



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

ASSIGNMENT OF BACHELOR'S THESIS

Title: Autocomplete in the system TA ČR Starfos
Student: Róbert Schönfeld
Supervisor: Ing. Jiří Novák, Ph.D.
Study Programme: Informatics
Study Branch: Knowledge Engineering
Department: Department of Applied Mathematics
Validity: Until the end of summer semester 2019/20

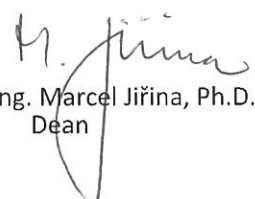
Instructions

- 1) Describe the system TA ČR Starfos and analyze the requirements for enhancing the user experience of search in the application.
- 2) Study methods and libraries for building a corpus of autocomplete suggestions. In particular the extraction of keywords and phrases from documents in Czech and English languages. Implement the appropriate methods for both languages.
- 3) Design and implement the autocomplete feature as an HTTP service using the web framework Django.
- 4) Study algorithms and libraries for ranking of autocomplete suggestions. Choose an algorithm to implement and evaluate.
- 5) Implement a feature for suggesting documents and their URLs instead of keywords.

References

Will be provided by the supervisor.


Ing. Karel Klouda, Ph.D.
Head of Department


doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 6, 2019



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

Autocomplete in the system TA ČR Starfos

Róbert Schönfeld

Department of Applied Mathematics

Supervisor: Ing. Jiří Novák, Ph.D.

January 20, 2020

Acknowledgements

I would like to thank my supervisor for his infinite supply of patience and my team at TA CR for their support, which made this work possible.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on January 20, 2020

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2020 Róbert Schönfeld. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Schönfeld, Róbert. *Autocomplete in the system TA ČR Starfos*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Našeptávače jsou rozšířená technologie ve vyhledávání, asistující uživateli při formulaci vyhledávacích dotazů. Tato práce se zabývá implementací našeptávače v existující aplikaci TA ČR Starfos. Je představena dekompozice problémové domény včetně problematiky uživatelských rozhraní vyhledávačů. Dosažena je uspokojivá implementace s použitím Apache Solr založená na autorem vytvořených klíčových slověch, která poskytuje základ pro budoucí vývoj.

Klíčová slova našeptávač, získávání informací, automatická extrakce klíčových slov, zpracování přirozeného jazyka, uživatelské rozhraní pro vyhledávání, Apache Solr, informační systém výzkumu a vývoje

Abstract

Query auto completion(QAC) is a widespread information retrieval technology for aiding query specification in search systems. This thesis is concerned with the implementation of QAC in an existing application TA ČR Starfos. A decomposition of the problem domain is provided including a study of search user interfaces. A satisfactory implementation using Apache Solr based on author-assigned keywords is achieved and provides a foundation for further development.

Keywords query auto completion, information retrieval, automatic keyphrase extraction, natural language processing, Apache Solr, search user interface current research information system

Contents

Introduction	1
Goals	3
1 Problem context	5
1.1 Technology Agency of the Czech Republic	5
1.2 IS VaVaI	5
1.2.1 CEP Projects and RIV Results	6
1.3 TA ČR Starfos	7
2 Search user interfaces	9
2.1 Models of information seeking	9
2.1.1 Iterative refinement	9
2.1.2 Berry-picking	10
2.1.3 The information foraging theory	10
2.1.4 Query taxonomy and QAC	10
2.2 Query specification and syntax	11
2.2.1 Boolean queries	11
2.2.2 Command languages	12
2.2.3 Modern keyword search	13
2.2.4 The search box	14
2.3 Current web search engines	14
2.3.1 Google search	15
2.4 Faceted search	16
2.4.1 Nested facets and dashboards	17
2.5 QAC design patterns	18
2.5.1 Keyword query suggestions	18
2.5.2 Direct entity suggestions	19
2.5.3 Search history suggestions	20
2.5.4 Interactivity and responsiveness	20

2.5.5	Presentation of suggestions	20
2.5.6	Instant search	21
3	Theory	23
3.1	Building the corpus of QAC suggestions	23
3.1.1	From search query logs	23
3.1.2	From searched documents	24
3.2	Retrieving suggestions	25
3.2.1	Matching	25
3.2.1.1	Brute force	25
3.2.1.2	Inverted index based on n-grams	25
3.2.1.3	Prefix trees	26
3.2.2	Ranking	27
3.2.2.1	Heuristic models	27
3.2.2.2	Learning to rank	28
3.3	Automatic keyphrase extraction (AKE)	29
3.3.1	Text pre-processing	29
3.3.1.1	Stopwords	29
3.3.1.2	Stemming	30
3.3.1.3	Morphology	30
3.3.2	TF-IDF	31
3.3.3	RAKE	31
4	Implementation	33
4.1	Software requirements	33
4.1.1	Establishing goals	33
4.1.2	Suggestions	34
4.1.3	User interface	34
4.1.4	Technical requirements	35
4.2	Author-assigned keywords	35
4.2.1	Parsing	36
4.2.2	Normalization	36
4.3	AKE	37
4.4	Matching and ranking	38
4.4.1	HTTP web service	39
4.5	Front end	39
	Conclusion	41
	Bibliography	43
	A Acronyms	49
	B Contents of enclosed CD	51

List of Figures

1.1	Domain model of key entities in IS VaVaI	6
1.2	Production version of Starfos search available at starfos.tacr.cz . .	7
2.1	Form to generate boolean search query in Google Patents	12
2.2	A part of the current Google “Advanced search” form	13
2.3	Selection of facets for the computer mice on alza.cz	16
2.4	Example of a complex facet in Google Patents	17
2.5	Example of advanced Google QAC suggestions	18
3.1	Basic structure of a QAC system	23
3.2	Example of trie schema	26
3.3	Query re-ranking pipeline	28
4.1	The current version of Starfos’ QAC in development	40
4.2	Additional examples of achieved QAC results based on author- assigned keywords	40

Introduction

Query auto completion (**QAC**), autocomplete, autosuggest, typeahead search and incremental search all refer to the same information retrieval technology and its corresponding UI element used in search applications. One of its earliest appearances was in the Bash command line interpreter, in which options retrieved from a dictionary of command and file names were displayed in alphabetical order after a double press of the Tab key. Word prediction software was subsequently studied with the aim of increasing the typing speed of individuals with physical disabilities. [1]

In 2004, Google rolled out their implementation of QAC called *Google Suggest*. A drop-down list of query suggestions, updating instantly after every key stroke, appeared below the search box.[2] With the emergence of mobile devices, where the input method is especially time-consuming and error-prone, QAC powered search boxes identified by the magnifying glass icon have become universally recognizable UI elements.

Google QAC has made its way into pop culture as a simple barometer of trends. There is a whole book[3] dedicated to presenting a collection of entertaining Google QAC suggestions, one per page. Examples include “i wish i were a”, “what happens if you put a” and “why do millennials”.

The use of QAC in search engines operating on large data sets is enabled by the availability of databases and network infrastructure capable of sufficiently responsive retrieval of the suggestions. My implementation utilizes the open source search engine Apache Solr.

Goals

The aim of the thesis is summarized in the following 3 objectives:

1. Study and provide a description of the problem domain including
 - a) search user interfaces,
 - b) the current version of Starfos,
 - c) the IS VaVaI data set,
 - d) methods for matching and ranking of QAC suggestions.
 - e) methods for generating the QAC corpus,
 - f) automatic keyphrase extraction (**AKE**),
2. Implement a QAC web service for Starfos using Python, the web framework Django and the search engine Apache Solr, with key requirements:
 - a) Provide keyword query suggestions in both Czech and English languages.
 - b) Include suggestions for documents and their URLs.
 - c) Include suggestions for precise filtering on related entities, which are available in a structured form, i.e. filter by organization or author.
 - d) Ensure sufficiently low response time, i.e. 100 - 200 milliseconds.

Problem context

The motivation comes from my involvement in the project **Proeval**, specifically the development of the system TAČR Starfos. Proeval is an EU-funded internal project of the Technology Agency of the Czech Republic (**TA ČR**) with the subtitle *"Evidence-Based management of R&D funding programmes, improvement of analytical and data services of TA ČR"* (author's translation).

This chapter provides an overview of the domain and the existing application.

1.1 Technology Agency of the Czech Republic

The Technology Agency of the Czech Republic (**TA ČR**), an *organisational unit of the state*, is one of the providers of public R&D funding in the Czech Republic. TA ČR's purpose is to administer public contests in which funding is allocated to *applied research* projects submitted by university faculties, private companies and other research institutions.

1.2 IS VaVaI

The key data source for Starfos is **IS VaVaI**[4], the Czech *current research information system*. It offers extensive records of state supported projects and their research results from the past 30 years. This includes detailed financial records per year and participating organization.

IS VaVaI is presently maintained by **RVVI** (*the Research, Development and Innovation Council of the Government of the Czech Republic*)[5]. The data is produced and submitted by R&D funding providers such as TA ČR. RVVI is responsible for overseeing the data collection process in accordance with relevant legislation, maintaining the database and making it publicly available via their own web application and JSON API.

Concurrently with the development of Starfos, the IS VaVaI data set has been processed and stored in **DAFOS**, TA ČR's data warehouse. A fair amount of work has gone into cleaning and enriching the data. In particular dealing with the errors and inconsistencies caused by the sub-optimal data structure and historically inconsistent maintenance.

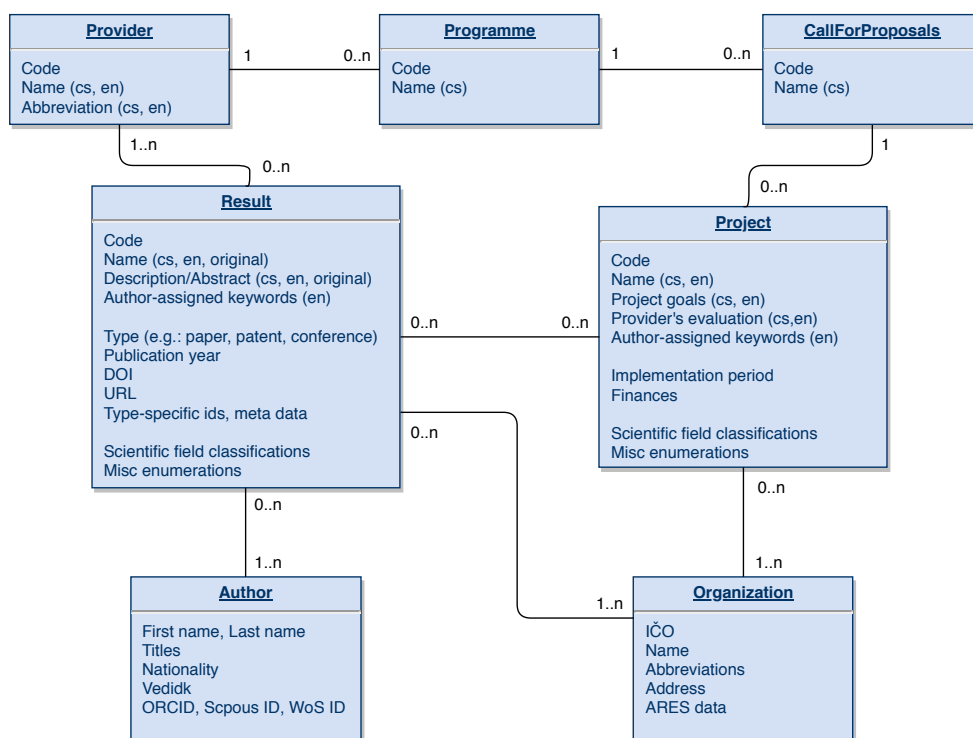


Figure 1.1: Domain model of key entities in IS VaVaI

1.2.1 CEP Projects and RIV Results

The two most valuable collections of documents both for Starfos and this work are *Projects* and *Results*.

CEP (Central register of R&D projects) A subsystem of IS VaVaI, containing 50k projects with roughly 2k being added each year. These represent a particular R&D activity with a defined goal and assigned state support.

RIV (Information Register of R&D results) Another subsystem, containing more than 1M results with 50k added annually. These are essentially meta data of scientific publications, or other types of results such as patents, software or organized workshops. Approximately 50% of them are linked to a project.

1.3 TA ČR Starfos

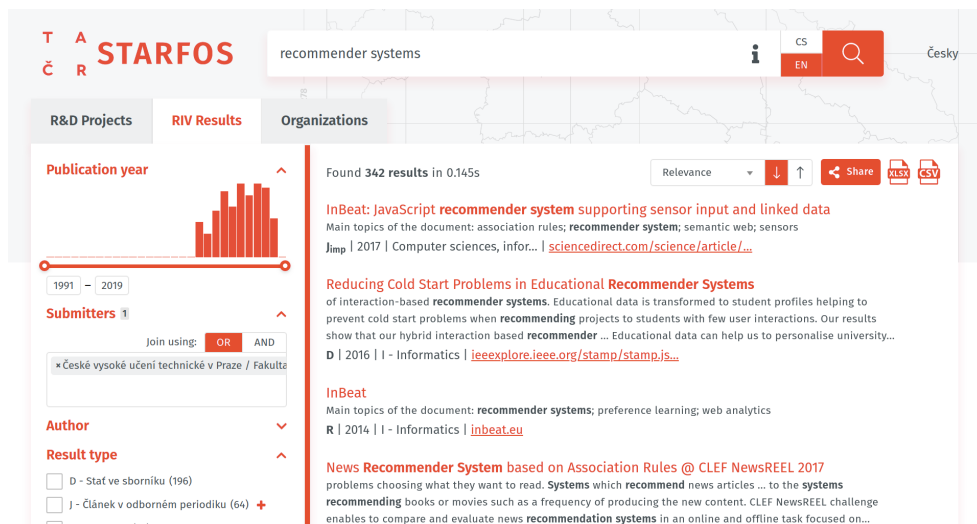


Figure 1.2: Production version of Starfos search available at starfos.tacr.cz

Starfos is a web application developed by TA ČR. Its purpose is to support both internal analytical processes and public presentation of Czech R&D.

The central feature is a full-text and faceted search interface on top of IS VaVaI data stored in DAFOS. Additional data sources in the R&D domain will likely be incorporated in the future. Extensive configuration options are available for the collections, filters and other components.

The functionality is not limited to search. There are document detail pages offering comprehensive data visualizations and reporting. A recent addition is TAČR v Datech, a simple content management system for publishing infographics which can take input from SQL queries.

Search user interfaces

Following UI standards is valuable especially for well-known interfaces like search and QAC. It removes the need for users to learn a new system. To users, the “big players” in the search engine market act as reference.[6] This chapter introduces some of these design patterns and describes a selection of popular search applications. A particularly useful resource for me was the book *Search User Interfaces by Marti Hearst* [7], which is outdated in places, but provides a very well-organized overview. It begins with this description of a search UI’s purpose:

The job of the search user interface is to aid users in the expression of their information needs, in the formulation of their queries, in the understanding of their search results, and in keeping track of the progress of their information seeking efforts.[7]

2.1 Models of information seeking

Let’s start by considering a high level description of how humans use information retrieval systems to fulfill *information needs*. The term *information seeking* is close in meaning to *search*, but emphasizes that this is a broader human activity and may occur over a long period of time.[8]

Although it is difficult to comprehensively model user behavior, examining some of the proposed models may be helpful for framing further discussion.

2.1.1 Iterative refinement

One simple model proposed in [8] views information seeking as iterative refinement of a query that is terminated when the user’s initial information need is met and consists of the following phases:

1. Recognizing a need for information

2. Accepting the challenge to take action to fulfill the need
3. Formulating the problem
4. Expressing the information need in a search system
5. Examination of the results
6. Reformulation of the problem and its expression
7. Use of the results

2.1.2 Berry-picking

Another view is provided by the berry-picking model[9] which acknowledges the more complex nature of some information needs. Information encountered in a search may shift the user's goals and queries in unexpected directions or create new goals and diverse queries. Moreover, the information need may not be satisfied by the retrieval of a particular set of documents, but rather by many bits of information gathered over consecutive searches and various interactions with search results.

2.1.3 The information foraging theory

The information foraging theory[10] is based on the idea that humans transfer food-finding cognitive mechanisms to exploring, finding and *consuming* information. The theory suggests that search strategies evolve to maximize the ratio of valuable information gained to the cost of searching and reading.[7] Searcher's behavior is modeled as constantly performing a probabilistic cost-benefit analysis for navigation. In other words making judgements on whether to continue reading a particular resource, refine or significantly alter the query, view more search results, etc.

An important concept for UI design associated with this theory is *information scent* — the estimated probability that a navigation choice will be successful.[11] The *navigation proposition*[12] states that in order for a resource to be findable within a navigation structure, there must be sufficient information scent at every step. Search result listings are designed to provide strong indications about what individual results contain.

2.1.4 Query taxonomy and QAC

Information needs and individual queries can be classified in various dimensions including the topic and underlying goal. This can be done either by automated analysis of search query logs, manual classification or questionnaires. The models described above are relevant especially for rather open

information needs exemplified by a task like compiling literature for a scientific paper.

From the point of view of QAC, it is useful to identify the following two types of information needs or queries:

known-item search The user is searching for a particular document or object, that is known in advance, e.g. app, setting, regularly visited web page, old email. In this case, the search or QAC system is merely the most convenient way of looking up the document. QAC in this context has a straight forward purpose of minimizing the time it takes to submit a query or execute a desired action. Also, it satisfies the users' need for feedback, confirming that the query is formulated correctly. The most extreme example are IDEs and other power-user tools.

exploratory search[13] The user does not have a fully defined goal or is searching in an unfamiliar domain. In these situations, QAC guides the user, suggesting sensible queries. In other words, QAC has the potential to present the data set and the search app's functionality in an interactive way.

2.2 Query specification and syntax

Query specification refers to the phase of the information seeking process in which the user expresses their internalized concepts in a query format that the search system can make use of [7]. These formats range from command languages and Boolean queries to keyword search and natural language questions.

Querying multimedia collections or searching by image is also possible but this overview is restricted to textual queries and documents.

2.2.1 Boolean queries

Search engines in the pre-web era often used Boolean queries or highly specialized command languages. Users were trained in the use of these systems and were willing to spend time formulating the queries, especially if charged per query.[7] This was simply because such systems were easier to implement rather than the sophisticated processing available today.

Boolean queries are expressions consisting of words and Boolean operators **AND**, **OR**, **NOT**. A simplified solution is using two prefix operators — **+(plus)** meaning that the search term must be present and **-(minus)** that it must not.

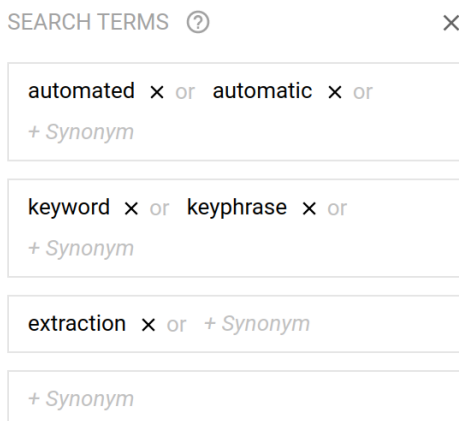
An interesting entry form to generate Boolean queries is used in Google Patents. It is displayed at all times in the left panel of the search results page and provides a certain level of abstraction, which may not require the users

2. SEARCH USER INTERFACES

to understand Boolean queries. The input shown in figure 2.1 generates the query

```
((automated) OR (automatic)) ((keyword) OR (keyphrase)) (extraction)
```

and inserts it into the search box.



SEARCH TERMS ? ×

automated × or automatic × or
+ Synonym

keyword × or keyphrase × or
+ Synonym

extraction × or + Synonym

+ Synonym

Figure 2.1: Form to generate boolean search query in Google Patents

2.2.2 Command languages

In addition to Boolean, command languages make use of other operators to specify options such as fields to search, stemming activation, prefix search or wildcard search.

One example still relevant today is the *Lucene Query Syntax* used in Solr's default *Standard Query Parser*, although it is not necessarily meant to be used by end users. For example the query

```
(title:foo OR title:bar)^2 OR (abstract:"foo bar"~4)
```

matches documents that contain either **foo** or **bar** in their title and also documents that contain the two words in the abstract in the given order within distance 4 of each other. At the same time it gives a boost factor of 2 to the first term's contribution to the relevancy score.[14]

Google search still supports Boolean queries and tens of other operators to alter search results. For example **site:** restricts results to a specific site and **intitle:** searches in web page titles only. However these are rather hidden features that are not expected to be used often or by many users. There is an "Advanced search" entry form (figure 2.2) for some of these functions, that translates them to their textual equivalent.

Advanced Search

Find pages with...	To do this in the search box.
all these words: <input style="width: 80%;" type="text"/>	Type the important words: tri-colour rat terrier
this exact word or phrase: <input style="width: 80%;" type="text"/>	Put exact words in quotes: "rat terrier"
any of these words: <input style="width: 80%;" type="text"/>	Type OR between all the words you want: miniature OR standard
none of these words: <input style="width: 80%;" type="text"/>	Put a minus sign just before words that you don't want: -rodent, -"Jack Russell"
numbers ranging from: <input style="width: 30%;" type="text"/> to <input style="width: 30%;" type="text"/>	Put two full stops between the numbers and add a unit of measurement: 10..35 kg, £300..£500, 2010..2011

Figure 2.2: A part of the current Google “Advanced search” form

2.2.3 Modern keyword search

Keyword search is the dominant query method used today. The query consists of keywords or phrases relevant to the user’s information need. These may be single words, compound nouns, several keywords, or more complicated phrases containing prepositions, conjunctions, etc.

There is an important trade-off between performing clever automated actions and user control. Without sufficient quality of results, automated query transformations may lead to frustrating experiences. The ranking algorithm is completely opaque to users. “ANDing” is problematic when the result set is empty or too small. “ORing” based on statistical functions such as TF-IDF may lead to confusing ranking. For example a document containing a rare term may be ranked high even if it matches no other query terms.[15] Another famous example is the query **to be or not to be** which yields no results in many search engines because of stop word removal.

In Google search, there is a *verbatim* option that disables stemming, synonym expansion, stopword removal, spellchecking and other query corrections. It requires all search terms to be present and omits results for related queries or concepts.

Techniques such as website popularity ranking, boosting occurrences of terms in close proximity, identifying entities or collocations and good quality search result snippets (fragments of matched text) proved to be effective solutions.[16] Especially in web search where the quality of the first few results is key and recall is not very important. For example 80% of search result clicks are on one of the first 4 results.

Research shows that first-time searchers often intuitively start by using natural language questions. It takes time to learn that keyword search is an iterative process and that effort can be made to improve the query and increase the likelihood of a successful search. In recent years, search engines are increasingly capable of intelligent query understanding and handling natural language questions. This trend was summarized well by a Google search engineer in 2007:

Search over the last few years has moved from “Give me what I typed” to “Give me what I want” [17]

Some of these techniques and algorithms will be described in more detail in later chapters.

2.2.4 The search box

In the end, the query entry interface often boils down to a search box, which has become one of the most recognizable UI elements. It should be prominently displayed on every page, where users expect to find it (top-right or top-center area) . The full input field should be displayed rather than initially being hidden behind the magnifying glass icon.

The size of the input field has been shown to influence the average query length[18]. Particularly multi-line boxes encourage longer text. A popular practice is placing a grayed-out placeholder text in the search box that disappears when clicked and contains a hint — either an instruction like **Search e-mail** or example queries, e.g. **code breaking, Alan Turing, enigma movie 2014**. A related practice is providing a selection of featured or trending searches in the form of hyperlinks under the search box.

An integral feature of the search box for aiding the query specification process is of course QAC.

2.3 Current web search engines

With the emergence of the Web, search became a technology widely adopted by the masses. Before that, information retrieval systems were restricted to narrow groups of highly trained users such as librarians, lawyers, journalists and scientific researchers. These systems were usually limited to meta data and searching was often followed by the physical retrieval of a copy of the resource.

At its core, the UI has remained unchanged since the first web search engines. There’s a search parameter entry form at the top of the page and a simple vertical list of search results.[7] This is because web search engines need to cater to an extremely wide range of users and information needs. Search is a mentally intensive process and usually only serves to support a larger task, rather than being the goal itself.

A 2005 study[19] found that even this simple design posed challenges for some older adults. Many participants struggled to understand the basic concept of keyword specification. Further simplifying the search results list has been found to substantially improve usability for this demographic.

The search results are contained in a narrow column on the left side of the search engine results page(**SERP**). This approach follows the philosophy

than narrower columns of text are easier to read and it leaves space for other content such as advertisements on the right.

2.3.1 Google search

Google is the dominant web search engine, commanding a global market share of around 90%.[20] 63 000 searches are processed every second with the average user performing 3-4 daily. 15% of these are new queries, that have not been seen before.[21] "Googling" has become a verb synonymous with web searching. Seznam, a local competitor, is maintaining a market share as high as 25-30% on desktops in Czech republic, which is quite rare, although it is on a steady decline from 50% in 2014.[22]

In recent years, Google has been incorporating new features into its SERPs. Dynamic content is displayed in between the standard results and in another column on the right. In 2012, *Knowledge panels* were introduced and now appear for roughly one third of queries.[23] This is a form of *semantic search* based on the *Knowledge Graph*, Google's knowledge base with information retrieved from sources such as Wikipedia. These have been criticized for incomplete source attribution and particular cases of being inaccurate.[24]

Another feature is a question answering system presented in the form of *Featured snippets* or *People also ask* cards. This blurs the lines between search and personal assistant apps. Users are now comfortable querying Google with simple questions in natural language, especially if confirmed by a QAC suggestion.

More broadly this is attempting to resolve the user's query directly on the SERP eliminating the need to be redirected to an external resource. According to [25], so called *zero-click searches* account for more than 50% of Google searches overall and more than 61% on mobile as of June 2019 in the US. This is met with disapproval from SEO marketers although for example reduced traffic because of Knowledge panels may be positive for Wikipedia.[26]

Because of their popularity and significance, researchers have attempted to study web search query logs to perform *contemporaneous forecasting*. For example attempting to predict near-term values of economic indicators in [27] and detecting influenza epidemics in [28]. Google publishes search statistics in a limited form on the website Google Trends.

Other components of the Google SERP include:

- content panels like maps, weather, sports results, dictionary definitions, population statistics charts, image/video carousels and many more
- widgets such as calculator, currency converter, timer/stopwatch
- search box to perform site search within a search result and *site links* — a selection of additional pages organized under one search result
- related searches sometimes also in the context of individual search results

2.4 Faceted search

The idea behind faceted search or *faceting* is to combine navigation with search, displaying hierarchical classifications of documents. Unlike standard keyword specification where the user needs to explicitly think up the search parameter, faceting provides explicit drill-down options. One eye tracking study[29] found that participants spent 50% of the time looking at facets, and that therefore they play a crucial role in exploratory search.

Facets allow filtering of the search results set based on various dimensions or attributes of the documents. This is used primarily in search applications operating on narrow collections of documents with structured fields. The classic example are e-commerce sites such as alza.cz 2.3

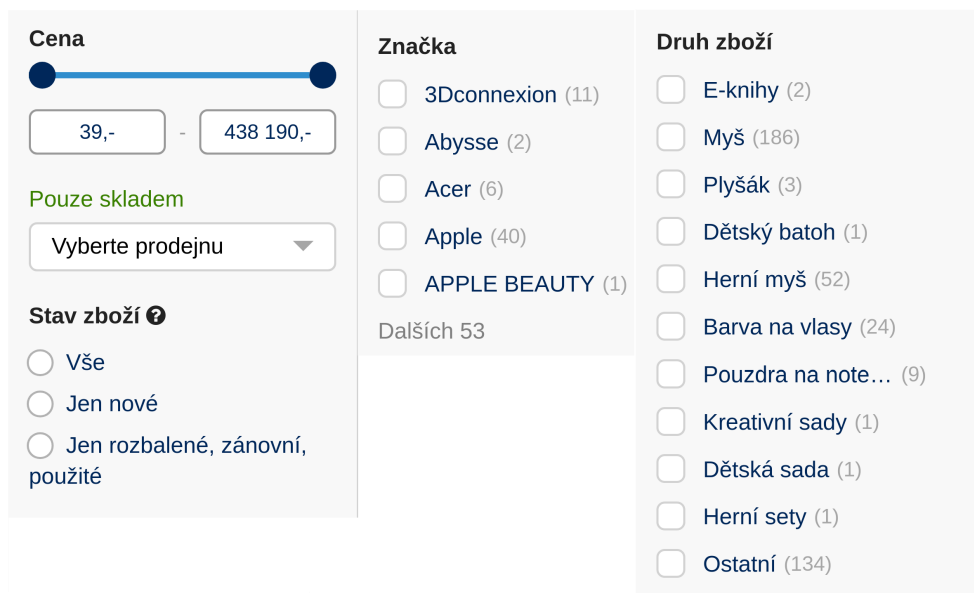


Figure 2.3: Selection of facets for the computer mice on alza.cz

In addition to filtering, facets serve an informational or analytical function by displaying *facet counts* in brackets — the number of documents in that category — which provide indications about the size of various result sets.

After some facet options are selected, the counts of other options within that facet represent the number of documents that would be added to the result set if selected. This behavior is referred to as *multi-select faceting*. Some other facet design patterns include hierarchical options, ordering by count, counts displayed in a histogram, and options containing images such as brand logos.

Varying sets of facets can be dynamically chosen and displayed for different queries. Facet options may also be based on labels automatically generated from unstructured text, e.g. indexed search terms or extracted keywords/-

topics. Carrot2[30] is an *Open Source Search Results Clustering Engine* that performs clustering online on the top search results and provides labels for these clusters in the form of a facet.

Another element related to navigation is selection of search scope. This is often achieved by a drop down menu or radio buttons close to the search box with the default being searching all collections. However, users often overlook this[7] and therefore it is good practice to never require the user to make a navigation choice before executing the query, but rather support it in the SERP.

2.4.1 Nested facets and dashboards

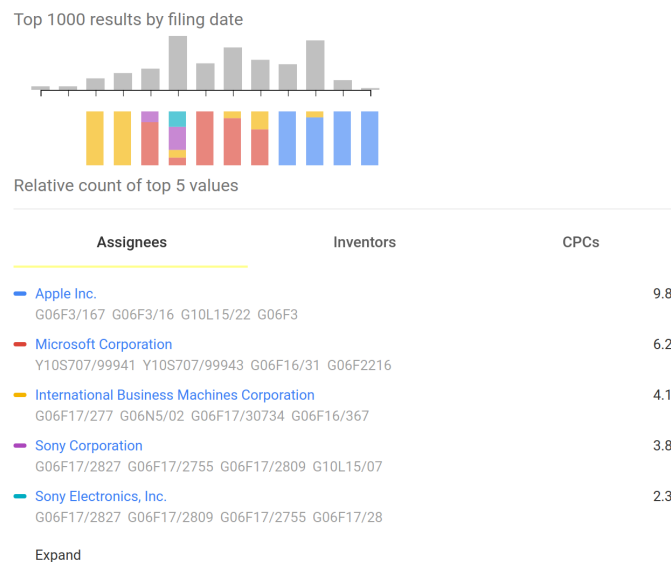


Figure 2.4: Example of a complex facet in Google Patents

The meaning and implementation of facets can be generalized to encompass any statistics or *aggregations* on the set of search results. In addition to the typical counts, these can be other functions such as sum, average, minimum, maximum, etc. Additionally, Solr and similar databases offer even more complex functionality — nested facets (not to be confused with hierarchical facets). This means that further aggregations can be computed on sets of documents yielded by the parent facet.

This may correspond to a 2D chart on the set of search results. Such visualizations can be implemented directly in search to supplement search results such as in Google Patents (figure 2.4). In this case, the filtering and analytical function of the facet is separated into two distinct elements — filtering by patent assignee can be done using a select box in the left panel

and statistics of assignees on top 1000 results are shown in a more complex visualization on the right.

This kind of system for filtering a data set and computing statistics on results can even be the primary feature in some applications such as business analytics dashboards, where a search results listing is not meaningful. This has become a popular use case for Elasticsearch (alternative to Solr also based on Lucene) paired with Kibana — essentially a GUI that exposes filtering functionality and enables easy creation of visualizations powered by faceting or aggregations.

2.5 QAC design patterns

QAC search boxes are an evolved UI element that is mature and many of its aspects are standardized. The Baymard Institute conducts UX analysis of e-commerce sites and provides an extensive evaluation of 153 QAC implementations.[31] A 2019 benchmark claims that 96% of major e-commerce sites include QAC functionality in their search. Good practices identified by this study and others will be introduced in this section.

2.5.1 Keyword query suggestions

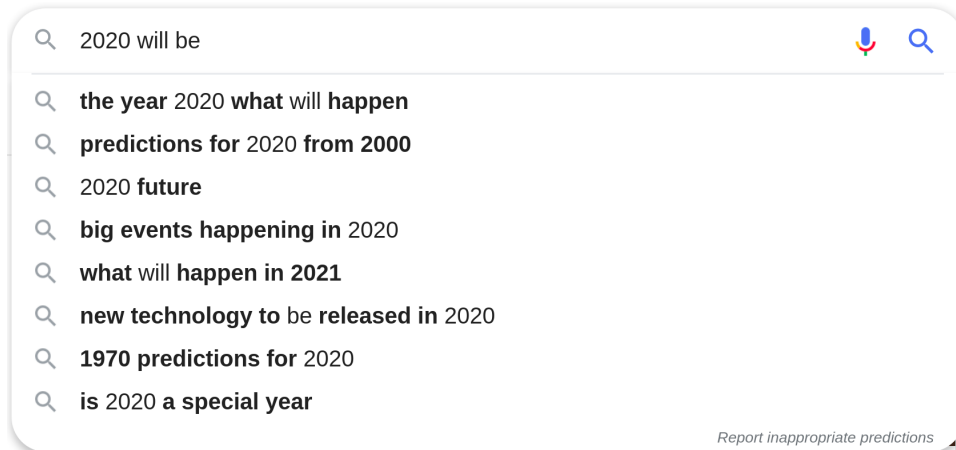


Figure 2.5: Example of advanced Google QAC suggestions

Our primary focus is on keyword query suggestions (in later text I simply use *QAC suggestions*). These are keyword queries recommended to the user by the search system. Such recommendations may be included in the SERP to aid *query reformulation* in the form of “Related searches” and *spellchecking* (showing potential spelling errors). However, our interest is in QAC — providing query suggestions in a list under the search box, updating as the user types.

Typically, suggestions are retrieved based on prefix match, therefore directly representing what the user might intend to type. Other schemes may be used such as including phrases that contain a fully entered word in the middle. A major feature is spellchecking — tolerance of misspelled words — which can be implemented in several ways. As we see in the following statement, the impact of QAC suggestions is not straightforward.

Our search usability testing reveals that having autocomplete suggestions appear as users type their search query isn't about speeding up the user's typing process. The true value comes from how autocomplete suggestions can assist and guide users toward submitting better search queries. When autocomplete suggestions are done well they inspire users about the types of queries to use, teach correct domain terminology, help avoid typos, and assist users in selecting the right search scope.[31]

While QAC has the potential to increase quality of submitted queries, it can also lead to incorrect search paths.[32] Therefore the quality must be sufficient to compensate for the distraction. Moreover, as a general principle, requiring users to make relevancy judgements should be avoided. For example similar queries like **paracetamol** and **paracetamol (acetaminophen)** should not be available. Quality should be prioritized over quantity for example by utilizing more aggressive normalization. Most solutions are based on lowercase suggestions consisting of alphanumeric characters.

For obvious reasons, it is undesirable to offer suggestions that yield empty result sets, however this may be unavoidable in combination with faceted search. Airbnb search remedies this by showing a button **Remove all filters and see X more results**.

2.5.2 Direct entity suggestions

In some applications, additional types of suggestion are appropriate. These can be either direct document suggestions or filtering by facets(entities). For example, search in an e-mail application may suggest individual messages directly and filtering by sender, date or label.

Crucially, this implements tolerance for search terms being entered into the wrong field. A solution with dedicated input fields for author or document identifier runs the risk of frustrating users who expect this function from the main search box. An alternative solution that simply incorporates these attributes into the main full-text search leads to unexpected results and reduced precision. For example, should a search for author name in a library return books in which the name is mentioned or only the ones he authored?

2.5.3 Search history suggestions

One study of Yahoo query logs found that 40% of queries were for re-finding, i.e. the user clicked on a result that they had already clicked on in a previous search session.[33]

For these know-item searches, it is valuable to support suggestions from the user's search history. In addition to normal matching, several recent searches may be shown in the default view with no characters typed.

2.5.4 Interactivity and responsiveness

A 1968 paper[34] identified 100ms as the maximum response time required to achieve the illusion of *direct manipulation*. In other words, making the user feel like content on the screen is changing directly as a result of their actions, rather than the action being performed by the computer. This is a key requirement for QAC, because relatively large portions of the screen are being rendered in response to key strokes. Large delays and inconsistency can be distracting and have negative impact on query specification.

One approach is to start showing suggestions only after several characters have been typed or to insert delays preventing triggering in case of quick successive key presses. However, if sufficiently low and consistent response time is achieved, instant refreshing with no loading animations seems to be preferred and has become standard. While providing suggestions already after the first character may be unlikely to predict the query, it makes it clear that the search box in fact has QAC functionality and prevents potential confusion.

Keyboard navigation using arrow keys is expected in addition to mouse clicks and should allow looping between the first and last suggestions. The suggestion that is currently selected with arrow keys can be automatically inserted into search box, enabling further typing (e.g. entering a space to get suggestions for another word in a phrase). A good practice is to include an "X button" to empty the search box on mobile devices and allowing the Escape key to close the QAC box on desktop.

2.5.5 Presentation of suggestions

Presenting a rather small number of suggestions is recommended so that they are not overwhelming. They should certainly fit into view on desktop rather than requiring a scroll bar.

An important guideline is to provide visual distinctions between various types of suggestions such as the ones identified above. The basic solution is grouping by type, although some applications manage to mix types into a single list and distinguish them by icons or other labels. The magnifying glass icon is sometimes used to indicate query suggestions. Less common designs use large QAC boxes with multiple columns or for example show large thumbnails for matched products.

“Inverted highlighting” is often used for keyword query suggestions with the rationale summarized in the following statement.

A common practice is to highlight the part that matches the user’s input; however when offering search suggestions, the inverse is true: it is important to highlight the part that is being suggested. This approach visually aids the user in distinguishing between suggestions as it highlights the differences.[35]

Other practices include restricting suggestions to a specific search scope, buttons for alternative actions and displaying the number of found documents for each suggestion.

2.5.6 Instant search

Instant search is a design that goes a step beyond QAC. Not only suggestions, but search results themselves are updated during typing. This works well when the searched collection is relatively small and prefix matching is used. For example searching for songs, albums or playlists in Spotify. Another reason to favor this design is when the input method is especially slow like arrow buttons on TV remote.

Prefix matching is most likely not suitable for full-text search. But there is the possibility to periodically refresh search results based on the top QAC suggestion at that moment. This was implemented in Google search in 2010. After the rise of mobile devices which have insufficient screen space for this approach, *Google Instant* was discontinued in 2017 with the aim of unifying the search experiences and avoiding maintaining two different systems.[36]

Theory

In the previous chapter, I explored the problem of QAC from the UI perspective. This chapter provides an overview of the methods and technologies for implementing the QAC back end — the service that returns the QAC suggestions based on text entered into the search box.

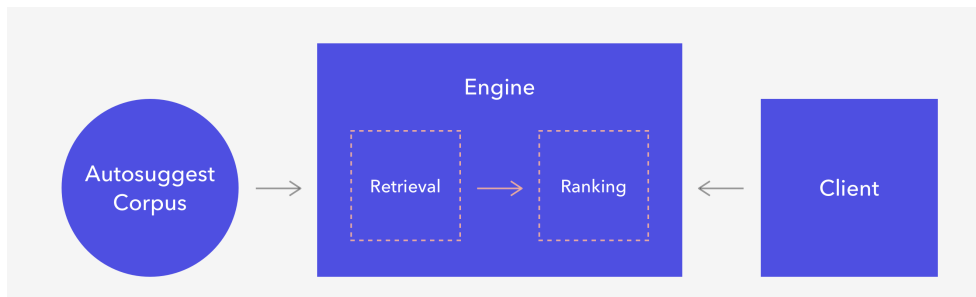


Figure 3.1: Basic structure of a QAC system

Source: [37]

3.1 Building the corpus of QAC suggestions

There are two commonly used sources for QAC suggestions: search query logs and the corpus of documents being searched.

3.1.1 From search query logs

According to a survey [38], mining search query logs is the basis of most researched QAC solutions. It enables major search engines to provide dynamic time and location sensitive suggestions.

Filtering out undesirable queries poses a major challenge. Queries can be submitted by accident and contain misspellings, questions in natural language, long passages of text or otherwise produce undesirable suggestions.

More importantly, there is risk of breaching user privacy by publishing suggestions containing sensitive information. Furthermore, the system has to contend with manipulation. Attackers may attempt to game QAC by submitting queries containing advertisements or malware, a new form of *black hat SEO*. [39]

In [39], a systematic analysis using NLP techniques is performed on query logs of major search engines. One of the findings is that there are at least tens of companies offering autocomplete manipulation services, mainly employing one of two strategies - *search log pollution* and *search result pollution*:

Particularly, we found that at least 0.48% of the Google autocomplete results are polluted. The security implications of the attack are significant. For example, our study shows that at least 20% of these manipulation cases are used for underground advertising, promoting content as gambling and even malware. [39]

3.1.2 From searched documents

Data in structured text fields can be used directly to supply keyword query suggestions. In certain types of documents such as scientific papers, author-assigned keywords are available. These have the potential to be a major source but come with certain challenges:

Authors usually do not assign keywords manually unless they are instructed to do so because it is annoying and time consuming. Additionally, an author will probably assign keywords which are most likely to draw attention to the document. Therefore, it is uncertain whether author-assigned keywords correspond precisely to the content of the document. [40]

The option that I mainly focus on — using automatic keyphrase extraction to generate suggestions from the searched documents — is introduced later the chapter.

The disadvantage of using documents is that they do not represent the users' needs nearly as well as the search query log, although it provides good support exploration. Moreover, query logs are not available in certain cases such as the initial launch of an application or for privacy and legality reasons.

3.2 Retrieving suggestions

The corpus of suggestions is normally computed offline and stored in a data structure for efficient retrieval. Processing a QAC query, i.e. returning a list of suggestions given the user input has two aspects:[38]

Matching Construct a list of candidate suggestions matching the user input.

Ranking Rank the candidates by relevance and return the top n.

3.2.1 Matching

The suggestions are usually matched based on prefix or other substrings such as full words. For simplicity, the following text only mentions prefixes.

3.2.1.1 Brute force

Let's start by considering the trivial **brute force** approach: iterate over the corpus of suggestions and select those with matching prefix. This has time complexity of $O(n * p)$, where n is the corpus size and p is the prefix length — clearly unacceptable for large data sets. An improvement can be achieved by using **binary search** - maintaining the suggestions in an ordered list. For example an SQL query to achieve this could look something like:

```
SELECT label FROM suggestions WHERE text LIKE 'prefix%';
```

3.2.1.2 Inverted index based on n-grams

It is important to mention that the indexing performance is much less critical, which can be leveraged to build data structures that offer better querying performance. This leads to the idea of computing all possible prefixes (*Edge character n-grams*) for each suggestion during indexing and building a map that yields the list of suggestions for a given prefix.

The inverted index, a key data structure powering full-text search is based on the same idea. In standard full-text search, the keys in the index are search terms, e.g. normalized stemmed words. In the case of QAC, suggestions are in the role of documents and all possible prefixes — n-grams rather than full words — are acting as terms.

This solution provides fast retrieval at the expense of space. The number of prefixes(terms) in the index is $O(MN)$ where N is the number of suggestions and M is the maximum length. For each prefix, a list of suggestions is kept, therefore the worst case space complexity is $O(MN^2)$. M can be limited by simply restricting support to a fixed length of prefix like 20 characters. Moreover, only a limited number of suggestions such as 10 are ever retrieved for a prefix. Therefore, based on the ranking solution, the number of suggestions

indexed for each prefix could be limited to a top selection based on static score.

In a simplified scenario, where only matching suggestions by full words is required, an improvement can be achieved by using two indexes. The first index mapping prefixes to all possible words and the second mapping words to suggestions.

3.2.1.3 Prefix trees

Another option are specialized data structures designed for fast retrieval of strings based on prefix — prefix trees or *tries* (a type of *search tree*). This solution was first proposed by René de la Briandais in 1959.[41] The inverted index approach requires all index terms to be treated individually, e.g. in a sorted array.[42] Tries make use of the fact that prefixes shared by multiple strings need only be stored once and only the remaining suffixes are stored individually.

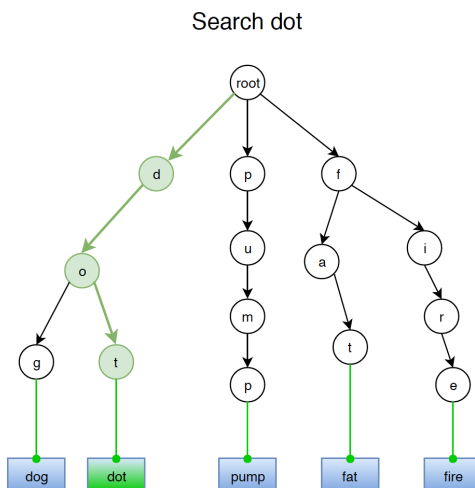


Figure 3.2: Example of trie schema

Source: [43]

The tree is interpreted as follows:

- The root represents the start of every suggestion, i.e. no characters.
- Edges represent characters.
- Non-terminal nodes represent prefixes.

- Leaves designate complete suggestions and the path from root to leaf is the sequence of characters of the suggestion.
- The k -th level in the tree contains the k -th character of all suggestions.

A value such as the suggestion ranking may be stored in the nodes. Therefore tries can be also used more broadly to implement an associative array where keys are strings.

Searching for a prefix is performed by traversing the tree from the root, at every level choosing the child corresponding to the next character in the prefix. After that, depth-first search is performed to find all suggestions in the subtree below that prefix.

This final step is costly for short prefixes. More advanced data structures similar to trie that mitigate this disadvantage are *Deterministic Finite Automata (DFAs)* and *Finite State Transducers (FSTs)*. These kinds of data structures are likely the basis of state-of-the-art commercial solutions.[38]

3.2.2 Ranking

From the potentially large number of suggestions matched by prefix, a ranking based on some criteria needs to be produced and several top results returned to the user. Existing solutions can be broadly classified into two categories: *heuristic models* and *learning-based models*. [38]

3.2.2.1 Heuristic models

Heuristic models compute a static score for each suggestion, attempting to represent the likelihood the user is intending to issue that query. Although the functionality of QAC is not limited to prediction, a ranking function can be viewed as estimating the probability $P(\text{suggestion}|\text{prefix})$.

Popularity in the query logs directly estimates this probability with successful results in practice, and plays a key role in web search engines. A straight forward *user-centered model* is *most popular completion (MPC)*. It simply assigns a static ranking to each suggestion, corresponding to its relative frequency in the query log.[38]

However it should be noted that simply considering all executed queries may be misleading. It fails to account for unsuccessful queries, where the information need was not met. Therefore a better signal might be giving more weight to successful searches. For example, an e-commerce site might go a step further and consider conversions such as purchase of a product.

As in building the corpus, document popularity might be used as substitute, but runs the risk of conflating supply and demand. The users may not be searching for a phrase even if it is popular in the corpus.

In web search, query popularity changes over time as new content such as news and other trends emerge. Additionally, certain queries are driven by

periodical patterns, e.g. **weight loss**, **movie tickets** or **sunset time**. This may be addressed by limiting the analyzed search query log to a fixed past period (*a time-sensitive model*). More sophisticated approaches leveraging *time-series analysis* attempt to predict future trends and handle the trade-off between supporting time-sensitive and consistently popular searches.

3.2.2.2 Learning to rank

QAC ranking and search ranking in general can be formulated as a supervised learning problem — often referred to as *learning to rank (LTR)*. In contrast to heuristic approaches, this method aims to collect many features that represent the characteristics of a suggestion.

The training data for such models can be harvested from query logs and consist of pairs (prefix, query) along with a label representing the relevancy of the query for the given prefix. In QAC ranking, it is common to use a binary label, i.e. “suggestion chosen” or “not chosen”. [38] The aim is to train a model that produces a ranking score between 0 and 1. An interesting possibility is using rankings from outside search engines as additional examples.

The prefix and query are represented as feature vectors. Commonly used features include query log popularity, document popularity, length and user interactions such as typing speed and clicking patterns harvested from high resolution query logs.

It is not surprising that search behavior and therefore appropriate suggestions vary dramatically across regions, demographics and individual users. [44] Therefore, user features such as age and past search behavior are incorporated to implement personalization.

An important technique to enable the use of such models is *query re-ranking* (figure 3.3). Running an expensive LTR model on the full list of suggestions for a short prefix is too expensive. Therefore, the ranking must be done in two phases. First, selecting a list of candidates based on a cheap ranking such as ordering by static score and consequently re-ranking the selected candidates using the expensive model. The first phase is normally done as part of the matching iteration.

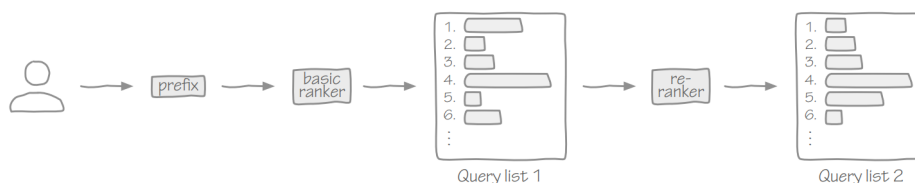


Figure 3.3: Query re-ranking pipeline

Source: [38]

3.3 Automatic keyphrase extraction (AKE)

Automatic keyphrase extraction (**AKE**) is a task concerned with the selection of representative phrases from unstructured text of a document. The term *keyphrase*, rather than *keyword*, expressly covers multi word expressions. I will use the two terms interchangeably. A linguist may define a keyword as:

A word, the frequency of which is statistically significantly higher in the text than in the reference corpus.[45] (author's translation)

In a supervised approach, AKE is treated as a classification problem. A binary classifier is trained to determine, whether a candidate phrase is a keyphrase in a given document. This requires careful preparation of training data.[46]

Unsupervised methods seem to be a more successful trend.[47] In the remainder of this chapter, I introduce several such algorithms.

In general, the functioning of these algorithms can be decomposed into two phases. The selection of potential keyphrases from the text followed by scoring based on various statistics and the selection of top candidates based on threshold or limit.

3.3.1 Text pre-processing

Because of the complexity of human languages, natural language processing(**NLP**) is notoriously difficult for computers. However, various advances make it possible for NLP to handle an increasing number of tasks.

Some AKE algorithms require a certain amount of text pre-processing — transforming text into a more structured format that enables further computation. In this section, I describe a small selection of relevant NLP concepts.

3.3.1.1 Stopwords

Stopwords are the most common words in a language such as prepositions and conjunctions. They provide little contribution to the meaning of a body of text and are discarded in many NLP tasks. Most of these are *closed-class words* meaning that new ones rarely enter the language.[7]

There is not a single decisive *stopword list*. The meaning of a stopword is not well defined and particular lists are appropriate for various tasks. The information stopwords carry should not be underestimated and stopword removal has to be done carefully. For example, when performing AKE, a significant number of potential keyphrases would be lost if stopwords were simply thrown away, e.g. **internet of things** or **man on the moon**.

3.3.1.2 Stemming

One major challenge of most languages is that small character variations in words are not relevant in some scenarios. For example, a document titled **cheapest plane tickets** should probably be retrieved for the full-text query **cheap plane ticket**. Similarly when identifying frequent phrases in a document, we would like to be able to identify these slight variations as the same phrase.

Stemming is the basic solution to normalize word forms by simple string based editing such as removing particular suffixes based on a list of rules. The result of stemming (a stem) has potentially no meaning. For example the words **organizing** and **organziation** are both stemmed to **organ** in Solr's default text analysis pipeline. Using stemming in full-text search is a very common practice. It increases recall and lowers precision.

Some stemmers are also available for Czech. Solr comes with an implementation of a light (rather than aggressive) Czech stemmer based on the paper [48] and a Python port is also available.

3.3.1.3 Morphology

Czech is an *inflective language*, which means words change form very frequently based on their meaning and grammatical function in a sentence. This is in contrast to a language like English where word order is crucial and there are very few inflection rules.

Morphology is the study of the relationship of diverse word forms. Morphological analysis provides several essential NLP functions including:

lemmatization Normalizing words into actual meaningful words (unlike stemming)

POS tagging The process of classifying a word form based on linguistic categories including POS (*part of speech*) and more fine-grained properties. The output of this process is called a *POS tag* and is usually in the form of a string of characters, each corresponding to a category. These may include POS, subPOS, gender, number, person, tense voice, etc.

morphological generation This is a generalization or an inverse of lemmatization. Given a lemma and a POS tag, the word form with the desired categories is generated.

Morphodita and related software is developed at the Faculty of Mathematics and Physics in conjunction with the Institute of Formal and Applied Linguistics of Charles University in Prague. They provide state-of-the art morphological analysis for inflective languages including Czech. This is enabled by processing a large annotated corpora containing word forms and their respective lemmas and POS tags. Language structures are detected

without providing explicit linguistic knowledge. The corpora is condensed into a *morphological dictionary* which provides an efficient implementation of the functions described above.[49]

3.3.2 TF-IDF

Unsupervised AKE methods aim to take advantage of the underlying structure of the text. To start with, it is likely that the author used keyphrases in the text repeatedly as he advanced his argument. This suggests a trivial approach — to simply select the words or n-grams most frequently appearing in the document. This basic approach runs into the problem that text contains common words, perhaps specific for the given domain, that do not represent it well even if they appear repeatedly.

TF-IDF is an important statistic, that represents the significance of a given word in a document, taking into account the context of the whole document corpus. With the increasing number of occurrences in the document, the term frequency TF and consequently $TFIDF$ increases. At the same time, with the increasing number of occurrences in the whole corpus, the document frequency DF increases and therefore $TFIDF$ decreases.

Single words or n-grams with the highest TF-IDF in a document can be identified and extracted as keyphrases. One disadvantage of this approach is that it is dependent on the whole document collection or the document frequencies are based on a larger corpus.

3.3.3 RAKE

RAKE(rapid AKE)[50] is an algorithm based on the idea that keyphrases are chunks of text in between stopwords and punctuation. Unlike in TF-IDF where text normalization is performed first, RAKE runs on raw text.

First, candidate keyphrases are selected by splitting the text at stopwords and punctuation. After that, individual words from the keyphrases are considered. The score for each word is calculated based on degree(number of unique words, with which it appears in a phrase) and frequency (total number of occurrences). Finally, the score of a phrase is computed as the sum of its word scores.

An additional mechanism attempts to detect keyphrases containing stopwords. If two candidate keyphrases appear in the document separated by the same stopword in that order at least twice, they form a new candidate.

Implementation

4.1 Software requirements

In this section, I outline requirements for the Starfos QAC feature. With the exception of recent searches and the administration interface, all of these were implemented to some degree as part of the thesis.

4.1.1 Establishing goals

We aim to support both known-item and exploratory searches. The two categories loosely correspond to two groups of users we can identify in Starfos.

The first group consists of TA ČR's employees and other professionals in the field of R&D funding. These users are familiar with the domain and may use Starfos to support their daily work tasks. Use cases include searching for known keywords, organizations, filtering by enumerations and looking up by document identifiers. Experienced users may appreciate features such as matching documents by code prefix or history of recent searches.

The second group are users less familiar with the domain, who may be discovering the website for the first time and are unaware of the available functionality. For them, we want our QAC product to serve as a possible introduction to the website. We may lean towards presenting a relatively large number of suggestions in multiple categories. Another proposal is to implement default suggestions, such as hand crafted or popular searches, that are displayed after the first click in the search box. In addition, this may enable us to address certain shortcomings of the current UI, such as filters that are difficult to find.

As a general principle in UI design, it is easy to make false assumptions about user behavior. Starfos query logs could be analyzed to gain reliable insights.

4.1.2 Suggestions

Implement suggestions in the context of the two search collections — projects and results. In addition to keyword query suggestions the following types should be included:

Filtering(facet) suggestions Take advantage of structured attributes such as result author, project participant or provider. Users expect to be able to enter these in the main search box. In IS VaVaI, a large number of structured metadata attributes is available and exposed in Starfos as facets. Important use cases are finding projects of an organization or results of an author. Precise search based on filtering is expected.

Direct document suggestions Include a small number of these to encourage exploration. Match in document title and use standard highlighting(not inverted). Rather than performing a search, directly redirect to document detail page.

Recent searches and visited pages Store several recent searches and visited detail pages in cookies. Insert recent searches that match entered prefix to the top of the suggestion list. Additionally, display both in the default view with no characters typed. Consider legal implications and ideally provide a setting to opt out of the functionality.

For facet and document suggestions, abbreviations should be supported where possible. Projects and results should be matched by code prefix. Facet enumerations should be matched by identifier. Abbreviations of Czech universities and other organizations are being collected and added to the IS VaVaI data. Organization suggestions should allow both filtering projects/results and redirecting to organization detail page.

4.1.3 User interface

We intend to follow most of the guidelines defined in the previous chapters with the focus on desktop. In particular:

- Instant QAC already after the first entered character with no loading animations.
- Support both for grouping and icons to visually distinguish types of suggestions
- The magnifying glass icon for keyword query suggestions.
- To enhance readability, we use inverted highlighting. Additionally, we align the first character of suggestions' labels with the search box text and use the same font. Implement the option for suggestions to include a secondary label displayed in smaller font.

- Interactivity features and a cautious response time goal of 200 ms maximum.

4.1.4 Technical requirements

Automatic corpus update The underlying ISVAV database is updated in regular intervals such as 1-4 weeks. Therefore a solution to update the suggestions is required. It is not required to support updates on-line and rebuilding the entire index is acceptable.

Administration interface and logging Even with automated processing, an interface to curate the corpus by hand is desirable. Storing suggestions in the database and using the Django admin interface will give control to a non-developer administrator. In particular, the ability to delete a suggestion is needed, especially if automated extraction from the search query log or documents were implemented. Even if the solution is limited to author-assigned keywords, the option to directly eliminate poor suggestions that are encountered for common prefixes would be beneficial.

4.2 Author-assigned keywords

Our dataset contains author-assigned(**AA**) keywords in English. This provides a great basis for automated extraction. AA keywords could allow us to evaluate AKE algorithms and even use supervised methods. In particular, we could perform a typical evaluation of precision and recall.

On the other hand, matching the quality of AA keywords of scientific publications and projects, by extracting from their relatively short abstracts seems to be extremely difficult, if at all possible. In fact, an English corpus built from AA keywords proved to be a satisfactory solution and will likely make its way into production. There seems to be little motivation to invest more time into the development of AKE.

The AA keywords seem especially fitting for the use case of scientific results and project where keywords are a well defined concept. And at the same time, Starfos has no ambition (and no expectation by the users) of handling queries that are much more sophisticated than straight-forward keywords.

The requirement of Czech QAC will possibly be dropped and English search will be promoted as highly preferred. There is another reason to support this decision — only 40% of RIV results include a Czech abstracts while the English one is available for 90%.

4.2.1 Parsing

Keywords available in IS VaVaI have been collected from various providers over a long period of time. They are not available as a structured list but rather contained in a single string. The keyword strings contained in the data set use inconsistent formatting. Most use “;” as a separator, but some use “,” or even “-”. Various aliases are used for blank values, e.g. “*”, “Neuvedeno”, “xxx xxxxx”. Another challenge is posed by strings containing no separators. They represent either a single long phrase, multiple one-word keywords, or even multiple phrases. Our implementation is based on simple logic - checking the total word count and the presence of stop words. Strings containing 2 words are likely to represent a two-word phrase. 7 words are more safely treated as 7 keywords, but if there are stopwords, it is likely a phrase.

The main objective is to supply suggestions for autocomplete. A straightforward algorithm to split and clean the text provides sufficiently accurate results.

Parsing mistakes are unlikely to yield the same wrong keyword with high frequency. Therefore we can rely on the ranking algorithm to ensure they do not appear often.

A more conservative approach would be outputting no keywords in case of low confidence, correcting mistakes by hand or simply discarding keywords occurring with a frequency below a certain threshold.

In addition to forming the corpus for QAC, these keywords are listed in the document detail page with hyperlinks redirecting to search.

4.2.2 Normalization

The parsing we performed simply aims to split the keywords by separators but does no additional processing. This is appropriate for displaying in the detail page, exactly as they were inputted by the author.

However, for QAC it is undesirable that they contain mixed case, abbreviations in parentheses, hyphenation and many other inconsistencies. As we established in previous chapters, similar suggestions with for example only small formatting differences would be highly undesirable.

Therefore we use a simple algorithm to normalize these keyword query suggestions. This could potentially be used on extracted keywords in the same way. Namely the algorithm removes parentheses including their contents, removes hyphenation and other punctuation, eliminates excessive white space and makes the suggestions lowercase.

This normalization reduces the number of AA keywords in our corpus from 1148033 to 951834 — a reduction of 20%. The document counts of resulting keywords that are the same are summed up.

Still we haven’t addressed the various word forms of the keywords, especially some of them are in plural. This is not trivial and we will likely use

some POS tagging to detect popular patterns such as adjective followed by noun and apply lemmatization based on these templates. This is an important concept that is used for AKE as well.

4.3 AKE

The corpus of processed AA keywords seems to yield a satisfactory QAC experience. The suggestions from AKE will hardly match this quality. Nevertheless, I pursued AKE for the following reasons:

- Author-assigned keywords are only available in English and Czech suggestions are also of interest.
- If sufficient quality is achieved by AKE, it promises higher consistency than the diverse human assignment.
- Possibility to generate more keywords.
- Potential to be a valuable ranking signal even if the corpus was limited to AA keywords.

Unlike most applications, we are interested in the corpus of all keywords rather than accuracy per document, which makes this task substantially easier. Just as with the AA processing, potential erroneous keywords can likely be eliminated by ranking or explicit threshold.

Fortunately, many AKE implementations are freely available for Python and provide simple basic usage. Although, Czech language support is an extra challenge and for some methods is not available. For example pyTextRank relies on the nlp library spacy for parsing text and identifying linguistic features and Czech is not supported.

In the end we implemented bare bones versions of the following AKE algorithms: basic term frequency, TF-IDF, RAKE, TextRank(English only).

An example result of TF-IDF on English project goals:

A novel ?idiotic spatial? version of the standard peak-interval timing procedure is proposed in order to study the role of temporal processing and interval-timing strategies in spatial navigation. These ?hybrid? tasks are derived from the study of duration discrimination and simultaneous temporal processing in the seconds-to-minutes range in combination with the investigation of flexible navigation and orientation in dynamic and moving world. As a consequence, both the temporal and spatial dimensions of the task involve dynamic cognitive processes. Understanding how these processes are integrated and conducted in parallel will be important for determining whether distortions in one dimension (e.g., time) affect processing in the other dimension (e.g., space). Such time/space integration is considered to be a fundamental property of cognition, but has never been elucidated.

For the document above, our TF-IDF implementation yields the keywords: **temporal processing, interval timing, temporal, spatial, navigation, timing, interval, processing, dimension, world consequence.**

More examples can be found in the included jupyter notebooks.

4.4 Matching and ranking

Solr Suggesters are based on tree data structures I described. Our initial implementation using Solr Suggesters proved to be unreliable for larger collections such as 1M RIV results and there is relatively little available documentation. It supports automatically sourcing suggestions from indexed search terms (*DocumentDictionaryFactory*) or loading from text file (*FileDictionaryFactory*). It may be the best choice for many applications because of its easy setup, and potentially better performance.

However it was not able to fulfill all of our requirements in a satisfying way. Various matching settings are available in the various *Lookup implementations*, however these do not provide the fine grained control achieved by custom Solr field types based on n-grams. It is possible to combine multiple Suggesters at the same time to implement suggestions grouping. However filtering is supported only in a limited way. Filtering is useful in our case for providing varying sets of suggestions in various contexts and search scopes.

The only disadvantages of the classic Solr index based on n-grams is worse performance and more laborious implementation. Fortunately, we found that this approach provides sufficient performance with response times hovering around 100 milliseconds on our local network, with much room left for optimization. Sematex autocomplete[52] is an example of an existing open-source project that is designed in this way. Standard Solr cores are a mature and popular technology, offering an extensive set of features including

- searching over multiple fields with varying text analysis pipelines,
- advanced query syntax,
- various ranking and relevancy tuning options,
- support for *query re-ranking* and *Learning to rank*,
- configurable highlighting,
- filtering,

We use three custom Solr fields for generating the index terms:

no_token_edge At index time, the text analysis pipeline of this field generates prefixes of size 1 to 20. At both index and query time, lowercasing and accent removal is performed.

shingles Generates all possible continuous word n-grams of size 1-3 at index time. This enables matching by full word anywhere in the suggestion.

code This field is used for matching document and facet suggestions based on code or abbreviation. It computes all character n-grams rather than prefix only.

For **ranking**, we use static score representing document popularity simply because it is the easiest solution to bootstrap the system. Unlike in web search engines with high traffic, mining query logs is unlikely to provide a comprehensive solution. At the same time it seems very fitting for our use case where there's high emphasis on exploratory searches. More work needs to be done on developing a more systematic approach including its evaluation.

The current implementation uses a rather simple **Solr query** that matches by any of the 3 mentioned fields and sorts results based on the static score. In addition, it defines highlighting parameters and grouping — the various types of suggestions are return in one query in separate groups of fixed limit.

4.4.1 HTTP web service

The feature is implemented using the HTTP GET method with the following parameters

q The text typed into the search box such as "n", "na", "nan"

language The user selected search language

context Either the selected document collection or current page. This enables varying suggestions sets to be used in different contexts.

4.5 Front end

I was not personally involved in coding the front end implementation. Although I provided major input into the design. The front end is not provided in the enclosed attachment. Only the back end implementation (HTTP service) is included.

Special actions such as redirecting to document detail page and appending filters to the current search payload were implemented. Suggestions grouping is supported and easily configurable, although we might pursue merging at least some of the groups.

4. IMPLEMENTATION

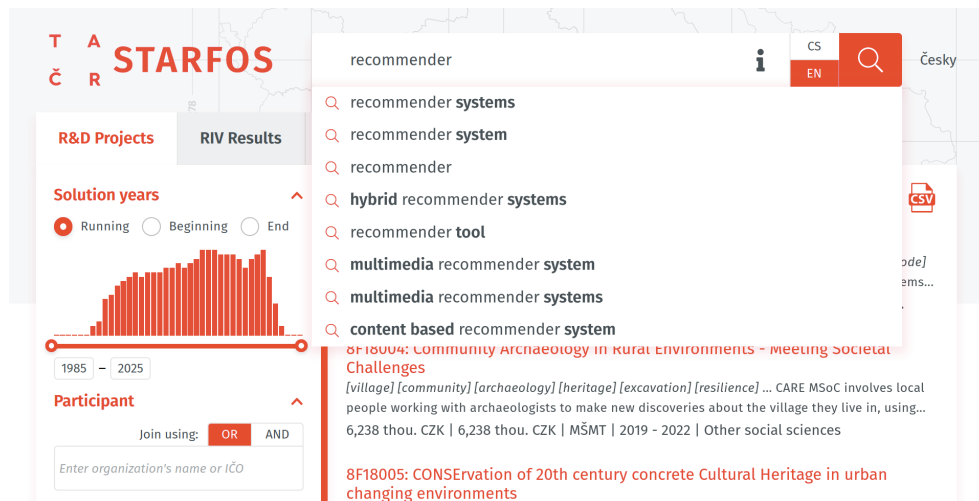


Figure 4.1: The current version of Starfos' QAC in development

auto	infor
Q autobiography	Q information systems
Q automata	Q information and communication technologies
Q autoantibodies	Q information and communication technology
Q automatic speech recognition	Q information technologies
Q autonomic nervous system	Q information sources
Q autoclaved aerated concrete	Q informatika
Q autopsy	Q information extraction
Q autotuning	Q information services
Projekty	Projekty
Q (1A8676)Autologous transplantation of bone marrow cells	Q (1ET200300413)Information technologies for development of continuous s
Q (1ET101210406)Automatic 3D Virtual Model Builder from PI	Q (1ET201450508)Information and communication system for creation and c
Q (1ET400750407)Automatic Acquisition of Virtual Reality Mo	Q (1F44K/055/050)Information overload of traffic system and mental capaci
Filtre	Filtre
Q (ico:00177041)ŠKODA AUTO a.s. (IČO: 00177041) Účastník	Q (ico:60801735)INFORMETAL, zájmové sdružení "v likvidaci" (IČO: 60801735) Účastník
Q (ico:00550264)Autoklub České republiky (IČO: 00550264) Účastník	Q (SAV02004-IS)Informační společnost 1 (SAV02004-IS) Veřejná soutěž
Q (ico:04308697)AUTOCONT a.s. (IČO: 04308697) Účastník	Q (SAV02005-IS)Informační společnost 2 (SAV02005-IS) Veřejná soutěž

Figure 4.2: Additional examples of achieved QAC results based on author-assigned keywords

Conclusion

Implementing QAC proved to be a demanding but rewarding challenge. It spanned a wide range of disciplines and contained sub-problems ranging from UI design to NLP and IR.

All requirements of the thesis were covered to some extent with the exception of evaluating the ranking solution and building the Czech corpus.

A skeleton implementation of several AKE algorithms and some other NLP functions was created. Much more development including laborious cleaning of the data set would be required to produce usable results. A real contribution of this effort is integrating several NLP libraries into our project.

A satisfactory solution for the English QAC corpus was achieved based on author-assigned keywords and will likely be deployed in production in the near future.

AKE, suggestions ranking, search query log analysis and other topics could have been studied in more depth. However, a decomposition of the problem domain was established.

Much effort was put into designing a real QAC product and exploring possibilities for improvement of Starfos search. In my view, the biggest success is the robust and flexible software architecture of the QAC system itself.

Bibliography

- [1] Tam, C.; Wells, D. Evaluating the Benefits of Displaying Word Prediction Lists on a Personal Digital Assistant at the Keyboard Level. *Assistive Technology*, volume 21, no. 3, Sept. 2009: pp. 105–114, ISSN 1040-0435, 1949-3614, doi:10.1080/10400430903175473. Available from: <http://www.tandfonline.com/doi/abs/10.1080/10400430903175473>
- [2] I've Got a Suggestion. Available from: <https://googleblog.blogspot.com/2004/12/ive-got-suggestion.html>
- [3] Hook, J. *Autocomplete: The Book*. Chronicle Books, May 2019, ISBN 978-1-4521-7784-7.
- [4] Veřejně Přístupná Data IS VaVaI. Available from: <https://www.rvvi.cz/>
- [5] Research and Development in Czech Republic. Available from: <https://www.vyzkum.cz/>
- [6] Ulbricht, D.; Elger, K.; et al. Exposing Relational Databases through an Emulated SOLR Search API. *EGU General Assembly Conference Abstracts*, Apr. 2018: p. 14610. Available from: <https://ui.adsabs.harvard.edu/abs/2018EGUGA..2014610U/abstract>
- [7] Hearst, M. *Search User Interfaces*. Cambridge University Press, Sept. 2009, ISBN 978-0-521-11379-3.
- [8] Marchionini, G.; White, R. Find What You Need, Understand What You Find. *International Journal of Human-Computer Interaction*, volume 23, no. 3, Dec. 2007: pp. 205–237, ISSN 1044-7318, doi:10.1080/10447310701702352. Available from: <https://doi.org/10.1080/10447310701702352>
- [9] Bates, M. J. The Design of Browsing and Berrypicking Techniques for the Online Search Interface. *Online Review*, volume 13, no. 5,

- May 1989: pp. 407–424, ISSN 0309-314X, doi:10.1108/eb024320. Available from: <https://www.emerald.com/insight/content/doi/10.1108/eb024320/full/html>
- [10] Pirolli, P.; Card, S. K. Information Foraging Models of Browsers for Very Large Document Spaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '98*, L'Aquila, Italy: Association for Computing Machinery, May 1998, ISBN 978-1-4503-7435-4, pp. 83–93, doi:10.1145/948496.948509. Available from: <https://doi.org/10.1145/948496.948509>
- [11] Experience, W. L. i. R.-B. U. Deceivingly Strong Information Scent Costs Sales. Available from: <https://www.nngroup.com/articles/wrong-information-scent-costs-sales/>
- [12] Furnas, G. W. Effective View Navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '97*, Atlanta, Georgia, United States: ACM Press, 1997, ISBN 978-0-89791-802-2, pp. 367–374, doi:10.1145/258549.258800. Available from: <http://portal.acm.org/citation.cfm?doid=258549.258800>
- [13] White, R. W.; Roth, R. A. *Exploratory Search: Beyond the Query-Response Paradigm*. Number 3 in Synthesis Lectures on Information Concepts, Retrieval, and Services, San Rafael, Calif.: Morgan & Claypool, 2009, ISBN 978-1-59829-783-6 978-1-59829-784-3, oCLC: 730260455.
- [14] The Standard Query Parser — Apache Solr Reference Guide 8.3. Available from: https://lucene.apache.org/solr/guide/8_3/the-standard-query-parser.html
- [15] Lake, M. Desperately Seeking Susan OR Suzie NOT Sushi. *New York Times. September*, volume 3, 1998.
- [16] The Influence of Caption Features on Clickthrough Patterns in Web Search — Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Available from: <https://dl.acm.org/doi/10.1145/1277741.1277767>
- [17] <http://www.nytimes.com/2007/06/03/business/yourmoney/03google.h>: p. 9.
- [18] Franzen, K.; Karlgren, J. Verbosity and Interface Design. *SICS Research Report*, 2000.
- [19] Aula, A.; Käki, M. Less Is More in Web Search Interfaces for Older Adults. *First Monday*, volume 10, no. 7, July 2005, ISSN 13960466, doi: 10.5210/fm.v10i7.1254. Available from: <https://firstmonday.org/ojs/index.php/fm/article/view/1254>

-
- [20] Search Engine Market Share Worldwide. Available from: <https://gs.statcounter.com/search-engine-market-share>
- [21] 63 Fascinating Google Search Statistics (Updated 2019). Available from: <https://bluelist.co/blog/google-stats-and-facts/>
- [22] Infografika: Podíl vyhledávačů Google a Seznam na českém internetu #2019. Jan. 2019. Available from: <https://www.evisions.cz/blog-2019-01-24-infografika-podil-vyhledavacu-google-a-seznam-na-ceskem-internetu-2019/>
- [23] Official Google Blog: Introducing the Knowledge Graph: Things, Not Strings. Available from: <https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html>
- [24] Kelley, L. The Google Feature Magnifying Disinformation. Sept. 2019. Available from: <https://www.theatlantic.com/technology/archive/2019/09/googles-knowledge-panels-are-magnifying-disinformation/598474/>
- [25] How Much of Google's Search Traffic Is Left for Anyone But Themselves? June 2019. Available from: <https://sparktoro.com/blog/how-much-of-googles-search-traffic-is-left-for-anyone-but-themselves/>
- [26] What Do We Make of Wikipedia's Falling Traffic? Jan. 2014. Available from: <https://www.dailydot.com/news/wikipedia-falling-traffic-meaning/>
- [27] Choi, H.; Varian, H. Predicting the Present with Google Trends: PREDICTING THE PRESENT WITH GOOGLE TRENDS. *Economic Record*, volume 88, June 2012: pp. 2–9, ISSN 00130249, doi:10.1111/j.1475-4932.2012.00809.x. Available from: <http://doi.wiley.com/10.1111/j.1475-4932.2012.00809.x>
- [28] Ginsberg, J.; Mohebbi, M. H.; et al. Detecting Influenza Epidemics Using Search Engine Query Data. *Nature*, volume 457, no. 7232, Feb. 2009: pp. 1012–1014, ISSN 0028-0836, 1476-4687, doi:10.1038/nature07634. Available from: <http://www.nature.com/articles/nature07634>
- [29] Kules, B.; Capra, R.; et al. What Do Exploratory Searchers Look at in a Faceted Search Interface? In *Proceedings of the 9th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL '09*, Austin, TX, USA: Association for Computing Machinery, June 2009, ISBN 978-1-60558-322-8, pp. 313–322, doi:10.1145/1555400.1555452. Available from: <https://doi.org/10.1145/1555400.1555452>

- [30] Carrot2 - Open Source Search Results Clustering Engine. Available from: <http://project.carrot2.org/>
- [31] 153 ‘Autocomplete Suggestions’ Design Examples - Baymard Institute. Available from: <https://baymard.com/ecommerce-search/benchmark/page-types/autocomplete-suggestions>
- [32] Ward, D.; Hahn, J.; et al. Autocomplete as Research Tool: A Study on Providing Search Suggestions. *Information Technology and Libraries*, volume 31, no. 4, Dec. 2012: pp. 6–19, ISSN 2163-5226, doi:10.6017/ital.v31i4.1930. Available from: <https://ejournals.bc.edu/index.php/ital/article/view/1930>
- [33] Teevan, J.; Adar, E.; et al. Information Re-Retrieval: Repeat Queries in Yahoo’s Logs. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’07, Amsterdam, The Netherlands: Association for Computing Machinery, July 2007, ISBN 978-1-59593-597-7, pp. 151–158, doi:10.1145/1277741.1277770. Available from: <https://doi.org/10.1145/1277741.1277770>
- [34] Miller, R. B. Response Time in Man-Computer Conversational Transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS ’68 (Fall, Part I), San Francisco, California: Association for Computing Machinery, Dec. 1968, ISBN 978-1-4503-7899-4, pp. 267–277, doi:10.1145/1476589.1476628. Available from: <https://doi.org/10.1145/1476589.1476628>
- [35] Cerdan, L. Algolia — Three Best Practices for Search Autocomplete on Mobile. Available from: [search-autocomplete-on-mobile](https://www.algolia.com/resources/articles/search-autocomplete-on-mobile/)
- [36] Google Has Dropped Google Instant Search. July 2017. Available from: <https://searchengineland.com/google-dropped-google-instant-search-279674>
- [37] Fernandez-Kincade, G. Building an Autosuggest Corpus, Part 1. Feb. 2018. Available from: <https://medium.com/related-works-inc/building-an-autosuggest-corpus-part-1-3acd26056708>
- [38] Cai, F.; de Rijke, M. A Survey of Query Auto Completion in Information Retrieval. *Foundations and Trends® in Information Retrieval*, volume 10, no. 4, 2016: pp. 273–363, ISSN 1554-0669, 1554-0677, doi:10.1561/15000000055. Available from: <http://www.nowpublishers.com/article/Details/INR-055>
- [39] Wang, P.; Mi, X.; et al. Game of Missuggestions: Semantic Analysis of Search-Autocomplete Manipulations. In *Proceedings 2018 Network and*

-
- Distributed System Security Symposium*, San Diego, CA: Internet Society, 2018, ISBN 978-1-891562-49-5, doi:10.14722/ndss.2018.23036. Available from: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07A-1_Wang_paper.pdf
- [40] Bohne, T.; Rönnau, S.; et al. Efficient Keyword Extraction for Meaningful Document Perception. In *Proceedings of the 11th ACM Symposium on Document Engineering - DocEng '11*, Mountain View, California, USA: ACM Press, 2011, ISBN 978-1-4503-0863-2, p. 185, doi:10.1145/2034691.2034732. Available from: <http://dl.acm.org/citation.cfm?doid=2034691.2034732>
- [41] Briandais, R. D. L. File Searching Using Variable Length Keys. 1959, doi:10.1145/1457838.1457895.
- [42] Hadraba, A. *Implementace Invertovaného Indexu*. Dissertation thesis, Masaryk University, Faculty of Informatics, 2015. Available from: <https://is.muni.cz/th/hsr4u/?lang=en>
- [43] Zhuang, R. Data Structure - Trie. Mar. 2016. Available from: <https://jojozhuang.github.io/popular/data-structure/data-structure-trie/>
- [44] Weber, I.; Castillo, C. The Demographics of Web Search. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '10*, Geneva, Switzerland: ACM Press, 2010, ISBN 978-1-4503-0153-4, p. 523, doi:10.1145/1835449.1835537. Available from: <http://portal.acm.org/citation.cfm?doid=1835449.1835537>
- [45] 2. Jazykové korpusy a jejich výstavba - FI-ULI: Úvod do lingvistiky pro informatiky: p. 109.
- [46] Hasan, K. S.; Ng, V. Automatic Keyphrase Extraction: A Survey of the State of the Art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 1262–1273, doi:10.3115/v1/P14-1119. Available from: <http://aclweb.org/anthology/P14-1119>
- [47] Alrehamy, H. H.; Walker, C. SemCluster: Unsupervised Automatic Keyphrase Extraction Using Affinity Propagation. In *Advances in Computational Intelligence Systems*, volume 650, edited by F. Chao; S. Schockaert; Q. Zhang, Cham: Springer International Publishing, 2018, ISBN 978-3-319-66938-0 978-3-319-66939-7, pp. 222–235, doi:10.1007/978-3-319-66939-7_19. Available from: http://link.springer.com/10.1007/978-3-319-66939-7_19

- [48] Dolamic, L.; Savoy, J. Indexing and Stemming Approaches for the Czech Language. *Information Processing & Management*, volume 45, no. 6, Nov. 2009: pp. 714–720, ISSN 03064573, doi:10.1016/j.ipm.2009.06.001. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S0306457309000685>
- [49] Straková, J.; Straka, M.; et al. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 13–18, doi:10.3115/v1/P14-5003. Available from: <https://www.aclweb.org/anthology/P14-5003>
- [50] Rose, S.; Engel, D.; et al. Automatic Keyword Extraction from Individual Documents. In *Text Mining*, edited by M. W. Berry; J. Kogan, Chichester, UK: John Wiley & Sons, Ltd, Mar. 2010, ISBN 978-0-470-68964-6 978-0-470-74982-1, pp. 1–20, doi:10.1002/9780470689646.ch1. Available from: <http://doi.wiley.com/10.1002/9780470689646.ch1>
- [51] Mihalcea, R.; Tarau, P. TextRank: Bringing Order into Texts: p. 8.
- [52] Sematext/Solr-Autocomplete. Sematext Group, Inc., Nov. 2019. Available from: <https://github.com/sematext/solr-autocomplete>

Acronyms

TA ČR Technology Agency of the Czech Republic

IS VaVaI Czech R&D Information System

UI User interface

QAC Query auto completion

AKE Automatic keyphrase extraction

IR Information retrieval

NLP Natural language processing

POS Part of speech

SERP Search engine results page

SEO Search engine optimization

Contents of enclosed CD

	readme.txt	the file with CD contents description
	thesis.pdf	the thesis text in PDF format
	zadani.pdf	the thesis assignment in PDF format
	src	the directory of source codes
	autocomplete	Django app containing QAC functionality
	textmining	Django app containing nlp functionality
	solr	Django app containing a simple Solr client
	thesis	the directory of \LaTeX source codes of the thesis
	images	the directory of images used in the thesis text