.2 Data acquisition

We developed a website[1] that serves as an alternative single-field interface for the travel-planning service of the Dutch railways[2]. We will use the terms *Trein-*



(a) The Treinplanner interface: a single-field search box. Inside the search box, an example query is shown (translated as *search example: tomorrow at eleven departing from Amsterdam to Utrecht*).



(b) The NSplanner interface: a multi-field search form. (We have emphasized the search form by darkening the picture surrounding of the form.)

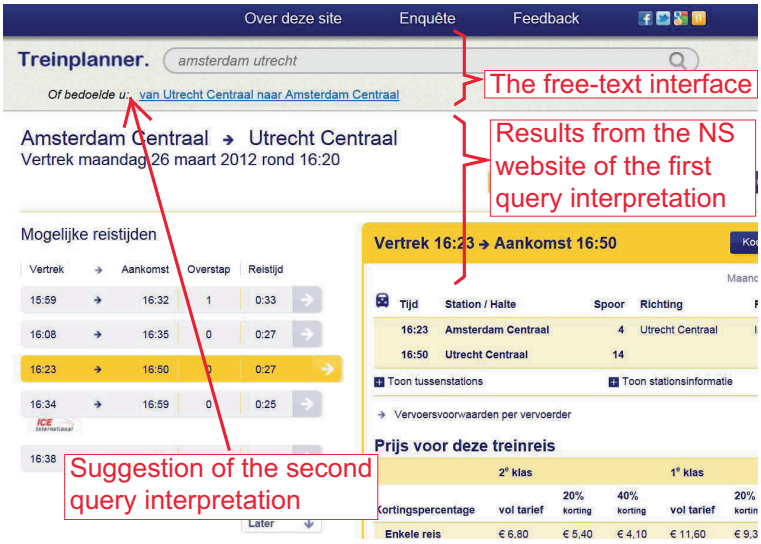
Figure 4.1: Single- and multi-field interfaces.

planner and *NSplanner* to refer to our alternative site and the Dutch railways site, respectively. Figure 4.1a illustrates the single-field interface of Treinplanner, and Figure 4.1b illustrates the multi-field interface of NSplanner. Using the rule-based approach introduced in Chapter 3, Treinplanner interprets the free-text queries and returns a list of interpretations (which denote filled out NSplanner

[1] `http://treinplanner.info` (May 14^{th} 2013)
[2] `http://www.ns.nl` (May 14^{th} 2013)

forms). If the end-user enters a valid free-text query, then Treinplanner returns one or more interpretations, i.e., ways in which NSplanner can be filled out. After receiving the list of interpretations, the client's browser automatically submits the top interpretation to NSplanner and displays the results from NSplanner to the user. Examples of an invalid query and a valid query are shown in Figure 4.2a and Figure 4.2b, respectively. The queries that were submitted to Treinplanner

(a) An invalid query, in this case due to a spelling error, Treinplanner returns the error message "geen aankomststation gevonden", which means "no destination station found".



(b) A valid query shows the results from NSplanner.

Figure 4.2: Results of the Treinplanner system.

were logged. Also, the method used to select query suggestions that were shown while entering a query was logged (i.e., using a mouse or a keyboard).

Visitors of the Treinplanner site can also provide feedback and participate in a study to assess the usability of NSplanner and of Treinplanner. They can provide feedback as either a general comment, or as specific feedback on the results for a particular query (i.e., giving a brief description of the error and annotating what they meant with the query). A visitor that wants to participate in the usability study must first enter some demographic information like his or her age, gender, and highest education level. The visitor must also indicate how much experience he or she has with both NSplanner and Treinplanner. A visitor who has enough experience with both systems can continue to the System Usability Scale (SUS) questionnaire. The SUS questionnaire is a simple, ten-item *Likert* scale giving a global view of subjective assessments of usability (Brooke, 1996). The questionnaire is administered for each system. This results in two scores ranging from 0 to 100, a higher score indicating a better usability of the system.

We gathered participants for our experiment through announcements on social media and through a number of press releases. In particular, we issued a tweet which was also re-tweeted by the official Twitter channel of the NS (which has over 30,000 followers). Furthermore, we crawled all tweets containing the keyword "Treinplanner" to gain qualitative data on the end-users' opinion (on Twitter) about using this single-field search interface to actually perform structured search. The data used for our analysis (query log data, usability study, opinion, tweets) were collected in 2012 from January the 23^{rd} till March the 25^{th}.

4.3 Free-text query log analysis and results

As stated by Jansen (2006), the first step after collecting data in a query log is to clean and prepare the data before it is further analyzed. Therefore, in Section 4.3.1, we first explain how we cleaned and prepared the data. Then we describe the following: in Section 4.3.2, we manually analyze a sample of the queries to find out what mistakes are made, and whether they are made by the system or by the end-user; in Section 4.3.3, we analyze the sessions. Specifically, we investigate how end-users change their queries during a session, we classify the types of changes, and we analyze whether end-users are consistent in the way they formulate their queries; in Section 4.3.4, we examine the amount of query variation; in Section 4.3.3, we examine how end-users make use of the query suggestions shown while typing a query; in Section 4.3.6, we present the findings from our questionnaire; and finally, in Section 4.3.7, we examine the opinions about Treinplanner from Twitter messages.

4.3.1 Cleaning and grouping the data

Cleaning the query log data involved two steps. First, we removed all queries issued by ourselves (based on our ip addresses). Second, we discarded all queries that did not contain a cookie ID (this could happen when client browsers did

not accept cookies). By using cookies, we can easily discriminate the queries of one client from another. Next, we grouped all queries by their cookie ID and then segmented those queries into *search episodes* and *search sessions* using the geometric session detection method proposed by Gayo-Avello (2009). A search episode denotes all actions performed by a single user within a search engine during, at most, one day. An episode comprises one or more sessions, and each session comprises one or more successive queries related to one single information need or goal.

Results

We collected a total of 36,271 queries, which, after cleaning, resulted in 30,472 queries. These queries were issued by 11,933 different clients, based on the client's cookie ID. The distributions of query lengths over valid, invalid, and all queries is depicted in Figure 4.3. Valid queries are slightly longer than invalid queries on average. More detailed query length statistics are given in Table 4.1. After grouping the queries by client and applying the geometric query segmentation method, we found 13,058 search episodes and 14,541 search sessions. This data

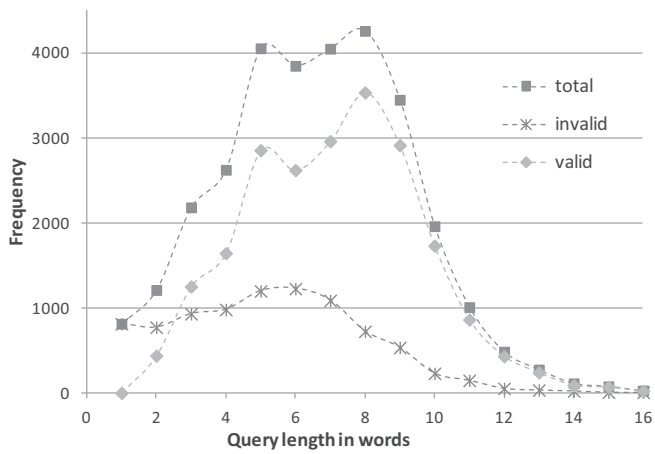


Figure 4.3: Distribution of query lengths.

Table 4.1: Query length in characters (c), words (w).

	Total Queries	avg. c (w)	min. c (w)	max. c (w)	std.dev. c (w)
Invalid	29%	33 (5)	1 (1)	100 (43)	16 (3)
Valid	71%	45 (7)	9 (2)	141 (34)	14 (3)
All	100%	42 (7)	1 (1)	141 (43)	16 (3)

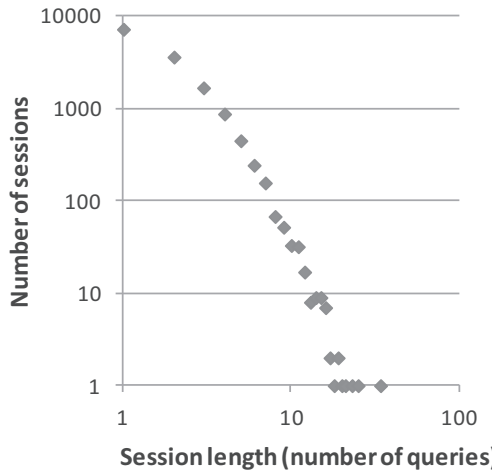


Figure 4.4: Distribution of session lengths.

Table 4.2: Query log entries by granularity level.

	Total
Queries	30,472
Sessions	14,541
Episodes	13,058
Clients	11,933

is summarized in Table 4.2. The distribution of the number of queries within a session is depicted on a log-log plot in Figure 4.4. It can be seen from the figure that the session length follows a Zipfian distribution (the dots are approximately linearly aligned), which is consistent with findings in related literature (Spink et al., 2001).

4.3.2 Manual sample analysis

We counted how many times Treinplanner incorrectly interpreted a query. We also determined and classified the cause of these mistakes. Let us first introduce some new terminology. Treinplanner only returns results if: *i*) it can detect at least a departure and an arrival station name in the query, so if the query contains *sufficient* information; and if *ii*) none of the departure, arrival, or via-stations are the same, so if the information is *non-conflicting*. Otherwise, Treinplanner either indicates that some information is missing or that there is conflicting information. We refer to queries that contain sufficient and non-conflicting information as *valid* queries. Conversely, we refer to queries that contain insufficient or conflicting information as *invalid* queries. Since Treinplanner does not apply spelling

corrections, queries with spelling errors for which the search intent is obvious, like "Amst*re*dam to Utrecht", are invalid because Treinplanner could not detect at least two stations[3].

If a user enters a valid query, it does not necessarily mean that the query will be *correctly interpreted*, e.g., Treinplanner might have extracted the wrong date or wrong time, the wrong station names, or might have missed some important pieces of information. Therefore, we assessed the correctness of the interpretations by manually inspecting a random sample of the query log. We distinguish between: *true negatives*, i.e., invalid queries that Treinplanner correctly marked as invalid; *false negatives*, i.e., valid queries that Treinplanner incorrectly marked as invalid; *true positives*, i.e., valid queries from which Treinplanner correctly extracted the right fields and values; and, *false positives*, i.e., valid queries from which Treinplanner extracted the wrong fields and values.

In our manual inspection, we used the following rules of thumb. For queries that are marked as invalid by Treinplanner: if a human annotator could make sense from the query (by finding sufficient and non-conflicting information) we marked the query as a false negative. For queries that are marked as valid by Treinplanner: if a human annotator could find meaningful pieces of information that were misinterpreted by Treinplanner, we marked the query as a false positive. In both cases, we noted what caused the error (e.g., a spelling mistake, a synonym that is not in Treinplanner's lexicon, or some concatenated words).

Finally, we manually analyzed the explicit user feedback on wrong query interpretations such as when Treinplanner has mixed up the intended input fields of the stations, or has failed to recognize a relevant piece of information.

Results

We randomly selected a total of 1,500 queries, or 5% of the query log: 750 valid queries, and 750 invalid queries. Table 4.3 summarizes our findings, it reports the number of accurate results (true positives and true negatives) and the number of mistakes (false positives and false negatives).

Table 4.3: Contingency table of the system's query interpretation accuracy.

System indicates valid query		System indicates invalid query	
Correctly interpreted (true positive)	704	66	Wrongly marked as invalid (false negative)
Incorrectly interpreted (false positive)	46	684	Correctly marked as invalid (true negative)

False positives. Out of the 750 queries that the system indicated as valid, 46 (6.1%) were incorrectly interpreted. In all false positives, Treinplanner failed to interpret certain meaningful parts of the query, such as: "1800", "ten past 2", "next month", and "around rush hour". Other frequent mistakes were the lack

[3] Amst*re*dam should be spelled as Amst*er*dam

of spaces between two meaningful parts (e.g., "Wednesday10 am") and spelling errors (e.g., "Amsterdam Utrecht elevn o'clock").

False negatives. Out of the 750 queries that the system indicated as invalid, 66 (8.8%) were wrongly marked as invalid. The most frequent mistake, causing just over half of the false negatives, could be attributed to the lack of certain synonyms for some station names. Therefore, given a query that is perfectly interpretable by a human annotator, Treinplanner would indicate that there was no departure station, or no arrival station. The second most frequent mistake was the use of dashes as a separator between meaningful parts, like "Amsterdam-Utrecht". The third reason, causing 7 false negatives, was that if a query was phrased in a particular way[4], there would be no result. Further, we observed many spelling errors and a frequent lack of spaces between two meaningful parts in a query. However, even if those mistakes could be corrected, most of those queries would still be invalid as they only mentioned one station or something that is not in the scope of Treinplanner (e.g., like street or place names).

Accuracy. In this random sample of 1,500 manually inspected queries, we found that the system made 110 mistakes. This means that 1,390 out of 1,500 queries were handled correctly, resulting in an accuracy of 92.7%.

Explicit feedback. There were 31 queries reported by users for which Treinplanner gave a misinterpreted result. Reasons for the most frequently reported mistakes could be attributed to: an incomplete lexicon (26%); usage of particular terms by which users indicate arrival time instead of departure time (23%); and, issues relating to either the recognition of times or the interpretation of times — e.g., five o'clock, but is that in the evening or morning? (31%).

4.3.3 Session analysis

We performed three kinds of analysis. First, we analyzed the sessions in terms of their query validity. We counted the number of sessions that started with a valid query; and in sessions that did not start with a valid query, we counted how many queries were needed on average to reach a valid query. Also, we counted the number of sessions did not have any valid queries at all.

Second, we analyzed the sessions in terms of the development or change of queries in a session. We classified successive queries within a session into one of the following query types:

1. **Lexical repeat.** A successive query is a lexical repetition of its previous query if the Levenshtein distance between the characters of both queries is less than some threshold[5].

2. **Specialization.** A successive query is a specialization if the set of terms of the successive query is a proper superset of the set of terms of the preced-

[4] A date (consisting of a day and a month), followed by a two-digit number, followed by some indication of a time (e.g., "am", "pm", "hour")

[5] We used different thresholds depending on the length l (in number of characters) of the longest query: 0, if $1 \leq l \leq 3$; 1, if $4 \leq l \leq 14$; 2, if $15 \leq l \leq 24$; and 3 for $l \geq 25$

ing query. For example, when one subsequently enters "Amsterdam" and "Amsterdam Utrecht", the latter query is a specialization query.

3. **Generalization.** A successive query is a generalization if the set of terms of the successive query is a proper subset of the set of terms of the preceding query. A generalization is the opposite of a specialization. For example, when one subsequently enters "Amsterdam Utrecht" and "Amsterdam", the latter query is a generalization query.

4. **Mixture.** A successive query is a mixture if both queries contain at least one term that is not in the other query, and if the intersection of the set of terms of both queries is not empty. For example, when one subsequently enters "Amsterdam Utrecht" and "Amsterdam Rotterdam", the latter query is a mixture query.

5. **New.** A successive query is new if the intersection of the set of terms of both queries is empty, while the queries themselves are not empty. For example, when one subsequently enters "Amsterdam Utrecht" and "Rotterdam Tilburg", the latter query is a new query.

6. **Semantic repeat.** A successive query is a semantic repetition of its previous query if the same set of key-value pairs can be extracted from both queries. For example, when one first enters "from Amsterdam to Utrecht" and then "to Utrecht from Amsterdam" (or just "Amsterdam Utrecht"), then the latter query is a semantic repetition.

Successive queries within a session were classified according to the specified order of the classes. That is, if a query could not be classified as the first class, *lexical repeat*, we tried the second class, *specialization*. If it could not be classified as specialization, we tried the third, and so on, until the query was classified. This way, we ensured that a query belonged to, *at most*, one class. Note the asymmetric output of this procedure: queries from the class lexical repeat could in theory also belong to the class semantic repeat; however, queries from the class semantic repeat could never belong to the class lexical repeat. Related work on query log analysis (see Section 4.1) normally concerns keyword search queries; however, due to our experimental setup, our query log contains free-text queries that contain structured key-value pairs. Therefore, we can classify a successive query as a *semantic* repetition with its previous query if the set of key-value pairs of both queries are the same.

Third, we analyzed the query formulation consistency and compared it with our findings from the laboratory experiment in Chapter 3. We adopted the same method as in the laboratory experiment. That is, we looked at the order of the information items in a query, and calculated the average Kendall's τ over all queries of a single user. In the laboratory experiment, each participant submitted 5 queries; therefore, we used sessions containing exactly five valid queries for our analysis.

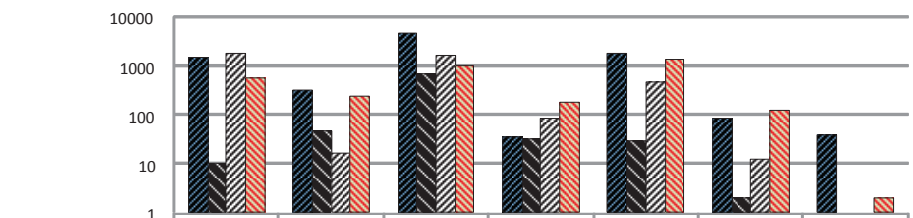
Results

Out of the 14,541 sessions, 7,256 sessions consisted of just one query, and 7,285 consisted of two or more queries. Table 4.4 shows how many sessions started with a valid query and how many started with an invalid query. It also shows the number of single-query and multi-query sessions. In the majority (67.8%) of the sessions, the very first query is already successful and returns results. From the 3,795 multi-query sessions that started with an invalid query, 3,192 (84%) contained at least one valid query, and 603 (16%) contained no valid queries. Further analysis showed that, if the initial query is invalid, it takes 1.7 queries on average to reach a valid query.

Table 4.4: Sessions grouped by their length and by their initial query's validity.

	Sessions where the first query is		
Session length	Valid	Invalid	Total
Single-query sessions	6,367 (87.7%)	889 (12.3%)	7,256 (100%)
Multi-query sessions	3,490 (47.9%)	3,795 (52.1%)	7,285 (100%)
Total	9,857 (67.8%)	4,684 (32.2%)	14,541 (100%)

During the classification, we noted that some queries could not be assigned to any class, and inspected those queries manually. Almost all unclassifiable pairs of queries looked like this: "tomorrow from Utrecht to The Hague" and "tomorrow from The Hague to Utrecht". That is, the queries contained the same terms, which is why they could not be classified as any of the specialization, generalization, mixture, or new query types. Further, the queries were neither lexical nor semantic repetitions of each other. Whether these queries should still be considered as mixture queries or as a new query type is open for discussion. For now, we will use the query type *other* to refer to these queries. Figure 4.5 depicts the query type distribution of successive queries within a session. The types are split into valid and invalid queries. From this figure we can see that,



	SPECIALIZATION	GENERALIZATION	MIX	NEW	LEX REPEAT	SEM REPEAT	OTHER
valid→valid	1490	305	4475	33	1647	78	36
valid→invalid	10	44	667	31	29	2	1
invalid→valid	1676	16	1528	82	452	12	1
invalid→invalid	544	236	974	176	1270	115	2

Figure 4.5: Distribution of query modification types in multi-query sessions.

apart from new and semantic repetition queries, the majority of successive queries within a session end up as valid queries. Specialization and mixture queries perform particularly well. A plausible explanation is that users naturally submit a specialization query after an invalid query when, for instance, the preceding invalid query did not contain both a departure and arrival station.

Regarding query formulation consistency, there were 125 sessions containing exactly five valid queries. In these sessions, the average correlation of the within-user query formulation was 0.88. This is a high correlation, which up to some degree, can be explained by the fact that some sessions contained queries of the type lexical repeat. Yet even in the sessions without lexical repeat, the average correlation (over the 50 sessions without lexical repeat) was 0.84. This suggests that end-users formulate their queries in a consistent manner.

4.3.4 Template extraction

Agarwal et al. (2010) consider a query as a sequence of keywords and domain attributes, e.g., station, date, or time. By abstracting away the specific instances of each attribute, a query can be represented by its query *template*. We extracted the query templates using the Treinplanner system in an automated process. Pieces of text that it recognized as stations were marked as `station`; keywords indicating the role of a station (departure, destination, or intermediate) were marked as *from*, *to*, or *via*, respectively; dates, times, or keywords indicating time of departure or time of arrival, were marked as `date`, `time`, or `arr/dep`, respectively. Finally, words that were not recognized were marked as `oov` (out-of-vocabulary). For instance, the query "Amsterdam, destination: Utrecht, arrival time 10 pm" is represented by the template: `station` *`to`* `station arr/dep time`.

Table 4.5: Most frequent query templates.

Template	frequency
`[INVALID] no destination given`	15.8%
`[INVALID] no departure given`	12.6%
`date time` *`from`* `station` *`to`* `station`	7.7%
`date` *`from`* `station` *`to`* `station`	6.8%
`station` *`to`* `station`	3.4%
`station station`	2.8%
`from` `station` *`to`* `station`	2.3%
`date` *`from`* `station` *`to`* `station time`	2.1%
`date time arr/dep` *`from`* `station` *`to`* `station`	2.0%
`oov` *`from`* `station` *`to`* `station`	1.9%
`date` *`from`* `station` *`to`* `station arr/dep time`	1.1%
`date station station`	1.0%
...	40.5%

Results

Table 4.5 lists the 12 most frequent templates. The first and second templates denote the type of error in invalid queries as perceived by Treinplanner. For example, if a users enters "to Amsterdam via Utrecht" or just "to Amsterdam", then Treinplanner would indicate that no departure station was given. Over 400 templates occurred at least 4 times, and in total almost 1,500 templates were logged. This shows that there is great variety in how users formulate structured information needs.

4.3.5 Query suggestion usage

Treinplanner's interface shows query suggestions while typing, but are these suggestions used? And does it have a positive effect to use the suggestions, i.e., do users who use query suggestions issue more valid queries than users who do not use suggestions? We compared the queries where no suggestions were selected to those where a suggestion was used once or multiple times. We applied the Pearson's chi-square test (χ^2) to find out whether or not using suggestions and entering valid queries are correlated (Manning and Schütze, 1999). The χ^2 test checks for dependence and does not assume a normal probability distribution.

Results

Table 4.6 summarizes how many times query suggestions were used, and describes how the suggestions were selected. Most queries were issued without using query suggestions. However, end-users could also have copied a suggestion simply by typing what was suggested, but we cannot know from the query log whether end-users actually saw the suggestions, or whether they were only looking at their keyboard while typing. If suggestions are used, we can see that they are more often selected using a mouse rather than using a keyboard. On the one hand, this is surprising since we assumed that the ease of formulating a query using only a keyboard would outweigh the effort of grabbing the mouse to make a selection from query suggestions. On the other hand, most queries were typed in completely, which may indicate that users do prefer to just use the keyboard. Nevertheless, the results suggest that one should select the query suggestions more often since the ratio between valid and invalid queries, when selecting and making use of the query suggestion (ratio of 3.1 : 1.0), is significantly higher ($p < 0.001$) than when not making use of the query suggestions (ratio of 2.3 : 1.0).

4.3.6 Usability study – quantitative analysis

We compared the usability scores of Treinplanner to those of NSplanner. We checked whether the scores differed significantly using the paired T-test, with $p < 0.05$. Since the T-test assumes a normal distribution on the data and since we do not know for certain that our data follows a normal distribution, we also applied the statistically weaker sign test which does make this assumption. The

tests were performed over all participants and over groups of participants to check whether or not the difference is significant across all groups.

Results

There were 116 participants (99 male and 17 female) that opted in and completed our online survey. The modal, average, and standard deviation of the ages of the participants were 25, 40, and 19 years, respectively. The education background of the participants is summarized in Table 4.7. The Simple Usability Scale (SUS) scores for NSplanner and Treinplanner are shown in Table 4.8. Overall, a sig-

Table 4.6: Statistics on the usage of query suggestions.

	Total queries	Valid queries	Invalid queries
No use of suggestions	22,308	15,517	6,791
Use of suggestions	8,164	6,171	1,993
Suggestion selection method			
Mouse only	6,447	4,902	1,545
Keyboard only	1,651	1,209	442
Both mouse and keyboard	66	60	6

Table 4.7: Participant's education level.

Education level	Studying	Completed
Elementary or middle school	0	1
High or junior high school	4	19
College or university	32	60

Table 4.8: Survey scores grouped by participants' experience with search engines (e.g., Bing, Google, or Yahoo). Numbers in bold are statistically significant.

	Group size	NSplanner score	Treinplanner score
All participants	116	71	**84**
Participants grouped by daily search engine use (e.g., Bing, Google, or Yahoo)			
Frequent—over 20 times a day	54	68	**85**
Often—5 to 20 times a day	53	72	**84**
Rare—less than 5 times a day	9	78	80

nificant number of participants prefer the Treinplanner system (first row in the table). By grouping the participants by their daily usage of search engines, we can see that as the daily usage increases, the difference between interface preference grows larger. A significant number of participants in the groups 'frequent' and 'often' prefer the single-field interface, whereas in the group 'rare' there is almost no difference between the two systems. In other words, the more frequent a participant uses search engines, the more that participant will prefer the free-text interface over the complex web form.

4.3.7 User opinions – qualitative analysis

We also performed a qualitative analysis of the user's opinions about Treinplanner. The user opinions were obtained from comments that were given at the Treinplanner site, and from tweets that mentioned "Treinplanner". From the comments, we counted how many mentioned positive aspects, negative aspects, bugs, or possible improvements. From the tweets, we counted how many mentioned positive aspects, and how many mentioned negative aspects of the Treinplanner system. Tweets that were otherwise neutral were discarded.

Results

During the period of January the 25^{th} to February the 29^{th} in 2012, we collected over 150 opinions on the Treinplanner site and over 300 tweets mentioning the Treinplanner. Out of all 150 opinions, 64% were positive, expressing statements like "great initiative", "nice!","works great and fast"; 21% gave suggestions on for improving Treinplanner; 6% were negative or skeptical of Treinplanner; and, 9% indicated mistakes like how a query was misinterpreted. It was even uttered 9 times, asking if such an interface would become available for another major travel planning site. From the Twitter messages, 6 tweets mentioned a bug or expressed doubts about Treinplanner (e.g., "is the Treinplanner interface an improvement?", or "the site does not work"), the rest were positive (re)tweets.

4.4 Comparison with other web search logs

According to a study of an Altavista general web search log, end-users tend to formulate short queries containing 2.3 words per query on average. Furthermore, search sessions are relatively short, containing just 2 queries on average (Silverstein et al., 1999). The Altavista log is very large and consists of about 1 billion entries. Two other search log studies, an Excite search log study (Spink et al., 2001) and an MSN search log study (Bendersky and Croft, 2009), have reported similar findings on query and session lengths. In the MSN study it was noted that, if we assumed direct relation between the reciprocal rank of the clicks and the effectiveness of the retrieval, the effectiveness decreases as the query length increases. The Excite and MSN logs contain roughly 1 million entries and 15 million entries, respectively. Compared to our query length and session length

findings, we see that normal web queries are over two times shorter than free-text queries, but that the number of queries in a session is comparable. This can be explained by the fact that users essentially specify the field values for a complex web form in their free-text query, so naturally their queries will contain more words. Also, due to the high accuracy of the system, end-users could quickly find the information they were looking for, which explains the relatively short sessions.

Other recent studies like Agarwal et al. (2010) and Li et al. (2009) have analyzed general search logs to extract query templates or to extract structured information from queries. The focus of these studies was to develop a suitable information extraction algorithm, which can explain why they did not report query length and session length statistics. However, these studies provide evidence that many queries in a general web search log follow a certain query template. This further motivates our study of the complexity of free-text queries, and how end-users search for structured content in a single-field interface, since web users are issuing such kinds of queries. As an aside, Hearst (2011) notes a trend toward more "natural" user interfaces in which end-users could use, amongst others, natural language queries. Our free-text query brings us one step closer toward more natural user interfaces.

4.5 Discussion

There are two issues related to our data collection method to be addressed here. First, while the collected queries may reflect the expected types of queries in a production system, it is possible that our sample contains an overestimate of test-queries. Some users may have been inclined to explore the limits of Treinplanner and issued many different queries not necessarily related to a real information need. However, even if our data sample contained an overestimate of test-queries, we expect that this would mainly impact the query variation results and the system's query interpretation accuracy: in the worst case, we may have reported a greater query variety and a lower system accuracy than what would actually be the case. Second, we note that the results presented in Section 4.3.6 and Section 4.3.7 were based on not-so-random sampling. It is possible that only those users who would like to say something were analyzed and not the silent majority (if any). That is, it could have been the case that only those people who were positive about our system gave positive feedback and people who were not positive simply left our website.

Regarding the results, one remarkable observation is the relatively high number of queries with insufficient or conflicting information. Search log studies typically deal with keyword queries since the search engine in question provides a keyword search service to its users. In contrast, Treinplanner allows its users to enter anything ranging from keyword queries up to complete natural language sentences. Therefore, the question arises whether the users' expectations will match the actual capabilities of Treinplanner. That is, will the users' queries be such that they make good use of Treinplanner's query understanding capabilities,

will they be too simple, or will they be too complex? This has also been referred to as the *habitability problem* (Thompson et al., 2005). After manually inspecting a sample of the query log, we noticed that some users had a dialog system in mind. For instance, after submitting a query, they submit a subsequent query like "no, from amsterdam". Another wrongful expectation was that Treinplanner could interpret more than Dutch train station names, e.g., like street names or station names in foreign countries. This is partly reflected by the large proportion (over 55%) of invalid to invalid semantic repetitions. In other words, users would re-phrase their information needs, which contained unrecognized places, hoping or expecting that Treinplanner would then understand their query if it was formulated differently. Finally, many queries contained just one station name. This could indicate that users expected Treinplanner to know their current location (similar to how modern mobile devices know their geographic location) and assume it as their departure location. This was explicitly given as feedback by some users. While the habitability problem might account for some of these invalid queries, we again point out the possibility that some users were plainly biased toward testing the limits of Treinplanner.

Another remarkable observation is that the most likely valid-query template (`date time` *`from`* `station` *`to`* `station`) corresponds to the template of the example query shown in the Treinplanner interface. In contrast, in the laboratory experiment of the previous chapter, the majority of the users did not follow the templates of the descriptions of the search tasks. Yet despite the apparent bias towards the template of the example query, we still found almost 1,500 unique templates and over 400 templates which occurred at least 4 times.

4.6 Conclusion

We have described an in-depth analysis of how free-text queries are formulated. In addition, we have conducted a user study and analyzed user opinions to find out whether or not users prefer a single-field interface or a multi-field interface for formulating structured queries. The research questions of this chapter can be answered as follows.

i) *Do end-users prefer to use our single-field, free-text interface over a multi-field, complex interface to enter structured queries?* End-users generally prefer the free-text interface. From our questionnaire we can conclude that the more experience a user has with general web search engines, the more pronounced the preference is for the free-text interface. Also, looking at the qualitative data obtained from the tweets and user opinions, the positive responses towards the free-text interface are most pertinent, which support our findings that end-users prefer the free-text interface over a complex interface.

ii) *How much variation does exist in the query formulations of end-users?* We have extracted almost 1,500 unique templates describing how users formulate their queries. With over 400 query templates occurring at least 4 times, we can say that there is great variation in how queries are formulated.

iii) *Are end-users consistent in their query formulations?* Yes, the within-user analysis showed a high correlation of 0.88 for the query formulations of end-users. In the previous chapter, we concluded that end-users are consistent in their query formulations in a laboratory setting. Now, we can conclude that end-users are also consistent in their query formulations in a real world setting.

iv) *Do end-users make use of query suggestions?* End-users actively selected a query suggestion in 27% of the queries. The mouse was the most frequently used medium to select a suggestion. On the one hand, this is surprising since we assumed that the ease of formulating a query using only a keyboard would outweigh the effort of grabbing the mouse to select a query suggestion. On the other hand, most queries were typed in completely, which may indicate that users do prefer to just use the keyboard. Further, the proportion of valid queries is larger in the group of queries where suggestions are used, than in the group where suggestions are not used. This shows the benefits of query suggestions.

v) *If the free-text search system returns unsatisfactory results, what was the cause?* The most common cause leading to unsatisfactory results could be attributed to the lack of certain synonyms for some station names. Especially when end-users use their own abbreviations for station names. Other causes include spelling errors made by end-users, or when end-users forget to separate words with a white space.

vi) *What is the free-text search system's accuracy in correctly interpreting free-text queries?* In this experiment we used a rule-based system to interpret and translate the user queries. Our manual analysis of 5% of the query log showed that 7.3% of the queries were incorrectly interpreted. The reasons for most mistakes could be attributed to the query containing spelling errors, or Treinplanner not containing enough synonyms. Certain types of errors might be challenging for a rule-based approach; but overall, Treinplanner correctly interpreted most queries with an accuracy of over 92%.

Overall, we can conclude that a very flexible system is needed to handle the large variety in free-text query formulations. Also, spelling errors are an important problem that can be partially solved by query suggestions. Finally, users suffer from the habitability problem which partially explains why they enter invalid queries. However, users are able to rephrase their query into a valid query, requiring less than two reformulations on average.

Chapter 5

A probabilistic approach to translate free-text queries into structured queries

"It's choice – not chance – that determines your destiny -"
– *Jean Nidetch*

In this chapter, we investigate how to translate a free-text query into a structured query when the free-text query contains OOV (out-of-vocabulary) words. In the previous chapters we discarded such words because we focused on cases where it was not necessary to use OOV words to fill out a form. However, in this chapter, we focus instead on cases where a form must be filled out using some OOV words in the query. This means that any sequence of one or more OOV words, called a token*, might be used to fill out a form. Therefore, we introduce three novel tokenization methods to split a free-text query into possible tokens, and four novel token models to decide which tokens should most likely be used. We investigate which combination of tokenization method and token models increases our odds of, essentially, guessing how to correctly fill out a form.*

Parts of this chapter have been published in Tjin-Kam-Jet et al. (2012b).

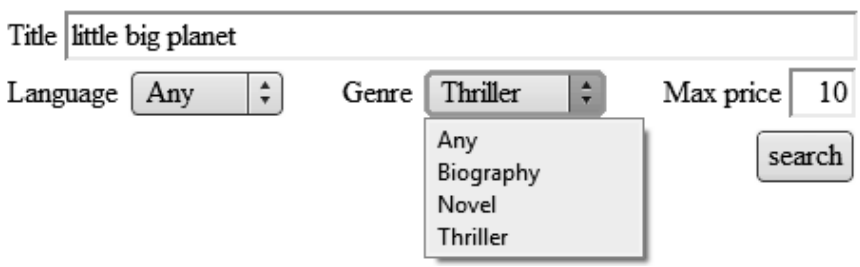
5.1 Introduction

A free-text interface enables end-users to enter free-text queries with words like "tomorrow" instead of fully written out dates like "05-08-2013". In a way, it inspires natural language-like queries, such as, "Tomorrow, I need to go from Amsterdam to The Hague". Queries, and in particular natural language queries, can be problematic because end-users may unknowingly use words that are not in the system's dictionary or vocabulary. Such words are referred to as *out-of-vocabulary* (OOV) words. If a query contains OOV words, a deep website may

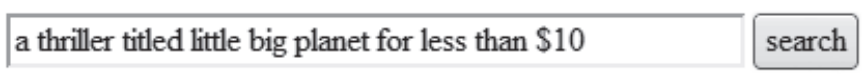
simply return no results, indicating that there is no item containing the given words. Yet a free-text search system must return results (i.e., filled out forms), even if a query contains OOV words. The reason is that a free-text search system just interprets queries and returns filled out forms that can be submitted to a deep website. The actual searching happens at the deep site, but only after the end-user has submitted the form. Therefore, it is important that the free-text search system returns correctly filled out forms, even if the query contains OOV words.

Anything that is entered in, or assigned to, a field of a form is referred to as a *token*. A token, as defined in Chapter 3, is a textual unit that can be assigned to a field, consists of one or more words, and belongs to one *type*. A type denotes a set of tokens. We say that a free-text search system's dictionary is *complete* when it contains all tokens that can be entered in a web form. If the dictionary is complete, then by definition, any word or any combination of words that is not in the dictionary cannot be entered in the web form[1]. As a consequence, a free-text search system can safely ignore all OOV words in the query. However, if the dictionary is incomplete, then the system cannot ignore OOV words since it may be necessary to use some OOV words for filling out a form.

We illustrate this by means of an example. Consider an online bookstore for which there is a free-text search system. The bookstore's web form is shown in Figure 5.1a. The form has three *closed* fields which accept a fixed set of tokens.



(a) A complex form of an online bookstore. The `Title` field is *open*, it accepts any value. In contrast, the `Genre` field is *closed*, it only accepts a fixed set of tokens.



(b) A free-text interface to the online bookstore.

Figure 5.1: An example of how a free-text query should be interpreted to fill out a complex web form.

[1]This must be nuanced; on the one hand, a web form can inhibit end-users from submitting OOV words, either by validating the input before submitting the form, or by restricting the possible tokens that can be entered (e.g., by using menus, check boxes, or JavaScript). On the other hand, a web form can allow end-users to submit forms containing OOV words, but may return either an error or no results.

For instance, the `Max price` field only accepts numbers between 0 and 100 as tokens (anything else would result in a warning); the `Genre` field only accepts the tokens "Any", "Biography", "Novel", and "Thriller"; and, the `Language` field only accepts, say, the token "English". The form further has an *open* field, `Title`, which accepts virtually anything and is not limited to a fixed set of tokens. Regarding the free-text search system, we can populate its dictionary with tokens that act as indicators to specific fields, e.g., "title" or "titled" for the `Title` field, and "max price" or "less than" for the `Max price` field. Additionally, since we know exactly what tokens can be entered in the closed fields, we can put also those tokens in the dictionary, but we cannot do so for the open fields. Therefore, in the query "a thriller titled little big planet for less than $10" shown in Figure 5.1b, the words "a", "little", "big", "planet", "for", and "$" are OOV words. Even so, we expect some of these OOV words to be correctly considered as a token (i.e., a textual unit) and filled out as shown in Figure 5.1a.

Note that even in the extreme case that the dictionary is empty and that, as a consequence, every word is OOV, we can still tokenize the query (i.e., split the query into tokens), and assign tokens to fields. Though we would not know exactly what the tokens are, we could model what a token should look like, e.g., we could assume that a token should never consist of more than, say, 5 words. We could use the model to determine the likeliness of a token, and rank the most likely tokens that should be extracted from the query. Then, we could decide the most likely fields to which the extracted tokens should be assigned, and fill out the form. Therefore, in this chapter, we answer the following research questions:

1. What are effective methods to tokenize a free-text query, when we need to take into account that OOV words must also be used to fill out a web form?
2. What are effective probabilistic models to rank the filled out forms such that the most likely ways of filling out a single form are ranked highest?
3. Does our probabilistic approach improve on the rule-based approach of Chapter 3?

The rest of this chapter is organized as follows. In Section 5.2, we motivate that a query can be seen as a sequence of events, and discuss a possible process for generating a query. We then introduce the Hidden Markov Model (HMM), and explain how it can be applied to infer the process that generated query. Essentially, this will allow us to infer the tokens and the fields to which they should be assigned. In Section 5.3, we further explain the specific probability models of our HMM. Then, in Section 5.4, we describe our experiment to test the effectiveness of our (rule-based tokenization and probabilistic ranking) approach, and discuss our results in Section 5.5. Finally, we conclude this chapter in Section 5.6.

5.2 Goal and problem decomposition

Given a free-text query and a target web form with a set of input fields F, the goal is to find the best mapping from parts of the query to fields. Specifically,

the query should be tokenized into tokens, and each token should be mapped to a field $f \in F$. F includes a special *junk* field for tokens that are not intended as actual field values for the web form. We identify two problems. First, how to split a free-text query into tokens; second, how to map these tokens to the intended fields. On a high level, our solution is to generate all tokenizations according to a tokenization strategy. Then generate all possible mappings for each tokenization and rank these mappings based on their probability. In the following subsections we first describe an intuitive model for formulating a free-text query, and then discuss the tokenization and ranking in more detail.

5.2.1 An action model for formulating free-text queries

We now define an action model to describe the actions a user takes to construct a free-text query. This model will be used to obtain and rank possible interpretations of a free-text query. We define our action model as follows. Given a complex web form with input fields F, a user: *i*) decides which input field $f \in F$ to use; *ii*) decides what token to enter in f; and, *iii*) either decides to use an additional field upon which the process repeats itself, or decides to use no more fields, upon which the process stops. Each form field may be used at most once, and only the special *junk* field may be used multiple times.

As an example, we illustrate how a user formulated the free-text query shown in Figure 5.1b for the complex web form shown in Figure 5.1a. According to our action model, the user first chose a `junk` field and entered the token {a}. The user then chose the `Genre` field and entered the token {thriller}; followed by the `Title` field with the token {titled little big planet}; and finally, the `Max price` field with the token {for less than $10}. The tokens *titled* and *for less than $* serve as indicators and are discussed in the next section.

5.2.2 Tokenization methods

Before we explain how we process OOV words, we first describe what words are in the vocabulary, and thus, what words can be recognized in a query: *i*) *dictionary entries*, i.e., tokens found in a dictionary; *ii*) *hard separators*, i.e., a sequence of characters consisting of at least a punctuation mark followed by a white space character (the character set is denoted by the regular expression `[- \?!;,\.]`); and, *iii*) *indicators*, i.e., a hint telling the system that an adjacent piece of text should be mapped to a specific field. Indicators reside either at the start or at the end of a token and are referred to as a prefix indicator or a postfix indicator, respectively.

The free-text search system will scan a query and try to recognize everything that is in its vocabulary. Each part of the query that has not been recognized is called a *left-over*. A left-over can consist of one or more words, all of these words are OOV (otherwise the system would have recognized them). The interesting part is how to treat the left-overs. We will investigate three tokenization methods: *naive*, *rigid*, and *tolerant*. To better explain the differences between these methods, we will show how these methods tokenize the query “Boston to New

York", given that the vocabulary contains only the prefix indicator 'to'. We will use ~~striked~~ text to denote junk tokens; surrounding braces { } to denote tokens; and surrounding brackets [ ] to denote indicators.

Naive. Each left-over is considered a single token on its own.
The example query would be tokenized as: {Boston} {[to] New York}.

Rigid. Each left-over consisting of more than one word is further segmented into multiple smaller tokens.
The example query would be tokenized as in the naive method, and *in addition*: {Boston} {[to] New} {York}, {Boston} {[to] New} ~~York~~, and {Boston} {[to]} ~~New~~ {York}.

Tolerant. Like rigid, but with the addition that indicators can be placed *between* the tokens created from left-overs.
The example query would be tokenized as in the rigid method, and *in addition*: {Boston to New York}, {Boston to New} {York}, {Boston to New} ~~York~~, etcetera.

Tolerant tokenization is needed when whatever the system recognizes as an indicator is actually part of the token that should be entered in some field. For example, if someone searches for book titles and the word 'titled' serves as an indicator, then only the tolerant method can yield the right tokenization for the query: "a titled maiden", which is the actual book title. In the naive and rigid methods, the indicator [titled] acts like a token separator, thereby splitting the input into two tokens.

5.2.3 Ranking with the Hidden Markov Model

We use a Hidden Markov Model (HMM) to find the most likely mappings from tokens to fields for a given tokenization. Readers familiar with the HMM can skip to the next paragraph. A HMM is a probabilistic finite state automaton (Rabiner, 1989; Rabiner and Juang, 1993). It consists of a set of states, a set of transition probabilities from state to state, and a set of emission probabilities to model how each state produces some specific output. There are two special states: a start state and an end state. Except for the special states, each state emits one output symbol. The sequence of symbols can be observed, whereas the sequence of states cannot be observed, i.e., it is "hidden". Although the state sequence cannot be observed directly, a sequence of symbols gives some information about the hidden sequence of states. More specifically, beginning from the start state and finishing at the end state, the HMM generated the symbol sequence $O = o_1, o_2, \ldots, o_k$ by making $k + 1$ transitions from one state to the next. Each state emitted one symbol with some probability. In other words, we can compute the most likely sequence of states that produced the sequence O, if we knew the parameters of the HMM. If we compare this with our action model described in Section 5.2.1, we see that our action model can be cast as an HMM problem: find the most likely sequence of input fields chosen by the user, given the observed sequence of tokens that the user has typed.

Building the HMM

Here we explain how to setup simple variants of the HMM parameters: the set of states, transition probabilities, and emission probabilities.

1. The HMM states can be setup as follows. Each input field $f \in F$ must be represented as a state. Additionally, there must be a start state and an end state.

2. The transition probabilities of transiting from one state to the next can be learnt from manually labeled free-text queries. That is, each query is manually tokenized, and each token is manually labeled with the intended input field. We then learn the maximum likelihood estimate (MLE) $P_{MLE}(f_j|f_i)$ of transiting from input fields f_i to f_j as follows:

$$P_{MLE}(f_j|f_i) = \frac{\text{Number of times } f_i \text{ appears before } f_j}{\text{Total number of times } f_i \text{ appears}} \ .$$

3. The token emission probabilities can also be learnt from manually labeled free-text queries. For each input field, we learn the maximum likelihood estimate $P_{MLE}(o_j|f_i)$ that the field f_i emitted the j-th token in the vocabulary and that we observed this as o_j as follows:

$$P_{MLE}(o_j|f_i) = \frac{\text{Number of times observation } o_j \text{ happened at } f_i}{\text{Total number of observations at } f_i} \ .$$

Applying the HMM

Once the HMM is built, we can use it to output the most probable field sequence that could have generated a given token sequence. The learnt probabilities are smoothed beforehand, otherwise, an observation containing one token that was never seen during training will have a probability of zero. Smoothing is discussed in the next section. In any case, when applying the HMM, we must find the state sequence F which maximizes the probability of the observation $O = o_1, o_2, \ldots, o_k$, given the model β:

$$F \ = \ \underset{F' \in \mathcal{Q}}{\arg\max} \ P(O|F', \beta) \ P(F'|\beta) \ ,$$

where $\mathcal{Q}$ denotes all possible state (field) sequences that could have generated the observation (tokens) O.

5.3 Token models and smoothing

Each input field has its own set of accepted tokens. As we explained in the introduction, a dictionary may not be exhaustive and a query could contain OOV tokens, i.e., tokens not listed in the dictionary. We need methods that can assign adequate probabilities to such OOV tokens. The MLE assigns a probability of zero to an OOV token, hence, it is not adequate for our purposes. One way to obtain non-zero probabilities for such tokens is to apply Laplace smoothing:

$$P_{Laplace}(o_j|f_i) = \frac{\text{Number of times observation } o_j \text{ happened at } f_i + \alpha}{\begin{array}{c}\text{Total number of observations at } f_i \ + \\ \alpha \ * \text{ total number of different observations at } f_i\end{array}},$$

where α is usually set to one. However, this method assigns the same probability to each and every OOV token: even two tokens that look completely different from each other get the same probability. This is also not adequate for our purposes. Therefore, we will investigate several token models that could differentiate between OOV tokens.

5.3.1 Token models for discriminating OOV tokens

We investigate four different aspects that can potentially be used to discriminate between OOV tokens. We create a separate probability model for each aspect. In the next subsection, we show how these models can be combined into larger token emission models.

Word *n*-grams approximate the probability that a field emitted a particular token, based on the word sequence in that token. Words are separated by (an optional punctuation mark (;,.!?) followed by) a white space.

Character *n*-grams approximate the probability that a field emitted a particular token, based on the sequence of characters in that token.

Word-length *n*-grams approximate the probability that a field emitted a particular token, based on the sequence of word-lengths in that token. As a variation, we can also consider the length *differences* between the words in a token, which can be measured as either absolute or relative differences. We can also decide to categorize the lengths or differences into, for example: zero, small, medium, and large lengths or differences. One benefit of using (categorized) length differences instead of plain lengths, is that it reduces the sparsity in the training data.

Poisson models approximate the probability that a field emitted a particular token, based on the token's average word length, conditioned on the number of words in that token. Unlike n-gram models which usually assign higher probabilities to smaller tokens, this model assigns higher probabilities if the token's average word length is closer to the average word length of the bin the token belongs to. For example, if the expected length per word

for tokens consisting of three words is 5 characters, and our OOV token (consisting of 3 words) has an average word length of 6.7 characters, then the Poisson probability of that token is $P_{Pois}(6.7, 5) = 0.11$.

5.3.2 Smoothing by using linear interpolation

N-grams operate under the assumption that the probability of an event in a sequence of events does not depend on all previous events, but only on the last $n-1$ events. Larger values for n generally increase the prediction accuracy of n-grams. For instance, if we must predict a word in some sequence, but without information about the previous words (so $n = 1$), then the most common word would be the best answer. Yet if we knew that the previous word was "or" (so $n = 2$), then "not" may be a better answer; and if we knew that the previous two words were "better or" (so $n = 3$), then "worse" might be an even better answer. However, with larger n, n-grams suffer more from data sparseness (i.e., when many items from the vocabulary are not present in the training data). One solution is to make a linear combination of an n-gram with n-grams that have smaller n which suffer less from data sparseness. If the combination results in a probability function, then this is also referred to as *linear interpolation*. We could make a linear interpolation of a trigram ($n = 3$), with a bigram ($n = 2$) and a unigram ($n = 1$) as follows:

$$P'_3(w_n|w_{n-2}, w_{n-1}) = \gamma_1 P_1(w_n) + \gamma_2 P_2(w_n|w_{n-1}) + \gamma_3 P_3(w_n|w_{n-2}, w_{n-1}) \ ,$$

where $0 \leq \gamma_i \leq 1$, and $\sum_i \gamma_i = 1$. When all functions being interpolated use a subset of the conditioning information of the most discriminating function, this is often referred to as *deleted interpolation* (Manning and Schütze, 1999). We can use linear interpolation both as a smoothing mechanism for n-grams, as well as a method for combining different kinds of token models like:

$$P(tok) = \lambda_1 P'_{word}(tok) + \lambda_2 P'_{char}(tok) + \lambda_3 P'_{len}(tok) + \lambda_4 P_{Pois}(tok) + \lambda_5 P_{dict}(tok) \ ,$$

where $0 \leq \lambda_i \leq 1$, $\sum_i \lambda_i = 1$, and $P_{dict}(tok) = \frac{1}{|dict|}$ if $tok \in dict$, 0 otherwise. In this case, $P(tok)$ assigns a probability to a token based on (smoothed) word, character and length n-gram models, a Poisson model, and a dictionary.

5.4 Experiment

We evaluated our system in the travel planning scenario of Chapter 3. The system had to find the best query interpretation without it knowing the actual Dutch train station names. Our token models were trained on non-Dutch train station names and should compensate for the lack of Dutch station names.

5.4.1 Training data for building the HMM

The training data for the **token emission models** of the *from*, *to*, and *via* input fields, was a list of station names crawled from Wikipedia; it contained stations

from Belgium, France, Indonesia, the UK, and Germany. The training data for the **token emission models** of the *junk* field was a list of words crawled from web blogs containing the words `ik`, `trein`, and `van` (in English: I, train, and from, respectively). We scraped the pages from the web blogs and used simple heuristics to create sentences: we split the text on certain HTML tags (like
 and <p>), on a question or exclamation mark, or on a dot followed by white space. Only those sentences containing at least two station names were tokenized with the naive method from Section 5.2.2 and the left-overs constituted the training data for the token emission model of the *junk* field.

The training data for the **field transition model** was manually created, and based on the field sequences reported in Chapter 3. Those sequences did not contain any 'junk' fields, so we created and added variations containing junk fields. For example, if the transition sequence "*from-to*" appeared x times, then we added these sequences to our training data: "*junk-from-to*" $\frac{1}{2}x$, "*from-junk-to*" $\frac{1}{4}x$, and "*from-to-junk*" $\frac{1}{4}x$. This reflects our belief that it is more likely for "junk" text to appear at the start of a query.

5.4.2 Validation and test data

We used a list of Dutch train station names from Wikipedia to create queries; we randomly selected 50% of the stations for creating a validation set and the other 50% for creating a test set. For each set, we used a script to randomly generate 50 train station name pairs (from and to) and 50 train station name triples (from, to, and via) for a total of 100 different *information needs*. As each information need can be *phrased* differently, we also generated 12 different query formulations based on query templates reported in Chapter 3. As an example, say that we have the information need ("`Amsterdam`", "`Hoek van holland haven`"), and two query templates "van `from` naar `to` op `date` om `time`", and "naar `to` `from` rond `time`". We substituted the station names in the corresponding slots of each template, and we simply replaced the time and date with 13:00 and 1-1-2012, respectively resulting in the *queries*: "van `Amsterdam` naar `Hoek van holland haven` op `1-1-2012` om `13:00`", and "naar `Hoek van holland haven` `Amsterdam` rond `13:00`".

5.4.3 Method – systems without station names

The same transition model (to score the sequence of field names) was used throughout all experiments. For each tokenization method, using the validation data, we applied a simple parameter sweep to find the best linear interpolation weights for the emission models (to predict the score of the tokens – see the final equation at the end of Section 5.3.2). Each λ could (initially) take a value of 0, 0.01, 0.1, or 1, which was then normalized. For example, if we combined the word and length models, we could have $0.01P'_{word} + 0.01P'_{len}$ which was normalized to $\frac{1}{2}P'_{word} + \frac{1}{2}P'_{len}$ (the character, Poisson, and dictionary models are ignored by setting their λs to zero). As another example, $0.01P'_{word} + 0.1P'_{len}$ would be normalized to $\frac{1}{11}P'_{word} + \frac{10}{11}P'_{len}$, and so on. We selected the systems with the

highest MRR (mean reciprocal rank) in the validation data. Then we compared them with two baseline systems using the test data.

Note that the dictionary did not contain any station names. It was used to assign a constant score to the dates and times, otherwise the emission probability of those fields would be zero. Furthermore, we fixed the dictionary's weight at 0.1 so that all other components could have a lower (0.01), equal (0.1) or higher (1.0) contribution to the total token score.

5.4.4 Upper bound – systems with station names

We compared our validated systems C, with two other "baseline" systems A and B which can be considered as upper bounds. System A is the system reported in Chapter 3: it knows all station names, ignores OOV parts of the query, and uses rules to rank the query interpretations. System B is almost like A, it knows all station names, ignores OOV parts of the input, but uses probabilistic ranking (i.e., the transition probabilities discussed in Section 5.2.3). The systems in C do not know the station names and must apply the tokenization methods and emission models which were introduced in Section 5.2.2 and Section 5.3, respectively. The same transition probabilities of system B are also used in C.

5.5 Results and discussion

5.5.1 Validation results

Table 5.1 shows the emission models that yielded the highest MRR results per tokenization. It also shows the components of the emission model, their corresponding weights are shown in Table 5.2. The reported average MRR is averaged over all parameter combinations (i.e., non-zero weights for each component) of the given emission model. The total average MRR per tokenization is averaged over all parameter combinations of all emission models (not just the stated components). From these results we can see that, for example for the tolerant tokenization, it is better to use an emission model consisting of a character and a dictionary component rather than any other combination of components, since on average, these two components yield a much higher MRR. We can also see that, from naive to rigid to tolerant, the difference between the maximum and the average MRR grows. This difference is consistent across all our validation results. This can be explained by the fact that each tokenization method induces a search space. A smaller search space contains less erroneous answers, but on the other hand, it may often fail to include the right answer in the first place.

5.5.2 Test results

The test results are shown in Table 5.3. The baselines (upper bound A and upper bound B) are equipped with a dictionary containing all Dutch train station names. Since they ignore OOV words, they have almost perfect performance; the

Table 5.1: Validation results: best emission models per tokenization method. The letters stand for word (W), character (C), length (L), Poisson (P), and dictionary (D). E.g., the emission model L+P+D consists of a length n-gram, a Poisson model and a dictionary.

Tokenization method	Emission model	Max. MRR over selected emission model	Avg. MRR over selected emission model	Avg. MRR over all emission models
Naive	L+P+D	0.481	0.477	0.455
Rigid	W+C+L+D	0.703	0.678	0.584
Tolerant	C+D	0.725	0.666	0.447

Table 5.2: Weights corresponding to the emission model's components that yielded the maximum MRR in Table 5.1.

Tokenization method	Normalized weights per component				
	Word	Character	Length	Poisson	Dictionary
Naive	-	-	0.048	0.476	0.476
Rigid	0.474	0.474	0.005	-	0.047
Tolerant	-	0.500	-	-	0.500

only thing they do not know for sure is which station name should be mapped to which field. In any case, the high MRRs indicate that the ranking heuristics and the transition probabilities are well suited to the task at hand. By using the same transition probabilities and discarding the dictionary, we see the impact of tokenization and token modeling. The tolerant and rigid tokenization methods perform much better than the naive method. This can be seen in both the validation and test runs. There is almost no difference between the rigid and the tolerant methods. A closer inspection of the data showed that the validation data contained train station names that contained indicators, while no names in the test data contained indicators. This explains why the tolerant method was better than rigid method according to the validation runs, but not according to the test results. This finding reveals two things. First, it confirms what we mentioned earlier, that the added value of the tolerant method over the rigid method is

Table 5.3: Test results.

	System	**MRR**
With dictionary	Upper bound A (heuristic)	0.953
	Upper bound B (probabilistic)	0.996
Without dictionary	System C-Naive	0.533
	System C-Rigid	0.738
	System C-Tolerant	0.732

apparent when the tokens contain indicators. Second, even when tokens contain no indicators, the benefits of the rigid method over the tolerant method are small in terms of retrieval performance.

System B is significantly better than system A ($p <= 0.01$). Systems A and B are both significantly better than (all variants of) system C. Finally, both the rigid and tolerant methods are significantly better than the naive method.

5.5.3 Discussion

Generalizability

Why would someone want to use train station names from other countries as training data to model Dutch station names? Normally, they would not. We used a large list of train station names that did not contain the actual Dutch names to avoid overfitting. We may reasonably assume that if the training data looks more similar to the actual testing data, it will have a beneficial effect on retrieval performance. More importantly, the station and junk training data had sufficiently distinct characteristics that allowed the system to distinguish train stations from junk tokens. We believe that our approach is generic and that it would work well in other domains. Ideally we would train our models from query log data. Again, to prevent overfitting, we used junk tokens that were extracted from blogs instead of query logs.

Flexibility

The fact that baseline B performs even better than baseline A shows that *our approach can alleviate developers from spending effort in designing ranking heuristics* because it is capable of learning a suitable ranking function. Also, once the system is up and running it could continuously adapt its ranking function given the stream of query log data.

5.6 Conclusion

We introduced and examined three tokenization methods (naive, rigid, and tolerant) and four token models (character n-gram, word n-gram, word-length n-gram, and a Poisson model). We adopted a probabilistic approach based on a Hidden Markov Model to assign tokens to fields, and focused on correctly assigning both tokens that are in the vocabulary, as well as tokens that are out-of-vocabulary (OOV). In contrast, previous work simply neglected OOV words. The research questions of this chapter can be answered as follows.

i) What are effective methods to tokenize a free-text query, when we need to take into account that OOV words can also be filled out in a single web form? Our results show that the rigid and tolerant tokenization methods are much more effective than the naive approach. However, the tolerant approach generates more candidates than the rigid approach and is therefore slower. Since the rigid and tolerant methods show comparable retrieval effectiveness when comparing the

overall system performance, we can recommend using the rigid method which is both effective and relatively efficient.

ii) *What are effective probabilistic models to rank the filled out forms such that the most likely ways of filling out a single form are ranked highest?* Overall, the Hidden Markov Model serves as an effective framework for ranking the filled out forms. We cannot simply say what are the most effective emission models, since these depend on the choice of tokenization method. However, our results do show that we can use and combine the four token models that we introduced in this chapter.

iii) *Does our probabilistic approach improve on the rule-based approach of Chapter 3?* Yes, the probabilistic approach does improve on the rule-based approach (in both systems, the dictionary is complete).

Overall, we can conclude that our probabilistic ranking improves over the rule-based baseline when our system is equipped with a dictionary. Also, even without a dictionary, we can still correctly interpret many queries; however, we *must* process the OOV tokens in a clever way, as the results show that the naive tokenization method is far inferior the other methods.

Chapter 6

A stack decoder for structured search

"A journey of a thousand miles begins with a single step -"
– *Lao-tzu*

In this chapter, we experiment with a novel method for translating free-text queries into structured queries. In contrast to the previous chapters, the focus of this chapter is on filling out multiple web forms, on handling spelling errors, and on efficiency. Rather than computing all possible interpretations for a given free-text query, we adopt a stack decoding approach which allows us to iteratively examine the most likely partial interpretation. In addition, it allows us to discard invalid interpretations as soon as they are found. We show that a stack decoding approach can be applied to a multi-domain, multi-site per domain setting; that our boosting and discounting heuristics increase efficiency; and, that our approach outperforms a well known baseline on a segmentation and labeling task.

Parts of this chapter have been published in Tjin-Kam-Jet et al. (2013).

6.1 Introduction

In the previous chapters, we focused on filling out a single form, given a free-text query. We first introduced a rule-based approach in Chapter 3, and showed its effectiveness in a situation where the free-text search system's dictionary is complete, containing all possible tokens. Then, we introduced a probabilistic approach in Chapter 5, and showed that, even if the dictionary is incomplete, we could use a probabilistic approach to effectively guess the right answer. In this chapter, we shift our focus on the ability to fill out multiple web forms. This means that the set of possible results is now larger, since there may be several relevant web forms that must be filled out for a given query. We revisit

the query segmentation and labeling problem of Chapters 3 and 5, but now, rather than a brute-force approach, we adopt a more efficient stack decoding implementation to deal with the larger search space. In addition, the stack decoding implementation enables us to easily detect mistakes such as words that are accidentally glued together, and tolerate spelling errors up to some degree. From our earlier experiments, we noted that free-text queries sometimes contain spelling errors or concatenated words and that our rule-based implementation could not cope with such mistakes. If the free-text search system could correct such mistakes, it could lead to improved query interpretation accuracy and to a more user-friendly system. In this chapter, we answer the following research questions:

1. What is an effective query translation method to fill out multiple web forms given a free-text query?
2. How can we increase the efficiency of the query translation method?
3. Does the query translation method improve on a state-of-the-art baseline?

The rest of this chapter is organized as follows. In Section 6.2, we discuss work related to query segmentation and labeling. We formalize the problem in Section 6.3 and and describe our stack decoding framework in Section 6.4. In Section 6.5, we describe our online experiment to gain the data used for further evaluation of the system. We show and discuss our evaluation results in Section 6.6, and conclude this chapter in Section 6.7.

6.2 Related work

The problem of translating free-text queries into structured queries relates to several fields of research. We first review studies about *segmentation*, giving insight and potential solutions to the *labeling* problem; then review work about both *segmentation* and *labeling*; then briefly discuss keyword search over databases, and conclude this section.

6.2.1 Query segmentation for web IR

In web IR, correct query segmentation can substantially improve retrieval results, e.g., grouping 'new' and 'york' as 'new york' can make a big difference. Li et al. (2011) argue that supervised methods require expensive labeled data and propose an unsupervised segmentation model that can be trained on click log data. Hagen et al. (2011) show that their segmentation algorithm, which uses only raw web n-gram frequencies and Wikipedia titles, is faster than state-of-the-art techniques while having comparable segmentation accuracy. Lastly, Yu and Shi (2009) train a CRF (Conditional Random Field[1]) with tokens from a database. They first predict labels for each word in the query, then segment at each start (S-) label.

[1] See Lafferty et al. (2001).

For example, given the query *Green Mile Tom Hanks* and the predicted labels {[S-MOVIE], [R-MOVIE], [S-ACTOR], [R-ACTOR]}, it is segmented as "Green Mile" and "Tom Hanks". It is unclear why Yu and Shi used this S/R (start, rest) labeling scheme instead of the more common BIO (begin, inside, out) labeling scheme proposed by Ramshaw and Marcus (1995). Perhaps because all query words were required to appear in the database at least once, there was no use for the O-label (to indicate that a word is out-of-vocabulary).

6.2.2 Query segmentation and labeling

The example in the previous section illustrates that CRFs can both indicate segment offsets (e.g., with start/rest labels) and assign entire segments to fields (e.g., ACTOR or MOVIE). However, CRFs need a lot of expensive (manually labeled) training data. To avoid the high costs of manually labeled data, Li et al. (2009) used two data sources to train CRFs: a pool of 19K queries labeled by human annotators; and a pool of 70K queries, automatically generated by matching entries from click logs with information from a product listings database. However, the generated queries did not contain all possible labels. Still, the highest performance was obtained when combining the evidence of both sources. In contrast, Kiseleva et al. (2010) train multiple CRFs solely on click log data. But unlike manually labeled data, click log data suffers from noise and sparsity. In a follow-up study (Kiseleva et al., 2011), they did use some manual data (brand synonyms and abbreviations) and artificially expanded their training set aiming to reduce data sparsity.

Sarkas et al. (2010) propose an unsupervised approach to segment and label web queries. They train an open language model (LM) on tokens derived from a general web log, and attribute LMs on tokens from the structured data residing in tables. They score results using a generative model of the probability of choosing: a set of attributes $T.\mathcal{A}$ from table T, a set of tokens $\mathcal{AT}$ given $T.\mathcal{A}$, and a set of free tokens $\mathcal{FT}$ given the table T. Further, they decide whether a query is intended as a web keyword query, or as a structured search query.

DATAMOLD, by Borkar et al. (2001), uses nested HMMs (Hidden Markov Models[2]) to segment and label short unformatted text into structured records. They introduce a taxonomy on the symbols (words, numbers, delimiters) in order to generalize the dictionary derived from training data. They modify the Viterbi algorithm[3] to include semantic constraints, restricting it from exploring invalid paths. Since this violates the independence assumption, they re-evaluate a path when some state transition is disallowed by the constraints.

Zhang and Clark (2011) describe a framework that uses the averaged perceptron algorithm for training and a beam search algorithm (which is essentially, a stack decoder with a small stack) for decoding, and apply it to various syntactic processing tasks, like joint segmentation and POS-tagging.

[2]See Rabiner (1989); Rabiner and Juang (1993).
[3]See Forney (1973).

Our approach distinguishes itself from these approaches in many respects, to name a few: it uses a stack decoder (Bahl et al., 1990) and incorporates additional information to enable pruning, boosting and discounting; it is not purely probabilistic and works without training; and while it does not require, it can benefit from training.

6.2.3 Keyword search over relational databases

The problem of converting non-structured queries to structured queries goes back as far as 30 years, and is largely motivated from the area of structured, relational databases. The aim is to map a keyword query directly to SQL queries (e.g., a join over two or more tables), and initially, solutions were proposed based on heuristics, grammars, and graphs. Unlike web IR where the result is typically a (list of) URL(s), the result of KSORD systems ranges from a single value to a complete table with multiple columns and rows. This raises issues like deciding what unit of information to return (e.g., a table, row, or single value) (Termehchy and Winslett, 2011). Calado et al. (2004) use a Bayesian network to score and rank the SQL queries based on data populating the database, while Jayram et al. (2006) use a rule based approach.

6.2.4 Conclusion

State-of-the-art segmentation and labeling methods are based on HMMs or CRFs and consistently outperform other baseline methods on the segmentation and labeling task. These methods operate in a pipelined fashion: first do a trivial segmentation by splitting the query into words separated by whitespaces or punctuation characters, then do labeling probabilistically. However, they were designed for cases with little or no constraints (see Section 6.3) and without the need for normalizing values, whereas several constraints apply to our case and values sometimes need to be normalized. Furthermore, these methods need large amounts of (manually labeled) training data, while fully unsupervised methods suffer from noisy training data. As a general remark, there is no agreed upon test collection for comparing these methods, which makes it hard to determine the best method. That is, if such a conclusion can be made at all, since each method has been developed for very specific use cases.

6.3 Problem description and approach

Our query translation problem can be formalized as:

Given a web form and a free-text query, find the intended values and assign the values to their intended fields, under the constraints imposed by the web form.

Here, a *web form* has one or more input *fields*; it only accepts queries as *structured information needs* consisting of a set of field-value assignments, e.g., $F_1 = v_1$,

$F_2 = v_2, \ldots, F_n = v_n$. Also, it may impose *constraints*, only allowing certain combinations of fields and values. Finally, a *free-text query* is an unstructured sequence of characters that describes an intended structured information need.

Next, we describe the different types of constraints, how they can aid translating free-text queries to structured queries, and our approach.

6.3.1 Hard constraints

Web forms may restrict the combinations of allowed fields and values in several ways. Queries that do not satisfy these constraints are not accepted by the web form, such queries are *invalid*. Otherwise, they are *valid*.

Mandatory fields. A web form may require certain fields to be filled out before it can be submitted. For example, it may require either the `make` field, or both the `min` and `max` price fields to be filled out before it can be submitted. Formally, *mandatory field constraints* are propositions of the form: $(F_i) \vee (F_j \wedge F_k) \vee \ldots$, stating that at least one set of fields must be present in the query.

Conditional fields. While a field may not be mandatory, it may be required if some other field is used. For example, consider a query that contains the text *5 miles near*, which states a radius (near some place). A web form with fields `radius` and `place`, may require that if you fill out `radius`, you must also fill out `place`. Formally, *assertive conditional constraints* are implications of the form: $F_i \rightarrow F_j$, stating that if some field F_i is present in the query, then so must F_j. *Negative conditional constraints* are implications of the form: $F_i \rightarrow \neg F_j$, stating that if some field F_i is present in the query, then F_j must not also be present.

Field frequency We refer to fields that allow only one value as *single-valued* fields and to fields that allow more values as *multi-valued* fields. Formally, *frequency* constraints are implications stating that if a field is single-valued, it can be used at most once.

Types. A type defines a set of values. For example, the type `base color` defines 'red', 'green' and 'blue' as values, while `year` could define numbers between 1970 and 2015 as values. Closed types have a limited set of values, which are typically stored in a dictionary. Open types have a limitless set of values, such as the set of real numbers. These are typically modeled by regular expressions. An input field will only accept values of one specific type.

Dependencies. The values allowed for one field may depend on the value of another. For example, if a `make` field has value *Ford*, then `model` may have *Fiesta*, but not *Laguna*. Formally, *dependency constraints* are implications of the form: $F_i \wedge F_j \rightarrow f(\lambda(F_i), \lambda(F_j))$, stating that if two dependent fields F_i and F_j are used, then the function f applied on their values $\lambda(F_i)$ and

$\lambda(F_j)$ must be true. Here, f can be any function that takes two values as input and returns a boolean.

6.3.2 Soft constraints

Assuming that a given query is valid, soft constraints indicate which filled out web form is more likely.

Patterns. A pattern determines when values of some type should be assigned to a particular field. For example, consider the query *to New York from Dallas* and assume that *New York* and *Dallas* are values of the type `city`, which can be assigned to the fields: departure or destination. The system would benefit from knowing that the segments just before each value contain hints indicating the fields to which the values should be assigned.

For a given field, a pattern specifies the field-specific hints that can precede or follow values of the type expected by the field. Formally, a pattern is defined as a 4-tuple {field name, prefixes, type, postfixes}. Prefixes and postfixes denote a set of words which may be empty. For example, a pattern for the destination field could be specified as: {destination, [to], `city`, []}.

Field order. Ideally, when a query contains a hint for some field F, followed by a value V of the type expected by F, then by all means, assign V to F. In practice however, queries may just contain values, like the query *New York Dallas*. The system would benefit from knowing that a particular field order is more likely then another, e.g., that P(departure, destination) $\geq$ P(destination, departure). We make the Markov assumption and model the probability of a sequence of fields as: $P(f_1, f_2, \ldots, f_n) = \prod_{i=1}^{n} P(f_i|f_{i-2}, f_{i-1})$.

6.3.3 Approach

Our approach consists of three steps: 1) segmenting, i.e., splitting the free-text query into smaller *segments* ready to be assigned to some field—a segment is a subsequence of the characters of the free-text query; 2) labeling, i.e., assigning a segment value to its intended field; and 3) normalizing, i.e., if necessary, (slightly) rewriting the field value into a format accepted by the web form.

Segmenting. The intended fields and values in a free-text query are described by specific segments of the query. In order to find these segments, we do a left to right search for known values at each character position in the query. Known values are defined by a regular expression or are contained in a dictionary. Our dictionary is based on a Bursttrie (Heinz et al., 2002), but is modified to: 1) be tolerant to spelling errors, provided that the beginning of the search string is error free; and 2) return search completions, even if the string being completed has a spelling error.

Whenever a value is found, it is added to the segment in which it was found. This process yields a set of segments, each segment containing a list

of values, e.g., the segment 'red' can contain the values '4' (a color), and 'red hat' (an operating system name).

Labeling. A label assigned to a segment indicates one of three roles, namely that the segment contains: 1) a value v that will be assigned to some field F; 2) a field name, hinting that the value of an adjacent segment must be assigned to the stated field; or 3) no useful information for filling out the web form.

The labeling process must not only determine appropriate segment labels, it must also determine a segmentation. A segmentation denotes a list of segments such that the whole query can be reconstructed by concatenating each segment from the list. This also implies that the segments may not overlap each other. In Section 4, we discuss how we apply our stack decoder for this labeling task.

Normalizing. A field determines a format in which a value must be specified. For example, a field may require that a time be entered as `hh:mm`, i.e., two digits for the hour, a colon, and two digits for the minutes. If the query contains a time as *ten to five am*, it should be normalized to `04:50`. For normalizing dates and times, we created a separate function. Conceptually, a function extends the set of input fields of a web form, thereby extending the possible formats in which values can be specified. Other normalizations, like when the color *red* actually has a value `4`, or when a word is misspelt, are dealt with using a dictionary.

6.4 Stack decoding

Given a free-text query, we first segment it into a set of segments, each segment containing a list of values. Next, we initialize a sorted stack with an empty *path*. A path has a score and a list of labeled segments. We then iteratively decode the query as follows: 1) remove the best path from the stack; 2) look up all segments S that follow immediately after the last segment in the path; 3) for each value in each segment $s \in S$, determine the possible labels and label the segment; 4) for each labeled segment, create a new path and add it to the stack. The process iteratively extends partial paths to become complete paths. When a path is complete, it is removed from the stack and stored as a result for further processing. The decoding stops when the stack is empty, or when some stopping criterion is met (e.g., some max decoding time t has elapsed).

6.4.1 Scoring

A path's score is based on the field values, and on the field order which was discussed in Section 6.3.2. The score of a value v from some closed type C is initially modeled as a uniform probability of $\frac{1}{|C|}$ for observing v. The score of a numeric value from an open type is determined heuristically: based on the

number of digits, it diminishes quadratically such that a 4-digit value gets the highest score, then 3-digit and 5-digit values, and so on. An important issue in stack decoders is the comparability of partial paths (Bahl et al., 1990; Zhang and Clark, 2011). We lower a partial path's score by the number of characters that must yet be processed. This basically estimates for any partial path what the score would be if the whole query was processed. Note that lowering the score too much causes the decoder to proceed in a depth-first search manner instead of best-first search manner.

6.4.2 Pruning

With enough time and memory resources, we could theoretically examine all possible paths, including invalid ones. In practice however, we have little time and resources and need to reduce the time spent on processing invalid paths. Therefore, we prune partial paths that violate the dependency, field frequency, or negative conditional constraints defined in Section 6.3.1. Pruning such partial paths will not prune possible valid complete paths from our search space.

6.4.3 Boosting and discounting

The speed of a stack decoder depends on whether it repeatedly chooses and expands the best partial path, until it reaches the most likely complete path. The choice is based on fields and values seen so far, without regard for possible further fields and values. This is not always desirable. For example, consider the query *BMW 2000 euro* and a form with three fields: `make`, `year` and `price`. The segment 'BMW' is labeled as `make` and we must now label the segment '2000'. If we only considered segments up to and including '2000', then both labels `year` and `price` would seem fine. However, if we would have looked ahead when labeling '2000' as `year`, we would have known that this label is not likely, therefore we would have lowered the position of this path in the stack. The process of looking ahead and deciding to raise or lower a path's position in the stack is referred to as *boosting* or *discounting*, respectively. We can rank the complete paths by their original scores or by the boosted and discounted scores.

6.5 Data used for evaluation

Our aim was to obtain realistic queries under the following three conditions:

1. **Multi-domain search environment.** Participants should ask queries in an environment where they get real-time query suggestions and where they can query about different domains, like travel planning or second hand cars.

2. **Multi-site domains.** Each domain should have different sites that may or may not offer the same search functionality. For example, in travel

planning, one site might offer bus travel results, while other sites offer train or flight results.

3. **Minimal query bias.** Participants should not be persuaded to any kind of information need nor to any structure in which the they can phrase their query.

6.5.1 Data acquisition

We setup an online search system covering 3 multi-site domains, and instructed the participants that they could search these domains. Next, we briefly describe the domains, the instructions for the participants, and how we obtained free-text queries from the participants.

1. **Travel planning.** This domain has 3 sites, each providing either bus, train, or flight travel information. Instruction: *Find travel advice (for example, a train trip to someone you know) and rate the result.*

2. **Second hand cars.** This domain has 5 sites, each having a web form with fields for at least minimum price, maximum price, make, and model. Instruction: *Find cars with specific characteristics (for example, find cars with characteristics like your own car or a car of someone you know) and rate the result.*

3. **Currency exchange.** This domain has 3 sites, each having a web form with three input fields (from currency, to currency, and amount). Instruction: *Find the exchange rate (of currencies of your choice) and rate the result.*

Participants started with a training session in which they could issue multiple queries in each of the three domains. Whenever a result was clicked on, a box appeared asking to rate the result as either: 'completely wrong', 'iffy', or 'completely right'. After rating a result, the system prompted for the next domain. It is natural to rephrase the query if a system returns no or unsatisfying results. However, if a participant believed that the query could have been answered correctly by the system, he/she could indicate this and optionally describe what kind of results should have been returned. During the training session, participants got acquainted with the system and discovered the search functionality by themselves. After introducing all domains, the participant was asked to conduct 10 different searches and rate at least one result of each search request. As an incentive to continue with the experiment, a score was shown based on, amongst others: the number of queries issued, the number of results rated, and the search functionality[4] discovered so far. Participants could quit whenever they wanted.

[4]Search functionality here means the number of different fields in all clicked results, divided by the total number of fields from all web forms configured in the system.

6.5.2 Manual analysis and labeling

We manually analyzed all submitted queries and specified which forms could return relevant results and how the forms should be filled out. For each form, we compiled a test corpus specifying the set of field-value assignments for the queries that make sense to the web form. We then measured how much our judgments agreed with those of the participants using the *overlap* between our manually assigned query-result pairs and those of the participants. Overlap is defined as the size of the intersection of the sets of relevant results divided by the size of their union, and has been used by several studies for quantifying the agreement among different annotators (Voorhees, 2000; Hiemstra and van Leeuwen, 2002; Demeester et al., 2012). We needed to compile the test corpora ourselves because: first, participants did not (and were not expected to) find and label all correct results. Second, the system may not have returned any correct results, making it impossible for participants to label all correct results.

6.5.3 Data obtained

In total, 47 participants interacted with the system and 23 participants opted to enter their age and gender information, resulting in 17 males (age ranging from 19 to 81, avg. 39) and 6 females (age ranging from 25 to 41, avg. 30). We analyzed 363 queries, nearly half of which were invalid: either missing mandatory fields or asking information that was out of scope. Examples of invalid queries are: *to Amsterdam*; *how long is the Golden Gate bridge*; *kg to pound*; and, *for sale: 15 year old Mercedes*. In total, we labeled 194 valid queries containing enough information so that we could fill out one of the web forms from our experiment. When multiple web forms could be filled out for a given query, we chose the ones in which we could specify most or all key-value pairs of the query. Our manual labeling results for the travel planning, currency exchange, and second hand cars domains are shown in Table 6.1. The rows 'A' to 'K' each correspond to a web form in the specified domain and contain: *Queries:* the number of queries submitted in that form; *Max:* the maximum number of different ways to fill out that form for a single query; *Avg:* the average number of filled out forms per query; and, *Std.dev:* the standard deviation from this average. The row 'All' shows the results when aggregating all forms into one, and should be interpreted as: 194 queries were submitted in this aggregated form; there was a query that could be filled out in 19 different ways; there were 2.99 differently filled out forms per query on average, with a standard deviation of 2.43.

A result (i.e., a filled out form) denotes a set of field-value pairs. On a result level, the agreement of our judgments and those of the participants is 0.33, which is consistent with the "key" agreement reported in (Demeester et al., 2012). Though it might seem low, it is a direct result of the strict comparison: one slightly different field value causes results to disagree completely. If we considered field-value pairs instead, and averaged the field-value agreement per result, the agreement is 0.68.

Table 6.1: Manual labeling results.

Travel	Queries	Max.	Avg.	Std.dev.
A	52	8	1.19	0.99
B	5	5	1.80	1.79
C	12	3	1.25	0.62
Currency				
D	61	1	1.00	0.00
E	61	2	1.03	0.18
F	62	1	1.00	0.00
Cars				
G	24	2	1.04	0.20
H	59	7	1.39	1.16
I	61	9	1.38	1.29
J	52	4	1.12	0.51
K	49	3	1.20	0.58
Merged				
All	194	19	2.99	2.43

6.6 Evaluating the stack decoder

We evaluated our system using the data described in Section 6.5.3. We investigated how different stopping criteria, boosting, discounting, and ranking on original or on boosted scores, affected the decoding time and retrieval performance. The performance was measured using MAP (Mean Average Precision[5]). Table 6.2 lists the 6 stopping criteria that we used. Each row states that the decoding should stop whenever: a maximum of r results was found; or, more than t time elapsed during decoding; or, the next result's score was lower than some absolute minimum *abs.min*; or, when it was lower than some minimum *rel.min* relative to the best result. Further, we tested two settings for pruning probably irrelevant paths based on the percentage of the query that was ignored. A path was discarded if more than j% was ignored (e.g., due to unknown words). One (fairly strict) setting required the system to interpret at least 60% of the query, while the other required only 20%.

6.6.1 Individual, "per form" evaluation

One at a time, we loaded a form's dictionary and constraints and ran its tests. We first ran the tests without training our system, i.e., we used a uniform field order distribution[6]. Table 6.3(a) shows the averaged results of the individual tests, weighted by the number queries per form. Then, we conducted a 5-fold

[5] See Voorhees (2000).

[6] Except in one form where we manually specified that "departure" fields were more likely followed by "destination" fields, instead of other fields. However, this was done before going online and gathering data, so before we had even seen the test data.

Table 6.2: Stopping criteria, sorted by number of results and "strictness".

	Results	Time	Absolute minimum	Relative minimum	Ignore%
A	10	0.5	-200	-150	40
B	10	0.5	-200	-150	80
C	10	0.5	-600	-550	80
D	50	45	-200	-150	40
E	50	45	-200	-150	80
F	50	45	-600	-550	80

cross validation and trained our system on the field order distribution, or field transitions, from the labeled queries. These results are shown in Table 6.3(b).

Table legend

In Table 6.3, the leftmost letters B, D, and R denote *boosting*, *discounting*, and *ranking* by original score, respectively. *Time* is the average query decoding time. *MAP1* and *MAP2* are the MAP of filled out forms, and of segmentation & labeling, respectively. The headers A–F denote stopping criteria (see Table 6.2). The seven MAP1 values under each header are represented in a color gradient: red denotes the lowest scores, yellow denotes moderate scores, and blue denotes the highest scores. Per table, the values for MAP1, time, and MAP2, are further accompanied by green, red, and blue bars, respectively. The length of a bar represents the value in a cell, longer bars correspond to higher values.

Results

In both the untrained and the trained systems, when we compare the results under headers A and B, and those under D and E, the results under A are consistently lower than those under B; and the results under D are consistently lower than those under E. Criteria A differs from B, just like D differs from E, only in terms of the percentage of OOV words that determines when a query should be discarded. The results show that paths in which 40% to 80% of the query is OOV should not be pruned.

When we compare the results of the trained system with those of the untrained system, we see that training increases the MAP performance and that it decreases the processing time. We also see that the best performance is obtained by ranking on the original scores when the system is not trained, but that we should rank on the boosted scores when the system is trained.

Regarding our boosting and discounting heuristics, we see that they do affect the system's performance, especially with relatively strict stopping criteria; and that as the criteria relaxes, the effect decreases. This is due to the relatively small search space in the individual tests. The stopping criteria limit the part of the search space that can be inspected, while the boosting and discounting try to sneak in as many relevant paths to this limited space as possible. Thus

when the stopping criteria are sufficiently relaxed, the effects of boosting and discounting will naturally decrease. Furthermore, both boosting and discounting

Table 6.3: Individual test results, averaged over individual forms.

(a) Averaged results of the untrained system.

			A			B			C		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.485	0.05	0.549	0.551	0.04	0.608	0.622	0.07	0.625
0	1	0	0.502	0.04	0.576	0.568	0.04	0.636	0.641	0.07	0.656
1	0	0	0.503	0.04	0.567	0.569	0.04	0.627	0.639	0.07	0.641
1	1	0	0.504	0.04	0.579	0.570	0.04	0.638	0.640	0.07	0.652
0	1	1	0.519	0.04	0.583	0.582	0.04	0.644	0.642	0.07	0.664
1	0	1	0.521	0.04	0.580	0.583	0.04	0.642	0.642	0.07	0.656
1	1	1	0.522	0.04	0.585	0.584	0.04	0.646	0.643	0.07	0.659

			D			E			F		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.485	0.05	0.548	0.551	0.05	0.608	0.629	0.31	0.627
0	1	0	0.501	0.05	0.575	0.568	0.05	0.635	0.647	0.30	0.653
1	0	0	0.503	0.05	0.566	0.569	0.05	0.626	0.647	0.16	0.645
1	1	0	0.504	0.04	0.577	0.570	0.04	0.637	0.649	0.15	0.656
0	1	1	0.521	0.05	0.586	0.583	0.04	0.647	0.649	0.31	0.666
1	0	1	0.522	0.05	0.583	0.585	0.05	0.645	0.649	0.16	0.664
1	1	1	0.523	0.04	0.588	0.586	0.04	0.649	0.650	0.16	0.668

(b) Averaged 5-fold cross-validated results of the trained system.

			A			B			C		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.530	0.03	0.613	0.607	0.03	0.680	0.687	0.05	0.696
0	1	0	0.552	0.03	0.636	0.629	0.03	0.703	0.709	0.05	0.718
1	0	0	0.553	0.03	0.635	0.629	0.03	0.701	0.711	0.05	0.717
1	1	0	0.553	0.03	0.637	0.630	0.03	0.704	0.711	0.05	0.716
0	1	1	0.537	0.03	0.637	0.616	0.03	0.703	0.698	0.05	0.718
1	0	1	0.538	0.03	0.635	0.616	0.03	0.702	0.701	0.05	0.717
1	1	1	0.538	0.03	0.637	0.617	0.03	0.704	0.701	0.05	0.717

			D			E			F		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.527	0.03	0.610	0.605	0.03	0.677	0.685	0.14	0.695
0	1	0	0.548	0.03	0.633	0.627	0.03	0.700	0.708	0.15	0.716
1	0	0	0.550	0.04	0.632	0.627	0.04	0.698	0.708	0.09	0.716
1	1	0	0.551	0.03	0.634	0.628	0.03	0.700	0.709	0.09	0.716
0	1	1	0.534	0.03	0.633	0.613	0.03	0.700	0.692	0.14	0.716
1	0	1	0.535	0.04	0.632	0.613	0.03	0.698	0.693	0.09	0.716
1	1	1	0.536	0.03	0.634	0.614	0.03	0.700	0.693	0.09	0.716

lead to higher MAP without increasing the processing time. For now, we can say that the improvements are statistically significant, but we will give an overview of the significance tests in Section 6.6.3.

Conclusions

From the individual tests, we can conclude that training, boosting, and discounting each increase MAP and reduce decoding time. Applying both heuristics is better than applying just one; and to achieve the best retrieval performance, we should apply both heuristics, train the system, and rank on boosted scores.

6.6.2 Collective, "aggregated forms" evaluation

We collectively loaded all forms into our system. This causes the search space to be much larger, and aside from determining how to fill out a form, the system must also determine which forms to return in the first place. We aggregated the tests and specified all forms that should be returned and all ways of filling out a form for a given query. Like in the individual tests, we first ran the tests without training the system, these results are shown in Table 6.4(a). Then, we trained our system and conducted a stratified, 5-fold cross-validation of which the results are shown in Table 6.4(b).

Table legend

The structure of Table 6.4 is similar to that of Table 6.3, and is described on page 88.

Results

Just like in the individual tests, the results of the collective tests show that we should not prune paths in which 40% to 80% of the query is OOV; and that training, boosting, and discounting each lead to statistically significant improvements both in terms of increasing MAP as well lowering decoding time. However, there are three notable differences between the collective tests and the individual tests. First, the MAP1 in the collective tests is much lower than in the individual tests. This could be because we increased the number of forms, but not the number of results to return. According to Table 6.1, the average number of ways to fill out a single form for a given query is close to one. Therefore, on average, there is just one path that leads to the correct result. However, other paths may be ranked higher than the correct path, so if the decoding stops because the specified number of paths (e.g., 10) have been decoded, then it is possible that the correct path is not among the retrieved paths. By increasing the number of forms, we also increase the number of incorrect paths that compete for the top-N positions, which may lower the odds of having the correct path among the top-N paths. While we did not specifically test for this hypothesis, there is evidence suggesting that we should also increase the number of results if we increase the number of forms. Consider the results of the trained systems from the individual tests in Table 6.3(b), and from the collective tests in Table 6.4(b), under criteria A and D. The decoding times under both criteria is always less than 0.5 seconds; therefore, the *practical* difference between A and D is only the number of results to return: 10 results in A, and 50 results in D. Under A, the average MAP1 in

Table 6.4: Collective tests results over the aggregated forms.

(a) Evaluation results of the untrained system.

			A			B			C		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.414	0.09	0.523	0.444	0.08	0.547	0.442	0.23	0.539
0	1	0	0.424	0.08	0.539	0.454	0.08	0.562	0.453	0.22	0.554
1	0	0	0.427	0.08	0.539	0.463	0.08	0.570	0.461	0.22	0.553
1	1	0	0.428	0.08	0.545	0.464	0.08	0.576	0.464	0.22	0.558
0	1	1	0.402	0.08	0.509	0.434	0.08	0.540	0.439	0.22	0.540
1	0	1	0.407	0.08	0.511	0.440	0.08	0.548	0.447	0.22	0.538
1	1	1	0.408	0.08	0.514	0.441	0.08	0.550	0.449	0.22	0.537

			D			E			F		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.446	0.11	0.556	0.475	0.10	0.581	0.504	1.64	0.597
0	1	0	0.455	0.11	0.571	0.484	0.10	0.596	0.516	1.66	0.611
1	0	0	0.462	0.14	0.573	0.495	0.13	0.603	0.514	1.09	0.608
1	1	0	0.461	0.13	0.576	0.493	0.12	0.606	0.517	1.16	0.613
0	1	1	0.417	0.10	0.527	0.452	0.10	0.558	0.479	1.88	0.579
1	0	1	0.424	0.13	0.533	0.457	0.12	0.567	0.477	1.23	0.579
1	1	1	0.422	0.12	0.532	0.456	0.12	0.567	0.478	1.25	0.582

(b) 5-fold, stratified cross-validation results of the trained system.

			A			B			C		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.445	0.09	0.560	0.494	0.07	0.597	0.486	0.21	0.574
0	1	0	0.463	0.08	0.585	0.510	0.07	0.620	0.506	0.21	0.597
1	0	0	0.465	0.07	0.580	0.517	0.07	0.620	0.512	0.20	0.586
1	1	0	0.465	0.07	0.588	0.517	0.07	0.629	0.513	0.20	0.592
0	1	1	0.456	0.07	0.589	0.509	0.07	0.626	0.501	0.21	0.598
1	0	1	0.458	0.07	0.582	0.512	0.07	0.623	0.510	0.20	0.589
1	1	1	0.458	0.07	0.591	0.513	0.07	0.634	0.512	0.20	0.596

			D			E			F		
B	D	R	MAP1	Time	MAP2	MAP1	Time	MAP2	MAP1	Time	MAP2
-	-	-	0.484	0.11	0.587	0.530	0.09	0.625	0.562	1.21	0.631
0	1	0	0.498	0.10	0.607	0.545	0.09	0.645	0.581	1.20	0.648
1	0	0	0.504	0.12	0.607	0.552	0.10	0.647	0.579	0.91	0.648
1	1	0	0.502	0.11	0.609	0.550	0.10	0.650	0.582	0.90	0.652
0	1	1	0.482	0.10	0.608	0.537	0.09	0.649	0.569	1.19	0.652
1	0	1	0.487	0.12	0.608	0.541	0.10	0.649	0.568	0.89	0.650
1	1	1	0.485	0.11	0.611	0.539	0.10	0.653	0.568	0.90	0.655

the individual tests is 0.54, while in the collective results it is 0.46. Under D however, the average MAP1 in the individual tests is still 0.54, while in the collective tests it increases to 0.49. This suggests that we should also increase the number of results to return if we increase the number of forms, at least, if we wish to obtain comparable MAPs. Second, both boosting and discounting increase MAP and decrease decoding time in the individual tests. However, in the aggregated

tests, this is only the case with the trained system (with a slight exception under criteria D). Note that this is also reflected in Table 6.5: boosting or discounting alone do not decrease decoding time if the system is not trained, but they do if the system is trained. With the untrained system, the heuristics actually reduce MAP if we do not also rank on the boosted scores (i.e., R must be zero). Third, regardless of whether or not the system is trained, we should always apply boosting and rank on the boosted scores, as these settings consistently yield the highest MAP.

Conclusions

From the collective tests, we can conclude that if we increase the number of forms, then we should also increase the number of results that should be returned if we want to obtain comparable MAP. To achieve the best retrieval performance, we should train the system, apply our boosting and discounting heuristics, and rank on boosted scores. Finally, we can conclude that our system can effectively act as a broker over different sites across different domains (e.g., travel planning, currency, second hand cars).

6.6.3 Efficiency

We used the sign test to see whether training, boosting, discounting, or ranking on original scores have a statistically significant effect with $p \leq 0.05$. Table 6.5 on page 93 shows the results of the 15 comparisons we have performed. Since we have multiple significance tests, we applied the Bonferroni method (Bland and Altman, 1995) and used an adjusted p-value of 0.003 per test. Table 6.5 shows that boosting, discounting, as well as boosting and discounting, improve over the basic stack decoder (i.e., the decoder without any heuristic). This holds both when the system is trained and when it is not trained. The table also shows that training improves over the untrained decoder, and that training and boosting and discounting improves over training. Lastly, compared to the untrained basic stack decoder, the combination of training, boosting, and discounting increases MAP by 12% and decreases decoding time by 28%, on average.

So far, the results show that it is best to train the system and apply the heuristics as this always increases MAP and lowers decoding time. However, if we look at Tables 6.3 and 6.4, we see that the stopping criteria also influence the MAP and the decoding time. Without the stopping criteria, the stack decoder would decode until its stack is empty, and potentially examine hundreds of millions of query interpretations, many of which would be highly unlikely. So, on the one hand, the stopping criteria are meant to keep the decoder from examining highly unlikely interpretations. On the other, they can speed up the stack decoder, albeit at the cost of reducing the MAP. If we plot the average MAP1 per criteria against the average query decoding time per criteria, as in Figure 6.1, we can indeed see that we can speed up the decoder, but at the cost of losing MAP. So, in a way, we can say that the best criteria depends on how much time you can afford.

Table 6.5: Comparisons for statistical significance ($p \leq 0.05$), testing whether certain combinations of training (T), boosting (B), discounting (D), and ranking by original score (R) improves over another combination. A statistically significant increase in MAP is indicated with ▲, while a significant decrease in decoding time is indicated with ▼.

			Individual		Aggregated	
TBDR	>	TBDR	MAP	Time	MAP	Time
– 0 1 0		– – – –	▲	▼	▲	
– 1 0 0		– – – –	▲	▼	▲	
– 1 1 0		– – – –	▲	▼	▲	▼
– 1 1 0		– 1 0 0	▲			▼
– 1 1 0		– 0 1 0				
– 1 1 1		– 1 0 1	▲		▲	
– 1 1 1		– 0 1 1	▲			
1 – – –		– – – –	▲	▼	▲	
1 – – –		– 1 1 1	▲	▼	▲	
1 0 1 0		1 – – –	▲	▼	▲	▼
1 1 0 0		1 – – –	▲	▼	▲	▼
1 1 1 0		1 – – –	▲	▼	▲	▼
1 1 1 0		1 0 1 0	▲			▼
1 1 1 0		1 1 0 0				▼
1 1 1 0		1 1 1 1	▲			

6.6.4 Baseline evaluation

To our knowledge, no other system translates free-text queries to filled out forms while at the same time normalizing values and checking against constraints. However, LingPipe[7] is a suitable baseline, as it recognizes named entities by segmenting & labeling text, and is a widely used text processing toolkit. We manually segmented the queries and labeled each segment. Filled out forms naturally correlate with segmented & labeled queries. However, due to normalization and constraint checking, there may not be a valid filled out form even if the query is correctly segmented.

We evaluated both systems on their prediction of which query segments contained field values and what label to assign to each segment. We used 3 data sets to simulate "untrained" up to "fully trained" systems: *set A* contains uniform field transitions and uniform token counts; *set B* contains field transitions from the queries, but uniform token counts; and, *set C* contains both field transitions and token counts taken from the queries. We cross-validated LingPipe using out-of-the-box settings for named entity recognition. In each test, we loaded the dictionary but no regular expressions because they cannot be used together (at least, not out-of-the-box). We cross-validated our system using the parameters from Table 6.3(a) and from Table 6.4(a) that gave the best filled out forms when returning 10 results (i.e., with the highest *MAP1*, and lowest time if *MAP1* is

[7] Alias-i. 2013. LingPipe 4.1.0. http://alias-i.com/lingpipe (accessed March 1, 2013)

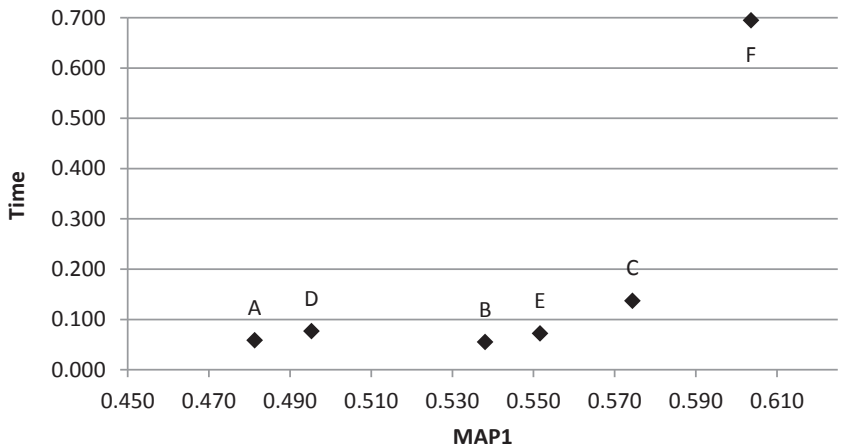


Figure 6.1: Average MAP1 and average query decoding time per criteria.

equal). So, for the individual tests we used {criteria=C; B,D,R=1,1,1}, and for the collective tests {criteria=B; B,D,R=1,1,0}.

The segmentation & labeling results are shown in Table 6.6. Row A denotes results of untrained systems (i.e., they are only "trained" on uniform distributions). Rows B and C denote 5-fold cross validation results of the systems. The collective cross-validations tests are stratified, i.e., 1/5-th of the queries of each form is used in each fold. As expected with no training (row A), LingPipe performs poorly, which contrasts with our untrained system. For now, our system can only train on field transitions (row B), and this already improves performance. Training LingPipe on only field transitions also improves performance; but training on both transitions and token counts (for which it was designed) gives the biggest improvement. Since LingPipe does not know that once it uses labels of one form it cannot use labels of others, it performs very poorly in the collective tests. Then again, it was not developed for such a task. Finally, in almost all cases where our system is better than LingPipe, the difference is statistically significant ($p \leq 0.05$). Only in the individual tests, when we compare our trained system (row B) against a trained LingPipe system (row C), the difference is not statistically significant. However, we have something in reserve and we believe that training our system on training set C could lead to a significant improvement over the LingPipe system trained on training set C.

6.6.5 Further discussion

After inspecting a sample of the results we noted that OOV (out-of-vocabulary) words and spelling errors were lowering retrieval performance. While some OOV words can easily be added (e.g., new car models), others constitute natural language phrases that should be interpreted in context and cannot easily be added.

Table 6.6: Segmentation & labeling results. Training set A involves no training. In B we train on field transitions, and in C on both field transitions and token counts.

(a) Results averaged over the individual tests per form.

LingPipe			Our system		
Training set	MAP	Time	Training set	MAP	Time
A	0.302	0.27	A	0.659	0.07
B	0.459	0.04	B	0.717	0.05
C	0.708	0.04	-	-	-

(b) Results of aggregating the web forms.

LingPipe			Our system		
Training set	MAP	Time	Training set	MAP	Time
A	0.117	66.88	A	0.576	0.08
B	0.207	5.16	B	0.629	0.07
C	0.289	5.11	-	-	-

Further research must be done in coping with these OOV words, and an online learning approach using click log data is potentially the cheapest solution. In Section 6.5.2 we mentioned two challenges related to using click log data for learning what kind of mistakes are made by the system. We noticed that only few labels occurred for numerical tokens, e.g., a number was often intended as a price, but never as the engine displacement. This makes it easier for LingPipe to guess the right label, as it is ignorant of the actual possible labels for each numerical token and just considers the labels seen during training.

6.7 Conclusion

We introduced a novel and flexible method for translating free-text queries to structured queries for filling out web forms. This enables users to search structured content using free-text queries. In contrast, web search engines struggle to index structured content from web databases, and users cannot enter structured queries in a typical web search engine. Our method consists of three steps: segmenting, labeling, and normalizing. We use the constraints imposed by web forms to prune the search space and apply boosting & discounting heuristics to further increase efficiency. Our results confirm that our heuristics are effective, reducing decoding time and raising retrieval performance. We also showed that without training, our system outperforms an untrained baseline on the individual and the collective tests. Compared to a trained baseline, our trained system showed better results on the individual tests, but the difference was not statistically significant. However, on the collective tests, our system outperformed the baseline.

The research questions of this chapter can be answered as follows.

i) *What is an effective query translation method to fill out multiple web forms given a free-text query?* Stack decoding is an effective method to fill out multiple web forms.

ii) *How can we increase the efficiency of the query translation method?* We have shown that the stopping criteria influence both the decoding time as well as the MAP, and that the best criteria depends on how much time one is willing to spend per query. Furthermore, regardless of what stopping criteria is used, the efficiency can always be increased by training, and by applying our boosting and discounting heuristics. Iteratively inspecting the most likely path and pruning invalid paths as soon as they are found are necessary to increase efficiency. We introduced boosting and discounting heuristics to aid the system in determining the most likely path. Our results show that these heuristics are effective, as they increase the retrieval performance and decrease the decoding time.

iii) *Does the query translation method improve on a state-of-the-art baseline?* To our knowledge, no other system translates free-text queries to filled out forms while at the same time normalizing values and checking against constraints. We therefore used LingPipe, a state-of-the-art baseline, to compare the segmentation and labeling performance, which correlates with the performance of filling out forms. We compared our system against the baseline in several scenarios, and our system was always better than, or at least as good as, the baseline.

Overall, we can conclude that our stack decoding approach is both effective and efficient in terms of selecting relevant web forms and adequately filling out the selected forms for a given free-text query. Our system works best when it is trained and when both boosting and discounting heuristics are applied.

Chapter 7

Conclusion

"The future is here. It's just not widely distributed yet -"
– *William Gibson*

In this thesis, we investigated some of the main challenges for creating a deep web search engine in which it is possible to search across multiple sites of different domains, while maintaining the ability to enter structured queries. A structured query refers to a query in which end-users can specify restrictions on one or more attributes of an item; it can be entered via a multi-field interface such as a web form with multiple input fields. In essence, a structured query consists of pairs of attributes and values. Rather than specifying pairs of attributes and values via a multi-field interface, we investigated the possibility of using a single-field, free-text interface in which end-users can enter a free-text query, which is a textual description of a structured query. Particularly, we investigated some technical aspects of deriving the intended structured query from a free-text query, and some aspects of how end-users tend to interact with a free-text interface. In this final chapter, we conclude this thesis by summarizing the research questions introduced in Chapter 1, by discussing the limitations of our approach, and by suggesting directions for future work.

7.1 Research questions revisited

In Chapter 1, we motivated the need for a deep web search engine to translate free-text queries into structured queries. Adequate query translation is paramount to a successful deployment of our proposed approach of distributed deep web search. The first set of research questions in Chapter 1 focuses on investigating the effectiveness of several approaches to the translation of free-text queries into structured queries. The second set of research questions focuses on studying the interaction of end-users with a prototype system based on these approaches. In the following two subsections, we discuss both sets of research questions, and end each subsection with the conclusions reached.

7.1.1 Translating free-text queries to structured queries

RQ1: What is an effective approach to translate free-text queries into structured queries, under each of the following constraints where the free-text search system:

a) fully knows what values can be entered in a single form and how these values are typically used in a free-text query

b) partially knows what values can be entered in a single form and how these values are typically used in a free-text query?

c) fully knows what values can be entered in multiple forms and how these values are typically used in a free-text query?

For the first constraint, when all values that can be entered in a single form are known, we showed that a rule-based query translation approach is effective. In an online experiment to test the rule-based approach, we obtained over 30,000 queries from nearly 12,000 end-users. We manually analyzed a sample of 1,500 queries and showed that the rule-based approach achieved an accuracy of 0.927 (see Chapter 4). For the second constraint, when not all values that can be entered in a single form are known, we showed that a probabilistic approach of first generating candidate structured queries, and then using a probabilistic model to rank the candidate queries, can be effective. In the extreme case where each query contained at least two tokens that were unknown to the system, the probabilistic approach achieved a MRR (mean reciprocal rank) of 0.738. In the other extreme where all tokens were known, it achieved a MRR of 0.996, whereas the rule-based approach achieved a MRR of 0.953 on the same dataset (see Chapter 5). Finally, for the third constraint, when all values that can be entered in multiple forms are known, we showed that a stack decoding approach can effectively translate free-text queries to structured queries, achieving a MAP (mean average precision) of over 0.55. Note that MAP and MRR are correlated, and although we did not report the MRR for the experiments in Chapter 6, we can say that, on average, the MRR was 1% higher than the MAP in the individual experiments and 16% higher than the MAP in the collective experiments. MRR only considers the first relevant result that is retrieved, while MAP considers all relevant results that are retrieved. Moreover, if there is exactly one relevant result per query, then MAP is equal to MRR. In the individual experiments, the average number of relevant results per query that can be retrieved is close to one, while in the collective experiments, it is close to three.

These measurements should be put into perspective. For the first constraint, we evaluated the rule-based approach on a web form with six input fields, in a relatively simple travel-planning domain. Our preliminary tests, and the laboratory experiment (see Chapter 3), allowed us to fine-tune the patterns (i.e., the translation rules, the cues or words that hinted at some field) and the vocabulary (i.e., the tokens and their synonyms that could be entered in the form) before setting up the public demo. Compared to formulating a query about the desired brand, model, and fuel capacity of a car, it is relatively simple to formulate a

query about how to go from one place to another. This relative simplicity, and the fact that we could fine-tune the system, could explain the high performance measures. For the second constraint, we evaluated the probabilistic approach on a synthetic dataset for the same web form with six fields. The aim was to measure what performance could be achieved if the system could detect all values except the station names. While the lack of station names makes it more difficult to correctly interpret a query, the underlying web form is still relatively simple. For the third constraint, we evaluated the stack-decoding approach with multiple web forms, most containing more than six input fields. Not only were there many more vocabulary terms to consider, the system also considered several spelling variations and considered whether or not terms were concatenated (which was not done in the first system), as one of our earlier experiments showed that end-users tend to make such mistakes. Moreover, we did not spend the same amount of fine-tuning as we did in the first experiment. The higher complexity and the lack of fine-tuning could explain the lower performance measures in this experiment, compared to those in the previous experiments. We also showed that our stack-decoding approach performed at least as good as LingPipe, which is a state-of-the-art tagger. The fact that another state-of-the-art system could at best perform as good as ours, is indicative for both the higher complexity in the scenario with multiple web forms, and the effectiveness of our approach.

RQ2: What is the trade-off between efficiency and effectiveness in translating a free-text query into a structured query?

In Chapter 6, we described a stack decoding approach to iteratively examine the best partial interpretation (and translation) of a free-text query. Unlike our rule-based approach (see Chapter 3) and our probabilistic approach (see Chapter 5), the stack decoding approach examines only a small fraction of all possible interpretations, which requires less processing time and is therefore potentially more efficient. However, because the approach only examines a fraction of all possibilities, it may fail to find the correct interpretation, resulting in a lower effectiveness. By adjusting the stopping criteria, we can influence the fraction that is examined, and hence also influence the total processing time and the retrieval performance. Our results show that there is a trade-off between efficiency and effectiveness, and that this trade-off is not linear. If we sort all stopping criteria by their average MAP in descending order, and compare the first criteria against the second, we see that the MAP drops 5% while the decoding time drops 80%. If we compare the second against the third, the MAP drops 4% while the time drops 47%; and, if we compare the third against the fourth, the MAP drops 2% while the time drops 24%. In other words, the trade-off between effectiveness and efficiency is not linear. Furthermore, we have shown that (in a given stopping criteria) training, boosting and discounting each increase MAP and decrease decoding time (i.e., there is a win-win situation). It is best to apply training, boosting and discounting together since this increases MAP by 12% and decreases decoding time by 28%.

Conclusions

From the answers to research questions RQ1 and RQ2, we can conclude that:

1. With an adequate set of translation rules and a well prepared vocabulary, our rule-based query translation approach can achieve an accuracy of 0.927, and a MRR (mean reciprocal rank) of 0.953.

2. When a free-text query contains out-of-vocabulary (OOV) words, our probabilistic query translation approach can effectively determine which OOV words to assign to which fields, achieving a MRR of 0.738.

3. When a free-text query does not contain OOV words, our probabilistic query translation approach can achieve a MRR of 0.996.

4. Our stack decoding approach is both efficient and effective for translating free-text queries to structured queries for multiple web forms, achieving a MAP of over 0.55.

5. The stopping criteria largely determine the time spent on decoding a query as well as the MAP that can be achieved. The criteria should be chosen such that the highest MAP is achieved within the amount of time one is willing to wait to decode a query.

6. The trade-off between effectiveness and efficiency is not linear.

7. In a given set of stopping criteria, training, boosting, and discounting each increase the efficiency (increasing effectiveness and lowering decoding time) of our stack decoding approach.

8. The effects of training, boosting, and discounting add up. Together they increase the efficiency our stack decoding approach, leading to a total increase in MAP by 12% and a total decrease in decoding time by 28%.

7.1.2 When end-users interact with the free-text search system

RQ3: Do end-users prefer to use a free-text interface rather than a complex web form for submitting structured queries?

In the case of a single domain and single site, we have shown that end-users prefer the free-text interface over the complex web form. Both in our laboratory experiment ($N = 17$) as well as in our public demo experiment ($N = 116$), a significant number of end-users indicated that they prefer the free-text interface over the complex form ($p \leq 0.05$). Furthermore, we have seen that the more frequent a user uses a web search engine, the more pronounced that user's preference is for the free-text interface. In the case of multiple domains and multiple sites per domain, further study is needed to find out whether or not end-users prefer one single free-text interface over multiple complex web forms.

RQ4: How do end-users phrase free-text queries when they intend to describe structured queries?

First, single users formulate free-text queries by consistently using a similar query template, as the within-user analysis showed a high correlation of 0.88 for the query formulations of end-users. However, there was great variety between end-users. After analyzing over 30,000 queries for a form with 6 input fields, we found almost 1,500 unique templates of which over 400 templates occurred at least 4 times. Second, in nearly 3 out of 4 queries, end-users fully typed in their query without making use of query suggestions. Even so, they perceived that they could obtain results faster when using our free-text interface instead of using a complex web form. This feeling was confirmed by our objective measurements: using our free-text interface, end-users finished their search tasks 9% faster than when using the original web form. Third, end-users seem to be susceptible to the example query shown in the free-text interface, as we have seen that the most frequent template (of valid queries) corresponds to the template of the example query: 7.7% of all queries had this template. Fourth and finally, after submitting an invalid query, end-users were able to rephrase their query into a valid query, requiring less than 2 reformulations on average.

RQ5: What are the most frequent mistakes that should be taken into account in future free-text systems?

The three most frequent mistakes that were causing unsatisfying results were: *i*) spelling errors in the free-text query (20%); *ii*) the use of tokens or synonyms that were not in the system's vocabulary (18%); and, *iii*) tokens that were concatenated by a dash, or due to the lack of a space between two tokens (17%). Perhaps the least difficult of the three problems is dealing with concatenated tokens since, of the three approaches that we have described in this thesis, the stack decoding approach is capable of detecting and separating concatenated tokens. The stack decoder can also correct certain kinds of spelling errors, however, its spelling correcting capabilities are limited. One alternative besides correcting spelling errors is to clearly indicate that the system cannot interpret or disambiguate a certain part of the query. Showing clear error messages to the end-user and providing the means to explicitly indicate how a query should be interpreted, e.g., by reverting to a complex form, may also be a possible solution for dealing with tokens that are not in the system's vocabulary.

Conclusions

9. In the case of a single domain and single site, a significant number of end-users prefer a free-text interface over a single complex web form ($p \leq 0.05$).

10. Using a free-text interface, end-users find results 9% faster than using a complex web form.

11. Query formulations are generally consistent within end-users, showing a within-user correlation of 0.88.

12. There is great variation in how queries are formulated. We found almost 1,500 unique query templates of which over 400 templates occurred at least 4 times.

13. After submitting an invalid query, end-users were able to rephrase their query into a valid query in less than 2 reformulations on average.

14. End-users are guided by the example shown in the free-text interface. The majority of the valid queries (7.7%) had the same template as that of the example query.

7.2 Future directions

In this thesis, we focused on "one-shot" queries, that is, when each query is evaluated on its own and the results of that query are not influenced by previous queries and previous results in the same search session. Consequently, each query should completely specify what information the end-user is searching for. However, there are situations where, rather than repeating the question and changing one or two words, it is easier and more natural for the end-user to ask a short follow-up question, as if the end-user is having a dialog with the system.

Free-text interfaces that support dialog sessions are a promising direction for future work. In such sessions, a new query may refer to information that has been specified either in previous queries or in previous results during the session. The key aspect here is that future systems should maintain some kind of dynamic contextual information for interpreting queries within a session.

We can generalize this notion of contextual information to include more than just information acquired from the dialogue. For example, we could consider the geographical locations of the end-user and of the named entities mentioned in the dialogue. Further research will have to point out what kind of contextual information is useful for better interpreting queries within a session. Though it is tempting to say that one should use user profiles that have been built up over longer periods of time, this potentially touches upon privacy concerns that should be clarified first. On a final note, we remark that it is not only search algorithms that have to change. User interfaces through which we specify our information needs should change accordingly in order to provide ample support for formulating complex queries. There is no point in having a system that can handle millions of data items per second, if the infrastructure to feed data to the system can only handle a few items per second. Though perhaps, the biggest change should be the way we humans, we creatures of habit, search on the web.

Bibliography

G. Agarwal, G. Kabra, and K. C.-C. Chang. Towards rich query interpretation: walking back and forth for mining query templates. In *Proceedings of the 19th international conference on World wide web (WWW '10)*, pages 1–10, New York, NY, USA, 2010. ACM.

F. Ahmad and G. Kondrak. Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT '05)*, pages 955–962, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

A. Alba, V. Bhagwan, and T. Grandison. Accessing the deep web: when good ideas go bad. In *Companion to the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '08)*, pages 815–818, New York, NY, USA, 2008. ACM.

M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro. Deepbot: a focused crawler for accessing hidden web content. In *Proceedings of the 3rd international workshop on Data enginering issues in E-commerce and services (DEECS '07)*, pages 18–25, New York, NY, USA, 2007. ACM.

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(01):29–81, 1995.

D. E. Appelt and B. Onyshkevych. The common pattern specification language. In *Proceedings of a workshop on held at Baltimore, Maryland*, TIPSTER '98, pages 23–30, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE '07)*, pages 6–20. IEEE, April 2007.

L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in speech recognition*, pages 308–319. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the 16th international conference on World Wide Web (WWW '07)*, pages 441–450, New York, NY, USA, 2007. ACM.

M. Bendersky and W. B. Croft. Analysis of long queries in a large scale search log. In *Proceedings of the 2009 workshop on Web Search Click Data (WSCD '09)*, pages 8–14, New York, NY, USA, 2009. ACM.

M. K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), August 2001.

J. M. Bland and D. G. Altman. Multiple significance tests: the bonferroni method. *BMJ*, 310(6973):170, 1995.

V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of the 2001 international conference on Management of data (SIGMOD '01)*, pages 175–186, New York, NY, USA, 2001. ACM.

J. Brooke. Sus: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.

R. R. Burton. Semantic grammar: An engineering technique for constructing natural language understanding systems. Technical report, Bolt, Beranek and Newman, Inc., Cambridge, MA., Dec. 1976.

M. J. Cafarella. Extracting and querying a comprehensive web database. In *CIDR*, 4th Biennial Conference on Innovative Data Systems Research, Jan. 2009.

M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the Very Large Database Endowment*, 1(1):538–549, 2008a.

M. J. Cafarella, J. Madhavan, and A. Halevy. Web-scale extraction of structured data. *SIGMOD Record*, 37(4):55–61, 2008b.

P. Calado, A. S. da Silva, A. H. F. Laender, B. A. Ribeiro-Neto, and R. C. Vieira. A bayesian network approach to searching web databases through keyword-based queries. *Information Processing and Management*, 40(5):773–790, September 2004.

A. Calì and D. Martinenghi. Querying the deep web. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT '10)*, pages 724–727, New York, NY, USA, 2010. ACM.

J. Callan. Distributed information retrieval. In *Advances in Information Retrieval*, pages 127–150. Kluwer Academic Publishers, 2000.

J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems (TOIS '01)*, 19(2):97–130, 2001.

J. G. Carbonell and P. J. Hayes. Dynamic strategy selection in flexible parsing. In *Proceedings of the 19th annual meeting of the Association for Computational Linguistics (ACL '81)*, pages 143–147, Morristown, NJ, USA, 1981. Association for Computational Linguistics.

J. G. Carbonell, W. M. Boggs, M. L. Mauldin, and P. G. Anick. The XCALIBUR project: a natural language interface to expert systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI '83)*, pages 653–656, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: observations and implications. *SIGMOD Record*, 33(3):61–70, 2004a.

K. C.-C. Chang, B. He, and Z. Zhang. Metaquerier over the deepweb: Shallow integration across holistic sources. In *Proceedings of the VLDB Workshop on Information Integration on the Web (VLDB-IIWeb'04)*, 2004b.

S.-L. Chuang, K. C.-C. Chang, and C. Zhai. Context-aware wrapping: synchronized data extraction. In *Proceedings of the 33rd international conference on Very large data bases (VLDB '07)*, pages 699–710. VLDB Endowment, 2007.

E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4:219–230, January 2011.

S. Dar, G. Entin, S. Geva, and E. Palmon. Dtl's dataspot: Database exploration using plain language. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98)*, pages 645–649, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

T. Demeester, D.-P. Nguyen, D. Trieschnigg, C. Develder, and D. Hiemstra. What snippets say about pages in federated web search. In *Proceedings of the 8th Asia Information Retrieval Societies Conference (AIRS 2012), Tianjin, China*, Lecture Notes in Computer Science. Springer, 2012.

E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. Divq: diversification for keyword search over structured databases. In *Proceeding of the 33rd International conference on Research and development in information retrieval (SIGIR '10)*, pages 331–338, New York, NY, USA, 2010. ACM.

E. C. Dragut, F. Fang, C. Yu, and W. Meng. Deriving customized integrated web query interfaces. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '09)*, volume 1, pages 685–688, Washington, DC, USA, 2009a. IEEE Computer Society.

E. C. Dragut, T. Kabisch, C. Yu, and U. Leser. A hierarchical approach to model web query interfaces for web source integration. *Proceedings of the Very Large Database Endowment*, 2(1):325–336, 2009b.

E. C. Dragut, W. Meng, and C. T. Yu. *Deep Web Query Interface Understanding and Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Communications*, 17(2):57–61, Apr. 2004.

D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Record*, 27(3):59–74, Sept. 1998.

J. Forney, G.D. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, Mar. 1973.

M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

D. Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Information Sciences*, 179(12):1822–1843, May 2009.

A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Special interest tracks and posters of the 14th international conference on World Wide Web (WWW '05)*, pages 902–903, New York, NY, USA, 2005. ACM.

M. Hagen, M. Potthast, B. Stein, and C. Braeutigam. Query segmentation revisited. In *Proceedings of the 20th international conference on World wide web (WWW '11)*, pages 97–106, New York, NY, USA, 2011. ACM.

A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *Proceedings of the twenty-fifth symposium on Principles of database systems (PODS '06)*, pages 1–9, New York, NY, USA, 2006a. ACM.

A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases (VLDB '06)*, pages 9–16. VLDB Endowment, 2006b.

B. He, T. Tao, and K. C.-C. Chang. Organizing structured web sources by query schemas: a clustering approach. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management (CIKM '04)*, pages 22–31, New York, NY, USA, 2004a. ACM.

B. He, Z. Zhang, and K. C.-C. Chang. Metaquerier: querying structured web sources on-the-fly. In *Proceedings of the 2005 international conference on Management of data (SIGMOD '05)*, pages 927–929, New York, NY, USA, 2005. ACM.

H. He, W. Meng, C. Yu, and Z. Wu. Automatic integration of web search interfaces with wise-integrator. *The VLDB Journal*, 13(3):256–273, September 2004b.

M. A. Hearst. 'natural' search user interfaces. *Communications of the ACM*, 54 (11):60–67, Nov. 2011.

S. Heinz, J. Zobel, and H. E. Williams. Burst tries: a fast, efficient data structure for string keys. *Transactions on Information Systems*, 20(2):192–223, Apr. 2002.

G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *Transactions on Database Systems*, 3(2):105–147, 1978.

D. Hiemstra and D. A. van Leeuwen. Creating an information retrieval test corpus for dutch. In M. Theune, A. Nijholt, and G. H. W. Hondorp, editors, *Computational Linguistics in the Netherlands 2001.*, volume 45 of *Language and Computers - Studies in Practical Linguistics*, pages 133–147, Amsterdam, The Netherlands, 2002. Rodopi.

B. J. Jansen. Search log analysis: What it is, what's been done, how to do it. *Library & Information Science Research*, 28(3):407–432, 2006.

T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin*, 29 (1), 2006.

T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '02)*, pages 133–142, New York, NY, USA, 2002. ACM.

T. Kabisch, E. C. Dragut, C. Yu, and U. Leser. Deep web integration with visqi. *Proceedings of the Very Large Database Endowment*, 3(2):1613–1616, September 2010.

E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar semantic search: a database approach to information retrieval. In *Proceedings of the 2006 international conference on Management of data (SIGMOD '06)*, pages 790–792, New York, NY, USA, 2006. ACM.

E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2010.

M. Kendall. *Rank Correlation Methods.* Second impression. Charles Griffin, 4th edition, 1975.

R. Khare, Y. An, and I.-Y. Song. Understanding deep web search interfaces: a survey. *SIGMOD Record*, 39(1):33–40, September 2010.

J. Kiseleva, Q. Guo, E. Agichtein, D. Billsus, and W. Chai. Unsupervised query segmentation using click data: preliminary results. In *Proceedings of the 19th international conference on World wide web (WWW '10)*, pages 1131–1132, New York, NY, USA, 2010. ACM.

J. Kiseleva, E. Agichtein, and D. Billsus. Mining query structure from click data: a case study of product queries. In *Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM '11)*, pages 2217–2220, New York, NY, USA, 2011. ACM.

N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15 – 68, 2000.

N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 729–737. Morgan Kaufmann, Aug. 1997.

M. H. Kutner, C. J. Nachtsheim, J. Neter, and W. Li. *Applied linear statistical models*. McGraw-Hill, 5th edition, 2005.

A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

S. Lawrence and C. L. Giles. Accessibility of information on the web. *Intelligence*, 11(1):32–39, 2000.

X. Li, Y.-Y. Wang, and A. Acero. Extracting structured information from user queries with semi-supervised conditional random fields. In *Proceedings of the 32nd international conference on Research and development in information retrieval (SIGIR '09)*, pages 572–579, New York, NY, USA, 2009. ACM.

Y. Li, B.-J. P. Hsu, C. Zhai, and K. Wang. Unsupervised query segmentation using clickthrough for information retrieval. In *Proceedings of the 34th international conference on Research and development in information retrieval (SIGIR '11)*, pages 285–294, New York, NY, USA, 2011. ACM.

W. Liu, X. Meng, and W. Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22 (3):447–460, 2010.

J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *Proceedings of the Very Large Database Endowment*, 1(2):1241–1252, 2008.

J. Madhavan, L. Afanasiev, L. Antova, and A. Halevy. Harnessing the deep web: Present and future. In *CIDR*, 4th Biennial Conference on Innovative Data Systems Research (CIDR '09), Jan. 2009.

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1st edition, May 1999.

F. McCown and M. L. Nelson. Search engines and their public interfaces: which apis are the most synchronized? In *Proceedings of the 16th international conference on World Wide Web (WWW '07)*, pages 1197–1198, New York, NY, USA, 2007. ACM.

F. Meng. A natural language interface for information retrieval from forms on the world wide web. In *Proceedings of the 20th International Conference on Information Systems (ICIS '99)*, pages 540–545, Atlanta, GA, USA, 1999. Association for Information Systems.

M.-J. Nederhof, G. Bouma, R. Koeling, and G. van Noord. Grammatical analysis in the ovis spoken-dialogue system. In *In Proceedings of the ACL/EACL Workshop on Spoken Dialog Systems*, pages 66–73, 1997.

L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition*. Prentice Hall, Inc., Upper Saddle River, NJ, USA, 1993.

S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, pages 129–138, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

L. A. Ramshaw and M. P. Marcus. Text Chunking using Transformation-Based Learning. *Computing Research Repository*, cmp-lg/950, 1995.

N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *Proceedings of the 2010 international conference on Management of data (SIGMOD '10)*, pages 771–782, New York, NY, USA, 2010. ACM.

P. Senellart, A. Mittal, D. Muschick, R. Gilleron, and M. Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceeding of the 10th ACM workshop on Web information and data management (WIDM '08)*, pages 9–16, New York, NY, USA, 2008. ACM.

L. Si and J. Callan. Modeling search engine effectiveness for federated search. In *Proceedings of the 28th annual international conference on Research and development in information retrieval (SIGIR '05)*, pages 83–90, New York, NY, USA, 2005. ACM.

C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, Sept. 1999.

A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American society for information science and technology*, 52(3):226–234, 2001.

S. Tata and G. M. Lohman. Sqak: doing more with keywords. In *Proceedings of the 2008 international conference on Management of data (SIGMOD '08)*, pages 889–902, New York, NY, USA, 2008. ACM.

A. Termehchy and M. Winslett. Using structural information in xml keyword search effectively. *Transactions on Database Systems*, 36:4:1–4:39, 2011.

C. W. Thompson, P. Pazandak, and H. R. Tennant. Talk to your semantic web. *IEEE Internet Computing*, 9(6):75–78, 2005.

K. Tjin-Kam-Jet. Research proposal for distributed deep web search. In *Proceedings of the 3rd workshop on Ph.D. students in information and knowledge management (PIKM '10)*, pages 33–38, New York, NY, USA, 2010. ACM.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Free-text search versus complex web forms. In P. Clough, C. Foley, C. Gurrin, G. Jones, W. Kraaij, H. Lee, and V. Mudoch, editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, pages 670–674. Springer Berlin / Heidelberg, 2011a.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Free-text search over complex web forms. In *Multidisciplinary Information Retrieval*, volume 6653 of *Lecture Notes in Computer Science*, page 14. Springer Berlin / Heidelberg, 2011b.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Deep web search: an overview and roadmap. Technical Report TR-CTIT-12-32, Centre for Telematics and Information Technology, University of Twente, Enschede, October 2011c.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. An analysis of free-text queries for a multi-field web form. In *Proceedings of the 4th Information Interaction in Context Symposium (IIIX '12)*, pages 82–89, New York, August 2012a. ACM.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. A probabilistic approach for mapping free-text queries to complex web forms. Technical Report TR-CTIT-12-33, Centre for Telematics and Information Technology, University of Twente, Enschede, January 2012b.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Using a stack decoder for structured search. In *Flexible Query Answering Systems*, volume 8132 of *Lecture Notes in Computer Science*, pages 519–530. Springer Berlin Heidelberg, 2013.

T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based interpretation of keywords for semantic search. In *The 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC '07/ASWC '07)*, pages 523–536, Berlin, Heidelberg, 2007. Springer-Verlag.

D. Trieschnigg, K. Tjin-Kam-Jet, and D. Hiemstra. Ranking xpaths for extracting search result records. Technical Report TR-CTIT-12-08, Centre for Telematics and Information Technology, University of Twente, Enschede, March 2012.

G. V. van Zanten, G. Bouma, K. Sima'an, G. van Noord, and R. Bonnema. Evaluation of the nlp components of the ovis2 spoken dialogue system. *CoRR*, cs.CL/9906014, 1999.

E. M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing and Management*, 36(5):697–716, 2000.

J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *Proceedings of the 12th international conference on World Wide Web (WWW '03)*, pages 187–196, New York, NY, USA, 2003. ACM.

J. Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9 (1):36–45, 1966.

T. Weninger, F. Fumarola, R. Barber, J. Han, and D. Malerba. Unexpected results in automatic list extraction on the web. *SIGKDD Explorations*, 12(2): 26–30, 2010.

R. W. White and G. Marchionini. Examining the effectiveness of real-time query expansion. *Information Processing and Management*, 43(3):685–704, 2007.

P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*, pages 47–57, April 2006.

W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proceedings of the 2004 international conference on Management of data (SIGMOD '04)*, pages 95–106, New York, NY, USA, 2004. ACM.

X. Yu and H. Shi. Query segmentation using conditional random fields. In *Proceedings of the First International Workshop on Keyword Search on Structured Data (KEYS '09)*, pages 21–26, New York, NY, USA, 2009. ACM.

Y. Zhang and S. Clark. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151, 2011.

Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *Proceedings of the 2004 international conference on Management of data (SIGMOD '04)*, pages 107–118, New York, NY, USA, 2004. ACM.

H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international conference on World Wide Web (WWW '05)*, pages 66–75, New York, NY, USA, 2005. ACM.

P. Zhao, L. Huang, W. Fang, and Z. Cui. Organizing structured deep web by clustering query interfaces link graph. In *Proceedings of the 4th international conference on Advanced Data Mining and Applications (ADMA '08)*, pages 683–690, Berlin, Heidelberg, 2008. Springer-Verlag.

S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '07)*, pages 894–902, New York, NY, USA, 2007. ACM.

Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: adapting keyword query to semantic search. In *The 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference (ISWC '07/ASWC '07)*, pages 694–707, Berlin, Heidelberg, 2007. Springer-Verlag.

List of the author's publications

Articles related to this thesis

K. Tjin-Kam-Jet. Research proposal for distributed deep web search. In *Proceedings of the 3rd workshop on Ph.D. students in information and knowledge management (PIKM '10)*, pages 33–38, New York, NY, USA, 2010. ACM.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Free-text search versus complex web forms. In P. Clough, C. Foley, C. Gurrin, G. Jones, W. Kraaij, H. Lee, and V. Mudoch, editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, pages 670–674. Springer Berlin / Heidelberg, 2011a.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Free-text search over complex web forms. In *Multidisciplinary Information Retrieval*, volume 6653 of *Lecture Notes in Computer Science*, page 14. Springer Berlin / Heidelberg, 2011b.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Onebox: Free-text interfaces as an alternative to complex web forms. Technical Report TR-CTIT-11-01, Centre for Telematics and Information Technology University of Twente, Enschede, 2011c.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Deep web search: an overview and roadmap. Technical Report TR-CTIT-12-32, Centre for Telematics and Information Technology, University of Twente, Enschede, October 2011d.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. An analysis of free-text queries for a multi-field web form. In *Proceedings of the 4th Information Interaction in Context Symposium (IIIX '12)*, pages 82–89, New York, August 2012a. ACM.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. A probabilistic approach for mapping free-text queries to complex web forms. Technical Report TR-CTIT-12-33, Centre for Telematics and Information Technology, University of Twente, Enschede, January 2012b.

D. Trieschnigg, K. Tjin-Kam-Jet, and D. Hiemstra. Ranking xpaths for extracting search result records. Technical Report TR-CTIT-12-08, Centre for Telematics and Information Technology, University of Twente, Enschede, March 2012.

D. Trieschnigg, K. Tjin-Kam-Jet, and D. Hiemstra. Searchresultfinder: federated search made easy. In *Proceedings of the 36th international conference on Research and development in information retrieval (SIGIR '13)*, pages 1113–1114, New York, NY, USA, 2013. ACM.

K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Using a stack decoder for structured search. In *Flexible Query Answering Systems*, volume 8132 of *Lecture Notes in Computer Science*, pages 519–530. Springer Berlin Heidelberg, 2013.

Public demonstrations related to this thesis

`http://treinplanner.info`

Other articles

K. Tjin-Kam-Jet and D. Hiemstra. Learning to merge search results for efficient distributed information retrieval. In *The 10th Dutch-Belgian Information Retrieval Workshop, Nijmegen, the Netherlands (DIR '10)*, pages 55–62, Nijmegen, January 2010. Radboud University.

SIKS dissertation list

2013-41 Jochem Liem (UVA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.*
2013-40 Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games.*
2013-39 Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems.*
2013-38 Eelco den Heijer (VU), *Autonomous Evolutionary Art.*
2013-37 Dirk Brner (OUN), *Ambient Learning Displays.*
2013-36 Than Lam Hoang (TUe), *Pattern Mining in Data Streams.*
2013-35 Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction.*
2013-34 Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search.*
2013-33 Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere.*
2013-32 Kamakshi Rajagopal (OUN), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.*
2013-31 Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications.*
2013-30 Joyce Nakatumba (TUE), *Resource-Aware Business Process Management: Analysis and Support.*
2013-29 Iwan de Kok (UT), *Listening Heads.*
2013-28 Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience.*
2013-27 Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance.*
2013-26 Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning.*
2013-25 Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System.*
2013-24 Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning.*
2013-23 Patricio de Alencar Silva(UvT), *Value Activity Monitoring.*
2013-22 Tom Claassen (RUN), *Causal Discovery and Logic.*
2013-21 Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation.*
2013-20 Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval.*
2013-19 Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling.*
2013-18 Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification.*
2013-17 Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid.*
2013-16 Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation.*

2013-15 Daniel Hennes (UM), *Multiagent Learning - Dynamic Games and Applications.*
2013-14 Jafar Tanha (UVA), *Ensemble Approaches to Semi-Supervised Learning Learning.*
2013-13 Mohammad Safiri(UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.*
2013-12 Marian Razavian(VU), *Knowledge-driven Migration to Services.*
2013-11 Evangelos Pournaras(TUD), *Multi-level Reconfigurable Self-organization in Overlay Services.*
2013-10 Jeewanie Jayasinghe Arachchige(UvT), *A Unified Modeling Framework for Service Design..*
2013-09 Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications.*
2013-08 Robbert-Jan Merk(VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.*
2013-07 Giel van Lankveld (UvT), *Quantifying Individual Player Differences.*
2013-06 Romulo Goncalves(CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.*
2013-05 Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns.*
2013-04 Chetan Yadati(TUD), *Coordinating autonomous planning and scheduling.*
2013-03 Szymon Klarman (VU), *Reasoning with Contexts in Description Logics.*
2013-02 Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.*
2013-01 Viorel Milea (EUR), *News Analytics for Financial Decision Support.*
2012-51 Jeroen de Jong (TUD), *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.*
2012-50 Steven van Kervel (TUD), *Ontologogy driven Enterprise Information Systems Engineering.*
2012-49 Michael Kaisers (UM), *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.*
2012-48 Jorn Bakker (TUE), *Handling Abrupt Changes in Evolving Time-series Data.*
2012-47 Manos Tsagkias (UVA), *Mining Social Media: Tracking Content and Predicting Behavior.*
2012-46 Simon Carter (UVA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.*
2012-45 Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions.*
2012-44 Anna Tordai (VU), *On Combining Alignment Techniques.*
2012-43 Withdrawn, .
2012-42 Dominique Verpoorten (OU), *Reflection Amplifiers in self-regulated Learning.*
2012-41 Sebastian Kelle (OU), *Game Design Patterns for Learning.*
2012-40 Agus Gunawan (UvT), *Information Access for SMEs in Indonesia.*
2012-39 Hassan Fatemi (UT), *Risk-aware design of value and coordination networks.*
2012-38 Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms.*
2012-37 Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation.*
2012-36 Denis Ssebugwawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes.*
2012-35 Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.*
2012-34 Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and applications.*

2012-33 Rory Sie (OUN), *Coalitions in Cooperation Networks (COCOON).*
2012-32 Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning.*
2012-31 Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.*
2012-30 Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making.*
2012-29 Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval.*
2012-28 Nancy Pascall (UvT), *Engendering Technology Empowering Women.*
2012-27 Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*
2012-26 Emile de Maat (UVA), *Making Sense of Legal Text.*
2012-25 Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.*
2012-24 Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval.*
2012-23 Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.*
2012-22 Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.*
2012-21 Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval.*
2012-20 Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.*
2012-19 Helen Schonenberg (TUE), *What's Next? Operational Support for Business Process Execution.*
2012-18 Eltjo Poort (VU), *Improving Solution Architecting Practices.*
2012-17 Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance.*
2012-16 Fiemke Both (VU), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.*
2012-15 Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes..*
2012-14 Evgeny Knutov(TUE), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*
2012-13 Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*
2012-12 Kees van der Sluijs (TUE), *Model Driven Design and Data Integration in Semantic Web Information Systems.*
2012-11 J.C.B. Rantham Prabhakara (TUE), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*
2012-10 David Smits (TUE), *Towards a Generic Distributed Adaptive Hypermedia Environment.*
2012-09 Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms.*
2012-08 Gerben de Vries (UVA), *Kernel Methods for Vessel Trajectories.*
2012-07 Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.*
2012-06 Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks.*
2012-05 Marijn Plomp (UU), *Maturing Interorganisational Information Systems.*

2012-04 Jurriaan Souer (UU), *Development of Content Management System-based Web Applications.*
2012-03 Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories.*
2012-02 Muhammad Umair(VU), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.*
2012-01 Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda.*
2011-49 Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.*
2011-48 Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.*
2011-47 Azizi Bin Ab Aziz(VU), *Exploring Computational Models for Intelligent Support of Persons with Depression.*
2011-46 Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.*
2011-45 Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection.*
2011-44 Boris Reuderink (UT), *Robust Brain-Computer Interfaces.*
2011-43 Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge.*
2011-42 Michal Sindlar (UU), *Explaining Behavior through Mental State Attribution.*
2011-41 Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control.*
2011-40 Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development.*
2011-39 Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games.*
2011-38 Nyree Lemmens (UM), *Bee-inspired Distributed Optimization.*
2011-37 Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.*
2011-36 Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach.*
2011-35 Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Training.*
2011-34 Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.*
2011-33 Tom van der Weide (UU), *Arguing to Motivate Decisions.*
2011-32 Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science.*
2011-31 Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.*
2011-30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*
2011-29 Faisal Kamiran (TUE), *Discrimination-aware Classification.*
2011-28 Rianne Kaptein(UVA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure.*
2011-27 Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns.*
2011-26 Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*
2011-25 Syed Waqar ul Qounain Jaffry (VU)), *Analysis and Validation of Models for Trust Dynamics.*

2011-24 Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.*
2011-23 Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media.*
2011-22 Junte Zhang (UVA), *System Evaluation of Archival Description and Access.*
2011-21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems.*
2011-20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach.*
2011-19 Ellen Rusman (OU), *The Mind ' s Eye on Personal Profiles.*
2011-18 Mark Ponsen (UM), *Strategic Decision-Making in complex games.*
2011-17 Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness.*
2011-16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity.*
2011-15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval.*
2011-14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*
2011-13 Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling.*
2011-12 Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining.*
2011-11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*
2011-10 Bart Bogaert (UvT), *Cloud Content Contention.*
2011-09 Tim de Jong (OU), *Contextualised Mobile Media for Learning.*
2011-08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*
2011-07 Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*
2011-06 Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*
2011-05 Base van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline..*
2011-04 Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference.*
2011-03 Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems.*
2011-02 Nick Tinnemeier(UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*
2011-01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*
2010-53 Edgar Meij (UVA), *Combining Concepts and Language Models for Information Access.*
2010-52 Peter-Paul van Maanen (VU), *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention.*
2010-51 Alia Khairia Amin (CWI), *Understanding and supporting information seeking tasks in multiple sources.*
2010-50 Bouke Huurnink (UVA), *Search in Audiovisual Broadcast Archives.*
2010-49 Jahn-Takeshi Saito (UM), *Solving difficult game positions.*
2010-48 Withdrawn, .
2010-47 Chen Li (UT), *Mining Process Model Variants: Challenges, Techniques, Examples.*
2010-46 Vincent Pijpers (VU), *e3alignment: Exploring Inter-Organizational Business-ICT Alignment.*

2010-45 Vasilios Andrikopoulos (UvT), *A theory and model for the evolution of software services.*
2010-44 Pieter Bellekens (TUE), *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain.*
2010-43 Peter van Kranenburg (UU), *A Computational Approach to Content-Based Retrieval of Folk Song Melodies.*
2010-42 Sybren de Kinderen (VU), *Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach.*
2010-41 Guillaume Chaslot (UM), *Monte-Carlo Tree Search.*
2010-40 Mark van Assem (VU), *Converting and Integrating Vocabularies for the Semantic Web.*
2010-39 Ghazanfar Farooq Siddiqui (VU), *Integrative modeling of emotions in virtual agents.*
2010-38 Dirk Fahland (TUE), *From Scenarios to components.*
2010-37 Niels Lohmann (TUE), *Correctness of services and their composition.*
2010-36 Jose Janssen (OU), *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification.*
2010-35 Dolf Trieschnigg (UT), *Proof of Concept: Concept-based Biomedical Information Retrieval.*
2010-34 Teduh Dirgahayu (UT), *Interaction Design in Service Compositions.*
2010-33 Robin Aly (UT), *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval.*
2010-32 Marcel Hiel (UvT), *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems.*
2010-31 Victor de Boer (UVA), *Ontology Enrichment from Heterogeneous Sources on the Web.*
2010-30 Marieke van Erp (UvT), *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval.*
2010-29 Stratos Idreos(CWI), *Database Cracking: Towards Auto-tuning Database Kernels.*
2010-28 Arne Koopman (UU), *Characteristic Relational Patterns.*
2010-27 Marten Voulon (UL), *Automatisch contracteren.*
2010-26 Ying Zhang (CWI), *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines.*
2010-25 Zulfiqar Ali Memon (VU), *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective.*
2010-24 Dmytro Tykhonov, *Designing Generic and Efficient Negotiation Strategies.*
2010-23 Bas Steunebrink (UU), *The Logical Structure of Emotions.*
2010-22 Michiel Hildebrand (CWI), *End-user Support for Access to Heterogeneous Linked Data.*
2010-21 Harold van Heerde (UT), *Privacy-aware data management by means of data degradation.*
2010-20 Ivo Swartjes (UT), *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative.*
2010-19 Henriette Cramer (UvA), *People's Responses to Autonomous and Adaptive Systems.*
2010-18 Charlotte Gerritsen (VU), *Caught in the Act: Investigating Crime by Agent-Based Simulation.*
2010-17 Spyros Kotoulas (VU), *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications.*
2010-16 Sicco Verwer (TUD), *Efficient Identification of Timed Automata, theory and practice.*

2010-15 Lianne Bodenstaff (UT), *Managing Dependency Relations in Inter-Organizational Models.*
2010-14 Sander van Splunter (VU), *Automated Web Service Reconfiguration.*
2010-13 Gianluigi Folino (RUN), *High Performance Data Mining using Bio-inspired techniques.*
2010-12 Susan van den Braak (UU), *Sensemaking software for crime analysis.*
2010-11 Adriaan Ter Mors (TUD), *The world according to MARP: Multi-Agent Route Planning.*
2010-10 Rebecca Ong (UL), *Mobile Communication and Protection of Children.*
2010-09 Hugo Kielman (UL), *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging.*
2010-08 Krzysztof Siewicz (UL), *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments.*
2010-07 Wim Fikkert (UT), *Gesture interaction at a Distance.*
2010-06 Sander Bakkes (UvT), *Rapid Adaptation of Video Game AI.*
2010-05 Claudia Hauff (UT), *Predicting the Effectiveness of Queries and Retrieval Systems.*
2010-04 Olga Kulyk (UT), *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments.*
2010-03 Joost Geurts (CWI), *A Document Engineering Model and Processing Framework for Multimedia documents.*
2010-02 Ingo Wassink (UT), *Work flows in Life Science.*
2010-01 Matthijs van Leeuwen (UU), *Patterns that Matter.*
2009-46 Loredana Afanasiev (UvA), *Querying XML: Benchmarks and Recursion.*
2009-45 Jilles Vreeken (UU), *Making Pattern Mining Useful.*
2009-44 Roberto Santana Tapia (UT), *Assessing Business-IT Alignment in Networked Organizations.*
2009-43 Virginia Nunes Leal Franqueira (UT), *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients.*
2009-42 Toine Bogers (UvT), *Recommender Systems for Social Bookmarking.*
2009-41 Igor Berezhnyy (UvT), *Digital Analysis of Paintings.*
2009-40 Stephan Raaijmakers (UvT), *Multinomial Language Learning: Investigations into the Geometry of Language.*
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin), *Service Substitution – A Behavioral Approach Based on Petri Nets.*
2009-38 Riina Vuorikari (OU), *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context.*
2009-37 Hendrik Drachsler (OUN), *Navigation Support for Learners in Informal Learning Networks.*
2009-36 Marco Kalz (OUN), *Placement Support for Learners in Learning Networks.*
2009-35 Wouter Koelewijn (UL), *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling.*
2009-34 Inge van de Weerd (UU), *Advancing in Software Product Management: An Incremental Method Engineering Approach.*
2009-33 Khiet Truong (UT), *How Does Real Affect Affect Affect Recognition In Speech?.*
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU), *Architectural Knowledge Management: Supporting Architects and Auditors.*
2009-31 Sofiya Katrenko (UVA), *A Closer Look at Learning Relations from Text.*
2009-30 Marcin Zukowski (CWI), *Balancing vectorized query execution with bandwidth-optimized storage.*

2009-29 Stanislav Pokraev (UT), *Model-Driven Semantic Integration of Service-Oriented Applications.*
2009-28 Sander Evers (UT), *Sensor Data Management with Probabilistic Models.*
2009-27 Christian Glahn (OU), *Contextual Support of social Engagement and Reflection on the Web.*
2009-26 Fernando Koch (UU), *An Agent-Based Model for the Development of Intelligent Mobile Services.*
2009-25 Alex van Ballegooij (CWI), *"RAM: Array Database Management through Relational Mapping".*
2009-24 Annerieke Heuvelink (VUA), *Cognitive Models for Training Simulations.*
2009-23 Peter Hofgesang (VU), *Modelling Web Usage in a Changing Environment.*
2009-22 Pavel Serdyukov (UT), *Search For Expertise: Going beyond direct evidence.*
2009-21 Stijn Vanderlooy (UM), *Ranking and Reliable Classification.*
2009-20 Bob van der Vecht (UU), *Adjustable Autonomy: Controling Influences on Decision Making.*
2009-19 Valentin Robu (CWI), *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets.*
2009-18 Fabian Groffen (CWI), *Armada, An Evolving Database System.*
2009-17 Laurens van der Maaten (UvT), *Feature Extraction from Visual Data.*
2009-16 Fritz Reul (UvT), *New Architectures in Computer Chess.*
2009-15 Rinke Hoekstra (UVA), *Ontology Representation - Design Patterns and Ontologies that Make Sense.*
2009-14 Maksym Korotkiy (VU), *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA).*
2009-13 Steven de Jong (UM), *Fairness in Multi-Agent Systems.*
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin), *Operating Guidelines for Services.*
2009-11 Alexander Boer (UVA), *Legal Theory, Sources of Law & the Semantic Web.*
2009-10 Jan Wielemaker (UVA), *Logic programming for knowledge-intensive interactive applications.*
2009-09 Benjamin Kanagwa (RUN), *Design, Discovery and Construction of Service-oriented Systems.*
2009-08 Volker Nannen (VU), *Evolutionary Agent-Based Policy Analysis in Dynamic Environments.*
2009-07 Ronald Poppe (UT), *Discriminative Vision-Based Recovery and Recognition of Human Motion.*
2009-06 Muhammad Subianto (UU), *Understanding Classification.*
2009-05 Sietse Overbeek (RUN), *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality.*
2009-04 Josephine Nabukenya (RUN), *Improving the Quality of Organisational Policy Making using Collaboration Engineering.*
2009-03 Hans Stol (UvT), *A Framework for Evidence-based Policy Making Using IT.*
2009-02 Willem Robert van Hage (VU), *Evaluating Ontology-Alignment Techniques.*
2009-01 Rasa Jurgelenaite (RUN), *Symmetric Causal Independence Models.*

Summary

The World Wide Web contains *billions* of documents (and counting); hence, it is likely that *some* document will contain the answer or content you are searching for. While major search engines like Bing and Google often manage to return relevant results to your query, there are plenty of situations in which they are less capable of doing so. Specifically, there is a noticeable shortcoming in situations that involve the retrieval of data from the *deep web*. Deep web data is difficult to crawl and index for today's web search engines, and this is largely due to the fact that the data must be accessed via complex web forms. However, deep web data can be *highly relevant* to the information-need of the end-user. This thesis overviews the problems, solutions, and paradigms for deep web search. Moreover, it proposes a new paradigm to overcome the apparent limitations in the current state of deep web search, and makes the following scientific contributions:

1. A more specific classification scheme for deep web search systems, to better illustrate the differences and variation between these systems.

2. *Virtual surfacing*, a new, and in our opinion better, deep web search paradigm which tries to combine the benefits of the two already existing paradigms, *surfacing* and *virtual integration*, and which also raises new research opportunities.

3. A stack decoding approach which combines rules and statistical usage information for interpreting the end-user's *free-text query*, and to subsequently derive filled-out web forms based on that interpretation.

4. A practical comparison of the developed approach against a well-established text-processing toolkit.

5. Empirical evidence that, for a single site, end-users would rather use the proposed free-text search interface instead of a complex web form.

Analysis of data obtained from user studies shows that the stack decoding approach works as well as, or better than, today's top-performing alternatives.

Samenvatting

Het Wereldwijde Web bevat miljarden documenten (en tellende). Het is dus waarschijnlijk dat er *ergens* een document zal zijn dat het antwoord of de inhoud bevat waar je naar op zoek bent. Hoewel het grote zoekmachines zoals Bing en Google vaak lukt om relevante documenten als antwoord op je vraag te leveren, zijn er ook talrijke situaties waarin ze minder capabel zijn om dit voor elkaar te krijgen. Dit is in het bijzonder merkbaar in situaties waarbij er gezocht wordt naar gegevens uit het *diepe web*. Zoekmachines worstelen om data uit het diepe web te verzamelen en te indexeren, wat vooral komt doordat deze data eerst verkregen moet worden via complexe webformulieren. Echter, het diepe web kan gegevens bevatten die *zeer relevant* zijn voor de informatiebehoefte van de eindgebruiker. Dit proefschrift geeft een overzicht van de problemen, oplossingen en paradigma's voor het *zoeken in het diepe web*. Tevens stelt het een nieuw paradigma voor om de huidige beperkingen te overwinnen en levert het de volgende wetenschappelijke bijdragen:

1. Een specifiekere classificatie van zoeksystemen voor het diepe web, om de verschillen en variëteit beter in kaart te brengen.

2. *Virtual surfacing*, een nieuw en volgens ons beter paradigma voor het zoeken in het diepe web wat de voordelen van de twee al bestaande paradigma's, *surfacing* en *virtual integration*, probeert te combineren en ook nieuwe onderzoeksmogelijkheden biedt.

3. Een *stack decoding* aanpak waarin regels en statistische gebruiksinformatie gecombineerd worden om de zoekvraag te interpreteren, om zodoende een webformulier mee te kunnen invullen.

4. Een praktische vergelijking tussen de nieuwe aanpak en gevestigde software voor de verwerking van natuurlijke taal.

5. Empirisch bewijs dat, voor een enkele site, eindgebruikers liever de voorgestelde zoekinterface gebruiken dan een complex webformulier.

Uit de analyse van data die is verkregen uit gebruikersonderzoeken blijkt dat de *stack decoding* aanpak net zo goed is als, of beter is dan, de best presterende hedendaagse alternatieven.