Visualization of Corpus Data

BACHELOR’S THESIS

Dominika Talianová

Brno, 2014
Declaration

I, hereby declare that this paper is my original authorial work, which I worked out by my own. All the sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Dominika Talianová

Advisor: Mgr. et Mgr. Vít Baisa
Acknowledgement

In the first place, I would like to thank my advisor Mgr. et Mgr. Vít Baisa for his help, guidance and assistance while working on this thesis. Then I would like to thank my parents, for without their help I wouldn’t be where I am, and especially my mother who has supported me the most. And I must not forget my friend Jan Volmut for his help with the administrative part of this thesis and while working on this text.
Abstract

This thesis focuses on corpus data represented in graphical form. More closely, it consists of a recherché on visualization tools and a website created to hold visualizations based on two features of Sketch Engine, namely Word Sketch and Sketch-diff. These visualizations represent collocations and their salience in connection to different lemmas. The data essential for these visualizations are processed with the use of JavaScript and its D3 library in a JSON format and are provided by Natural Language Processing Centre at Masaryk University, Faculty of Informatics in Brno.
Keywords:
corpus, Word Sketch, Sketch-diff, d3.js, collocation, lemma, salience, JSON, visualization, JavaScript
## Contents

1 Introduction .................................................................................................................. 3

2 Corpus data .................................................................................................................. 4
   2.1 Natural Language Processing ...................................................................................... 5
   2.2 Sketch Engine ........................................................................................................... 5
      2.2.1 The Word Sketch ............................................................................................... 7
      2.2.2 The Word Sketch Difference ............................................................................. 9
      2.2.3 JSON ................................................................................................................ 11

3 Data Visualizations ...................................................................................................... 12
   3.1 Visualization tools – recherché ............................................................................... 12
      3.1.1 R ........................................................................................................................ 12
      3.1.2 Prefuse ................................................................................................................. 12
      3.1.3 Flare .................................................................................................................... 13
      3.1.4 Protovis ............................................................................................................... 13
      3.1.5 Processing .......................................................................................................... 13
      3.1.6 Raphaël ............................................................................................................... 14
      3.1.7 Wordle ............................................................................................................... 14

4 practical part ................................................................................................................. 15
   4.1 D3.js up close .......................................................................................................... 15
      4.1.1 New features to JavaScript .................................................................................. 15
   4.2 Accessing the data ................................................................................................... 17
   4.3 Sketch-diff visualization ......................................................................................... 18
      4.3.1 Form .................................................................................................................... 19
      4.3.2 Obtaining the data ............................................................................................. 20
      4.3.3 Sorting the data ................................................................................................. 20
      4.3.4 Setting up parameters of the SVG ..................................................................... 22
4.3.5 Assigning place in the grid ................................................................. 23
4.3.6 Visualizing the data ........................................................................... 24
4.3.7 Sketch-diff in its full size ................................................................. 27
4.4 Word sketch visualization .................................................................. 30
  4.4.1 Form ................................................................................................. 30
  4.4.2 Obtaining data ................................................................................ 31
  4.4.3 Sorting the data ............................................................................... 31
  4.4.4 Visualizing the data ........................................................................ 33
4.5 Putting it all together ......................................................................... 38
  4.5.1 Sketch-diff page ............................................................................ 38
  4.5.2 Word Sketch page .......................................................................... 41
5 Conclusion .............................................................................................. 43
1 Introduction

Visualization of data of whichever type makes the data easier to take in. It is no difference with corpus data. There is a large amount of techniques to extract corpus data from, such as dictionaries, thesauruses, collocations, frequency statistics, etc. For easier study of natural language, sorted data can be easier to look at in graphical form.

One of the purposes of this thesis was to research the methods used to visualizing corpus data. The second part consisted of creating particular visualizations, which are possible to access through a website created for this purpose only. The data essential for these visualizations were obtained from corpora that are available at the Natural Language Processing Centre [see section 2.1]. To be exact, these visualization are based on two of the features of Sketch Engine [read more on Sketch Engine in section 2.2], namely Word Sketch and Sketch-diff.

The web site allows user to select from two corpora. The first is a corpus of English language named Araneum Angelicum Minus, and the second one, called Bruna Bohemica, is a corpus of Czech language. Then, user can chose the word they want to be visualized and the number of words to be visualized. The visualizations are interactional and also provide the user a direct link to Sketch Engine’s version of these features.

The thesis is practically divided into theoretical and practical part. The theoretical part serves as an introduction for the next part, contains information on corpus data and the research on existing methods in data visualization.

In the practical part is a short description on a JavaScript library used to create the visualizations called D3, and then there’s a description of how the particular visualizations were created and how the website, for both of them was implemented.
2 Corpus data

The word “corpus”, in plural - corpora, comes from Latin and means “body”. In the connection to linguistics it represents large and structured collections of texts taken from spoken or written language and composed in a certain language. Corpora can be monolingual (containing text in one language) or multilingual (containing text from multiple different languages). Basically any collection of words can be called a “corpus”, but in modern linguistics, it is not that simple. Today, the term is often used to refer to systematic text collections that have been computerized. Where “systematic” means that the structure and contents of the corpus follows certain extralinguistic principles [1]. The corpora usually consists of rather large amount of data (millions or sometime even billions of words) therefore they need to be electronically stored and processed.

Each corpus has its own specifications, but there are some properties that are common for all corpora. For easier use and more useful linguistic research, the corpus is subjected to a process, called “annotation”. This means that some particular metadata is attached to every word of the corpus. These metadata are also attached to every grammatical relation or cluster of given words. For example, each word is given a particular part of speech “tag” (e.g. noun, verb, adjective, etc.).

The words that appears in these corpora are called “lemmas” (“lemma” in singular), or “headwords”. Lemma is a canonical form, dictionary form or citation form that represents the inflected forms of a set of words. (e.g. in corpus lemma “do” represents the inflected forms of “does”, “doing”, “did” and “done”). Lemmas are mostly significant for highly inflected languages, such as Spanish or Czech.

As a result of processing corpus data can be considered another element useful in analysing the data called concordance. Concordance shows how the given lemma changes according to context of the text it is related to.
2.1 Natural Language Processing

Natural Language Processing is a field of computer science that focuses on linguistics, interactions between humans and computers, and its main focus is natural language understanding. One of the challenges characteristic for natural language processing is “teaching” computers to “learn” and “understand” human speech. It is a branch of artificial intelligence that is similar to the area of human-computer interaction. Collecting and utilizing lexical data, creating and further improving the corpora and text analysis are the ways to obtain these goals.

In Natural Language Processing Centre (later on only NLP Centre) at Faculty of Informatics, Masaryk University in Brno\(^1\) the research is directed at theoretical and applied results in corpus linguistic, lexical databases, representations of knowledge, meaning representation of expressions of natural language and utilizing the methods for machine learning leading to disambiguate the corpus data. The research here, involves the development of several projects, such as for example Sketch Engine.

2.2 Sketch Engine

Sketch Engine is a web-based program with a number of language-analysis functions. It is a Corpus Query System, taking as input corpus with appropriate level of linguistic mark-up of any language. Sketch Engine is a product of a small research company called Lexical Computing, founded by Adam Kilgarriff in 2003 and is, since then, being developed by Mgr. Pavel Rychlý Ph.D., at NLP Centre at Masaryk University in Brno\(^2\).

The core language-analysis functions are the Concordancer and the Word Sketch. “The concordance is the basic tool for anyone working with a corpus” showing the raw data, the analysis is based on [3]. It displays corresponding

---

\(^1\) NLP Centre at Masaryk University, Faculty of Informatics in Brno

concordances for chosen lemma, with the possibility of adjusting the output by user’s needs. (I.e. user can decide, for example, what query type he wants to search for. Whether it’s a simple word, phrase or lemma, etc.) The output in this case is an index of sentences, or parts of sentences, showing the contextual occurrence of the chosen lemma. The word sketches are corpus-derived summaries of a words grammatical and collocational behaviour [see more in section 2.2.1].

Sketch Engine provides corpora in different languages, from Arabic, through English, Greek, Russian to Vietnamese, Welsh and many others. With the exceptions of few, these corpora are mostly accessible to registered users only, but regarding basic research, every user (with, or without, an account) has the same possibilities to see and try out several features that Sketch Engine provides for each corpus (some of these corpora are quite large as it can be seen on figure 2-1 as they may contain several millions of words). For full understanding and easier use of Sketch Engine’s many functions, there’s a “user manual”, where is explained every aspect of each individual feature and its property\(^1\). Among these features belong, for example, Word sketch and Word sketch difference.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Language} & \text{Corpus name} & \text{Tokens} & \text{Words} \\
\hline
\text{Czech} & \text{Bruna Bohemica (czen 14.04) 445 M} & 444,737,675 & 354,132,863 \\
\text{Czech} & \text{Bruna Bohemica Minor (czen 14.04) 121 M} & 121,046,115 & 94,685,757 \\
\text{English} & \text{Brown} & 1,175,676 & 1,007,299 \\
\text{English} & \text{Araneum Angloicum Minus (En Web 14.04) 119 M} & 119,484,404 & 91,938,147 \\
\text{English} & \text{Penn Corpus of Historical English} & 4,404,931 & 3,800,629 \\
\text{German} & \text{GerManC} & 800,783 & 667,310 \\
\text{German} & \text{GerManC with WS on original cogs} & 800,783 & 667,310 \\
\text{Tajik} & \text{Tajik Web} & 109,805,133 & 93,151,897 \\
\text{Tajik} & \text{Tajik Web} & 109,805,133 & 93,151,897 \\
\hline
\end{array}
\]

\[\text{Figure 2-1. Open corpora preview}\]

\(^1\) Getting Started with the Sketch Engine <https://www.sketchengine.co.uk/documentation/wiki/SkE/GettingStarted>
2.2.1 The Word Sketch

Word sketch is, along the Concordancer, one of the two basic features that Sketch Engine provides. It is a one-page, automatic, corpus-derived summary of a word’s grammatical and collocational behaviour [4]. Word sketch uses statistic and lemmatization based on the use of grammar patterns. Meaning, it considers the aspect of each grammatical relation the word participates in. The full helpfulness of the Word sketch is seen on a large amounts of data, since Words sketch for a rare item can be analysed by hand easily enough.

Word sketch takes as an input one lemma and based on the corpus it belongs to, requires the adjustment of a “part-of-speech” (PoS) factor, which tells the programs what lexical category the lemma comes under. Word sketches working with English corpora require the attribute “lempos”, which is a combination of the lemma and the PoS tag. (E.g. in English, the word “work” can be taken as a verb, and also as a noun, the “lempos” attribute creates a string that looks like “work-v” (meaning verb) or “work-n” (meaning noun) depending on what PoS tag was selected. By this suffix it can be easily decided what lexical class the lemma belongs to. While working with Czech corpora this parameter is not required.)

Word sketch also offers additional settings, which can define the minimum frequency (the lemma is used in collocation with a particular word), minimum score (which shows the salience score for the collocate), set the maximum number of items displayed in each grammatical relation or for example select which grammatical relation the user wants to see. User can also choose between monolingual or bilingual word sketches.

The output, based on the selected parameters, consists of a table or set of tables. Each table represents one grammatical relation and displays the collocations (i.e. the word from the collocation with the given lemma). The given lemma can be called a “node word” as it is connected with every line of the tables into collocations. Every single word has its score and frequency. By clicking on the frequency, the user is redirected on a page where they can see the concordance matching the node word and its chosen collocation.
What follows (in figure 2-2) is a preview of Word sketch. This is an example of the lemma “mother” in an open corpus of English language (Araneum Angeli-cum Minus).

More about Word Sketch can be found in an article by Vladimír Benko called “Compatible Sketch Grammars for Comparable Corpora” [5].

![Figure 2-2. Word sketch preview](attachment://word_sketch_preview.png)

As it can be seen, the given lemma is in the left upper corner, followed by its lexical category, then there is displayed the name of selected corpus and general frequency it occurs in the corpus. The maximum number of items, was limited to ten and each table represents particular grammatical relation, where every word in connection with the node word creates a collocation. The collocates are sorted by their salience (score). The frequency value, supplies a direct link to concordance. The example of concordance for the collocation “mother grandfather” is shown in the small fragment in figure 2-3.

![Figure 2-3. Concordance preview](attachment://concordance_preview.png)
2.2.2 The Word Sketch Difference

The Word sketch difference (Sketch-diff) displays the variations between words. Whereas it can illustrate the contrasts between total opposites it is mostly useful for its ability to help determine the difference between nearly-synonyms or synonyms.

The input of Sketch-diff are two lemmas. For some corpora the “part-of-speech” parameter is required, and therefore both lemmas have to be of the same lexical category. There are some advanced options to select from as well, such as minimum frequency, maximum number of items displayed in one grammatical relation or a user can choose to display the collocations for both lemmas in one box or in separate boxes, where the output is divided into three boxes (one for “Common Patterns”, one for “first lemma only pattern” and one for “second lemma only pattern”). The boxes (tables) are further divided into grammatical relations.

Both options show collocations of the lemmas with other words. Here, both given lemmas are called node words, as they represent one part of the collocation shown in the tables. In every table and on every line (i.e. for every word of the table) there is information about frequency it is used in collocation with both of the lemmas, as well as score. By clicking on the frequency one can see the concordance for that certain collocation. It can occur, that a word doesn’t have a collocation with one of the node words, and the score and frequency are null in that occasion. For better imagination and orientation in the tables, the collocations are aligned and coloured, so that the words with higher score in the collocation with first lemma are green and due to the bottom of the table, and on the top are words coloured red, with higher score in the connection with the second lemma. Words that are used almost evenly or evenly with both node words are in the middle of the table and are coloured white.
This example, shown in figure 2-4, was made from an open corpus of English language, (Araneum Anglicum Minus). The chosen headwords are lemmas “work” and “job” and the lexical category is “non-verb”. Collocates with higher salience in connection with “work” are at the bottom of the tables and are coloured green. And due to the top of the tables are red coloured collocates with higher salience with “job”. This sketch-diff has a limited number of items displayed in each grammatical relation (i.e. in each table). From this example, one can observe, that even though they are synonyms, some collocations are exclusively used in connection to only one of them. Words like “progress” are used only in collocation with work in “progress in work” and can hardly be ever seen in “progress in job”.

As can be seen in figure 2-4, some collocates are evenly or almost evenly used with both headwords. These are the ones in the middle of the tables, and are in white background. As the difference between the words decreases, the colours are more evenly distributed.
2.2.3 JSON

All corpus data from Sketch Engine are obtainable in JSON format. JSON stands for JavaScript Object Notation\(^1\) and it is a lightweight interchange format. JSON is easy to work with for both humans and computers and it can be used as an input or output format.

\(^1\) Web JSON Documentation: <http://www.json.org/>
3 Data Visualizations

To visualize information in general means creating images, diagrams, or animations, mainly to “prove a point” or to “highlight” some interesting or important part of message. In connection to corpus data, the data used to visualize consist of corpora. The visualization then displays the amount of use of each word or relationships and associations between words.

3.1 Visualization tools – recherché

There's unlimited amount of data to be visualized and different ways to do it, with the usage of different programming languages, namely object-oriented, supported by special libraries.

3.1.1 R

One of the many ways to visualizing data is by using R. R is an environment and language that computes statistical data and produces graphical output. It was created by Ross Ihaka and Robert Gentleman in 1993. [6] It is a GNU project, similar to S language and environment, and is licensed as an Open Source. By the Google Trend Chart it is, to this day, considered one of the most closely visualization tools associated with data visualization [7].

3.1.2 Prefuse

Other option is to use the Prefuse visualization toolkit, which is a set of software tools for interactive data visualizations. Prefuse uses Java 2D graphics library and requires Java plug-in in web browsers. A project based on Prefuse, called Enron was created by Jeffrey Michael Heer to present a data visualization system integrating automated applied natural language processing techniques [8].
3.1.3 Flare

Then there's an ActionScript library called Flare, for visualizations that run in the Adobe Flash Player. Flare is used for example for BBC News' to map "Top 100 sites on the Internet". A combination of Prefuse and Flare was used at the IBM Visual Communication Lab for the Many-Eyes visualization service [9], which can be used for visualizing corpus data, as we can see in Jeff Clark's Word Tree visualization [10]. Many Eyes service however require a Java plug-in so later was created a Many Eyes.js library which provides interactive data visualizations using the native web technologies of JavaScript and HTML5 [11].

3.1.4 Protovis

Based on experience from developing Prefuse and Flare, Heer, Michael Bostock and Vadim Ogievetsky created Protovis\(^2\), which was released as an open-source under the BSD license (i.e. the code can be edited, published or used for commercial and non-commercial purposes) and uses JavaScript and SVG for web-native visualizations. The development of Protovis was stopped in 2011 to focus on developing a new visualization library, D3.js. [read more about d3.js in section 4.2]

3.1.5 Processing

Another approach is by using Processing, which is a programming language, development environment and online community [12]. It's based on Java, allows to make interactive visualizations and is free to use. It is used, for example, in a project called 3D Dewey Data Visualizations by Syed Reza Ali, which explores the topics of 3D Space, particle systems, OpenGL and java, alpha blending, bill boarding, user interactivity, self-organizing algorithms (Kohonon), and electromagnetic attractions & repulsion [13]. Processing.js is the sister project of Processing and is designed for web, allowing the code to run

---

1 flare Data Visualization for the Web <http://flare.prefuse.org/>
2 Protovis at GitHub <http://mbostock.github.io/protovis/ex/>
by any HTML5 compatible browser [14]. An example of using Processing.js is a Wiki visualization by Matt Ryall\(^1\). In this visualization is also used a JavaScript library called Raphaël.

### 3.1.6 Raphaël

Another way of approach is by using Raphaël, small JavaScript library, which purpose is to simplify the work with vector graphics on the web. Every graphical object created in Raphaël is a DOM as well, so it can be attached event handlers or modified later [15]. It is developed by Dmitry Baranovskiy and is licensed under MIT.

### 3.1.7 Wordle

In concern with corpus data, it may be interesting to mention a project named Wordle\(^2\). It is was created by Jonathan Feinberg and is based on Java 2D API, therefore requires Java plug-in installed in a browser. Wordle allows free and easy “word cloud” visualization of any text found online and submitted through URL or manually typed right in to the form of the application.

---

\(^1\) Wiki Visualization <http://mattryall.net/blog/2008/11/wiki-visualisations-with-javascript>

\(^2\) Wordle <http://www.wordle.net/create>
4 practical part

One of the purposes of this thesis was to create a practical tool that visualizes particular data (i.e. displays given data in graphical form). The data, in this case, are taken from the corpus of English language called Araneum Angelicu- cum Minus and Czech language corpus called Bruna Bohemica. The visualizations are built on the functions of Sketch Engine, namely Word Sketch and Sketch-diff, and provide a different look at the data obtained through these features.

The source code that creates the visualization is written in JavaScript and uses the D3.js library, while the data is accessed by php proxy.

4.1 D3.js up close

D3 stands for Data-Driven Documents\(^1\) and the name is an allusion to the W3, World Wide Web, which serves as an implication of technologies underlying the D3 itself [16]. D3 is a JavaScript library with improved support for animation and interaction. It is built on many of the concepts of Protovis including the usage of wildly implemented Scalable vector Graphics (SVG) and JavaScript as well as HTML5 and Cascading Style Sheets (CSS3) standards. Therefore, D3 doesn't require any special plug-ins just a modern web browser.

D3 can be considered a rather young library as its development was noted only in 2011. It was released as an Open Source and is licensed under BSD on GitHub [16].

4.1.1 New features to JavaScript:
One of many improved and simplified aspects is, that D3 allows to bind data directly to DOM (Document Object Model) elements in the web page. This is done through functions called “selections”. What previously had to be done by

\(^1\) D3 <http://d3js.org/>
document.getElementById(elementID) can be now done by simply “selecting” the element by calling d3.select(“#elementID”) or to call more elements, one can use d3.selectAll(“#elementID”), which doesn’t require the use of loops and such methods.

These selections can be used to create a new element, that doesn’t exist in the DOM as of yet, by applying a specific selection .enter() or to remove element by calling .remove().

Selectors are supported by modern web browsers without bigger problems, and solution for older browsers is provided in the form of a JavaScript CSS selector engine called Sizzle¹.

D3 also introduces the possibility of using dynamic properties, meaning styles, attributes and other properties can be specified as functions.

D3 provides large number of graphical layouts. Each one of them has its own defined properties and attributes. These layouts don’t provide an already created visualization, but transform the data according to the needs of the specific attribute of given visual task. Some of the most common layouts are: Pie, Force, Treemap, Tree, Partition, etc.

It was created and is being developed with the goal to process large amount of data, so it is extremely fast and supports dynamic behaviours for interactive visualizations and animations. And therefore the input can be in a JSON format or even a geoJSON, which is a specific use of JSON that provides encoded geodata for web applications.

For this purpose, D3 has its own variation of a function to load JSON encoded data. It is d3.json(). This function requires an input in the form of an external script or using XMLHttpRequest, or XHR. This allows the data to load asynchronously and the rest of the page is displayed while the data is loading. Second part, that is for d3.json() obligatory is a callback function. The code that depends on the loaded data, should be contained within the callback function.

¹ Sizzle <http://sizzlejs.com/>
Usage of the D3 library can be seen, for example, in the D3 gallery on GitHub\(^1\). A great example is work by Daniel G. Taylor, who made a visualization of The Holy Bible contradictions\(^2\). It’s a whole complex of website with interactive visualization with active links. This project was inspired by the Reason Project’s poster of biblical contradictions\(^3\), which was in turn inspired by Chris Harrison’s Bible Visualization\(^4\). Concerning corpus data, Harrison made a visualization called Web Trigrams: visualizing Google’s Tri-Gram Data. Tri-grams are parts of sentences consisting of three words, taken from the Google corpus. The visualization shows interesting comparison between the first words and the following words for each one of them [17]. A more often data visualization example is a Word Cloud. For example, by Jason Davies\(^5\).

### 4.2 Accessing the data

The data is loaded by `d3.json()`, which as an input takes an URL. To avoid problems with its asynchrony [see section 4.1.1], data is accessed with the use of a short, simple php script, utilizing php’s proxy properties. The use of a php script also prevents possible cross-domain limitations.

In the file “proxy2.php” is defined simple `$_GET` method to retrieve the URL form the main script. As the URL is handed over in its encoded form, in the php script it needs to be decoded, using function `urldecode()` . This helps profoundly in manipulation with words from Czech language. In this script, it is also specified, that the output is in JSON format.

---

\(^1\) GitHub D3 Gallery <https://github.com/mbostock/d3/wiki/Gallery>

\(^2\) Visualization of The Holy Bible contradictions <http://bibviz.com/>

\(^3\) Project Reason <http://www.project-reason.org/gallery3/image/105/>

\(^4\) Chris Harrison <http://www.chrisharrison.net/index.php/Visualizations/BibleViz>


4.3 Sketch-diff visualization

This visualization shows the difference between words and their collocations. It is based on the Sketch-diff feature of Sketch Engine [see section 2.2.2], and therefore it preserves most of its properties. To partially match the appearance it is coloured in white with shades of red and green.

The onlooker can select between Czech corpus and English corpus. Based on selected corpus, they then write in first and second lemma. If the English corpus was selected, the tool requires the determination of an appropriate lexical category, which has to match both given lemmas. Then there’s the possibility to limit the number of collocates to be visualized, based on limiting the number displayed for every grammatical category. This limitation is not required and, while not specified differently, the default number is ten.

The layout of the visualization also, to some extension, mirrors Sketch Engine’s version of Sketch-diff. The lemmas are at the top and bottom, posing as opposite poles, while their collocations are displayed between them, and arranged due to their salience in collocation with the first, or the second lemma. The salience is also shown in the font size of each word (i.e. the bigger is the salience, the bigger is the size of font).

The collocations are divided into seven individual categories, due to the class attribute, obtained from loaded data. This matches their salience and frequency of use in connection to first or second lemma, too.

In addition, it is also possible to access the concordance of every displayed word by clicking on it. By clicking on one of the headwords, one can go to Sketch Engine’s version of Sketch-diff.

The basic idea of this visualization is an abstract grind, where each field is reserved for one word. This guarantees that the words of the same salience, won’t be written over each other, and therefore, the final result will be readable and every visualized word can be accessible, by clicking on it.
4.3.1 Form

The attributes that are obligatory for the visualization and affect it, are entered by the user into a form at the top of the web page of the particular visualization. The form is then processed in an external JavaScript file, called “myFormSd.js”, in function parseForm(form). In this file is the processed whole code for sorting the data and the visualization.

Here (in function parseForm(form)), each attribute is firstly tested for containing any unacceptable value. If such value is found, the user is warned by a pop up box alert(), which stands the possible problem. This also happens if user enters values, by which it is impossible to find the corresponding sketch-diff, for example if one selects corpus of Czech language and as lemma enters a word in English.

By sorting the individual attributes obtained in the form, parseForm(form) assigns them values that are recognizable for Sketch Engine in a query. These attributes are:

a) corpus – representing language corpus, it can possess values: “en_araneum_minus”, or “cz_bruna” depending on the chosen corpus.

b) lemma1 – first lemma, for Sketch Engine recognizable as “lemma”

c) lemma2 – second lemma, in the query detected as “lemma2”

d) lempos – for Sketch Engine, known as “lpos”, obtaining the value of “part-of-speech” attribute. The value being “-x” for non-verb words and “-v” for verbs. It is possible to select blank option, which is sufficient only if the Czech corpus was chosen for the query. To prevent human errors its default value is an empty string, and only if the English corpus is chosen, the user is alerted by a pop up box, informing them about their mistake.

e) maxCommon – representing maximum number of collocates in one grammatical relation, in the query represented as “maxCommon”. This is needed in the query, but it is not obligatory to fill in the value by user. If they chose to leave the field empty, the default value is 10.
Once, the attributes are assigned their value, they’re used to compose the query referring to the specific sketch difference in Sketch Engine, with the help of the “proxy2.php” script file. The data is obtained in JSON format, by adding “&format=json” at the end of the query and are handed over to method urllibencode() to help with parsing additional characters (e.g. some of the Czech alphabet letters). The output of this external script is a string, consisting of proxy2.php?url and the generated link from above.

4.3.2 Obtaining the data

The data is obtained through d3.json(), which as an input takes the returned values of the parseForm(form) function and makes a query to Sketch Engine, (demanding sketch difference characterized by the values included in the query) and the values are handed over to the “callback” function as data.

4.3.3 Sorting the data

The acquired data is in JSON format. More accurately, it is an object, containing objects that contain objects or arrays of objects, or objects of arrays of objects. Example of this can be seen in figure 4-1.
To work with data with this kind of hierarchy can prove rather difficult and for the purpose of the visualization the important part is located in the `Rows` array. Because of that, the needed data is stored in a variable `com` representing the `common` key from `data`. As of now, it is an array of objects of arrays and it is still not simple enough to access individual keys and their values.
Therefore, new variable called `newObject` is introduced, and represents a simple array of objects. Each object in this array contains:

a) grammatical relation of each word
b) the word from the collocation
c) frequency for each word
d) attribute `seek` for each word (important for creating link to concordance [see end of section 4.3.6])
e) colour, containing string “color”, for now
f) rank of each word, representing the salience of the word in relation to the lemmas
g) category, representing the `class` attribute, used in further division of the data

To guarantee that the position of a word, in the final visualization, will be moved on the y axis due to the salience, the data from `newObject` is divided into seven separated arrays due to the `category` attribute. This arrays are named from `height1Array` to `height7Array`. In this part, the categories are also assigned their colour. Arrays `height1Array` to `height6Array` represent a range from the top to the bottom, where `height1Array` is at the top and is coloured green, and `height6Array` is at the bottom and is coloured in red. Array `height7Array` is the middle part and is coloured white.

The arrays are then connected into single array of objects, by function `concat()`. The new array is called `joinedData` and has the same attributes as `newObject` but the order of the objects is sorted so that each can be assigned their place in the grid easily.

### 4.3.4 Setting up parameters of the SVG

Parameters here, represent the aspects that affect the final visualization. The height and the width is in this case set to a given number and it doesn’t change, but the count of words that need to be visualized affects the general look and readability of the visualization.
Function `setParameters(newObject)` is where this is taken care of. It takes as an input `newObject` because it needs to know the count of objects in the array. The count is store in variable `temp` for easier manipulation through the function. There are given 4 possibilities that can occur and they are distinguished by the count of words to be visualized. In each option is set different number of columns (variable `colCount`). This helps to determine the count of rows (`rowCount`), the height of each row (`rowHeight`) and the width of each column (`colWidth`).

The rows are counted as count of words divided by the count of columns. In case the count of the words is an odd number, function `Math.floor()` is applied to this, and to ensure that no word will be left out there is 1 line added to the rows. To set the variables `colCount`, `rowCount`, `rowHeight` and `colWidth` is the purpose of this function.

### 4.3.5 Assigning place in the grid

The place in a grid of every individual word is set by variables `ypos` indicating its position on y axis (starting at value of `rowHeight + 40`), `xpos` referring to its position on x axis, and `index` which helps later with keeping the index of the word, that already has set its position. This positions are modified and allocated to the objects in the function `prepareData(joinedData)`.

This function takes `joinedData` as an input, assigns them to temporary variable `newData`, while going through two loops, assigns each object its position. The first is `while()` with the condition at the beginning and it is set so, that the position on y axis will be increasing, until it reaches the height of the SVG minus 40 pixels (the reason for this will be explained later). Inside the `while()` the function computes words’ position on the x axis as width of a column multiplied by the current number of the column. All the x positions are moved a little bit to the right, by adding `padding` to the final calculated x position. This is repeated while index doesn’t reaches the length of `newData`.

Inside the inner loop are these positions added to a new variable called `usableData`. To this array are later pushed two, manually created objects with the same keys. These objects represent `lemma1` and `lemma2`. Their position is “outside” the grid and is computed separately. The x axis position is set to be
the exact middle of the whole width of the SVG. It is done as width/2 –
(lemma1.length/2). The same principle applies to lemma2. Using just
“width/2” would make the word start at the centre of the x axis, but, since the
word doesn’t consist of only one point, in the final effect it would be still closer
to the right edge. That is taken care of by shifting it to the left by half of the
length of the given word.

Concerning the y axis, it is set to be at 40px for the lemma1 and 5px above
the bottom edge of the SVG (the 5 pixels at the bottom serve as a padding).
Both words need empty space, and that’s the reason for the ypos to be set to
rowHeight + 40 and for the condition to control, if the height didn’t cross
height-40 boundary.

4.3.6 Visualizing the data

First of all is used function to remove the SVG, even if there is none to remove
yet. This makes sure, that when the script is called the second time (when some-
one wants to visualize some different or even the same data again) the new
SVG will be created in the place of the old one, and not under it.

The creation of new SVG is a series of basic chain of commands from d3.js
library and it is appended to an already prepared DOM element visualization.
Which is, right under it, appended shape of a rectangle that serves as a
background for the visualization.

The actual visualization is then created as text with input data usable-
Data. The text displayed is taken from the object’s key “word” and there is an
exception for lemma1 and lemma2, making it so that they acquire the values
entered in the form at the beginning. What follows is a set of attributes. The
x, y and fill (colour) attribute are extracted from the objects from usable-
Data.

The font-size is set with the help of a function called setRangeEdges() and
d3.scale.linear(). The size of font depends on the size of the “box”
the word is placed in. As the d3.scale.linear() works with two parts,
first it takes an input domain of values and then needs an output range. In this
case, the domain is limited with the minimum and maximum score. These val-
ues are calculated with d3.min and d3.max. They take as an input an array of
numeric values, and therefore it is easier to create a new auxiliary array of score values, taken from the `newObject`. As these data was had a string values it needed to be parsed and converted into numbers, before it could be used with the functions maximum and minimum. Then the range is calculated in the `setRangeEdges()` function.

The font size alters by the score for all words but `lemma1` and `lemma2` which have the size set to 40px. The size of other words is handled as `sizeScale(d.rank)` depending on the higher score.

The interactivity of the visualization is assured by `mouseover` function, which helps the user to orient in the text by enlarging the word, the cursor is currently hovering over.

By clicking on the word is, in the reserved place above the visualization, displayed link leading to a concordance of the chosen word. This is done with the help of attribute `seek`. Each word has two `seek` attributes and the function chooses the one with higher score. With this helps the function `returnMaxScore(d)`. By clicking at one of the headwords, one can create link leading to Sketch Engine’s Sketch-diff. This takes is taken care of by the function `createLink(d)`. This is shown in figure 4-2 on next page, where mouse cursor is hovering over the word “creative” and it was clicked at, so in a space over the visualization can be seen link, leading to concordance.

In the picture below, in figure 4-3, is shown what this particular concordance looks like.
Figure 4-2. Sketch-diff visualization preview

special participation in his own creative work, blessed male and female saying, "Increase special participation in his own creative work. Thus he blessed male and female saying for this unplanned hiatus in the creative work, I would never have gotten to it. Tobey Just as all are included in the creative work of God "in the beginning," so all are joy of spending living energy in creative work; rather the shameless, merciless driving special participation in His own creative work, blessed male and female, saying: " Truth, and Happiness in our daily creative work of building a better brain. Current research in making articles consists in creative work. This means, not that you shall design involved. The attribution in creative work has never gotten the spotlight it deserves

Figure 4-3. Concordance for the words “work” and “creative” preview
4.3.7 Sketch-diff in its full size

The bigger the amount of data to be visualized, the harder it becomes to read and decipher the words in the main visualization. For this reason, there is a “zoom” version available on the site. Clicking at a button “show” that will be created after executing function `visualize()` will open new tab, where the visualization is displayed in its full size.

It has the same input aspects as the main visualization. These attributes are handed over from the main page with method `get` and then parsed in external script “myFormSd2.js”. As it is taken care of in the main visualization, there is no need to check for human errors like unfilled obligatory attribute (e.g. part-of-speech tag).

The major difference between this visualization and the main one, is the function that computes the width and height aspects of the final SVG. While in the main one, set values are width and height, here are set attributes: `colWidth` and `rowHeight`. Again, there are four possibilities to arrange the data and in each the width is computed as `colWidth` multiplied by `colCount` and the height is calculated as `rowHeight` multiplied by `rowCount`. 
What can be seen in figure 4-4 is a Sketch-diff visualization with maximum number of words in one grammatical relation set to 70. It is clear that such visualization isn’t readable at all. Because of this, there is the possibility to open it in new tab. This can be seen in figure 4-5. (This picture had to be split into two as it is rather large.)
Figure 4-5. Enlarged Sketch-diff visualization opened in new tab preview
4.4 Word sketch visualization

This visualization displays the salience of words in collocations with given headword (lemma). It is built on the Sketch Engine’s Word Sketch, but in comparison to the Sketch-diff visualization [section 4.2] its appearance is completely different.

This tool as an input requires to specify corpus, headword and the number of the words to be visualized. If the corpus of English language was chosen, it is obligatory to select “part of speech” corresponding to the given lemma.

In the final result, an onlooker can see a colourful ring, surrounded by visualized words. The layout used in this visualization is called pie and the type of this pie is a donut (because of the empty space at the centre). In the centre of the ring is displayed the given headword. These words are chosen by their salience, making the first one at the top of the ring, the one with highest score and continuing clock-wise to the end of the circle. The various colours represent grammatical relation for each word.

4.4.1 Form

The attributes necessary for the visualization, are also (as they are in Sketch-diff) entered into a form at the top of the page. The whole visualization is executed by clicking on the button “visualize” which, in return calls function submitParameters(form). First of all, the function submit(form) is called.

Here, as well, is firstly taken care of the possibility that user leaves some obligatory form field empty, or the value filled in isn’t correct [see section 4.2.1]. The major difference is that while Sketch-diff needs two headwords, Word sketch requires only one, and therefore, only one lemma is entered into the form. Then user enters the number of words they want to be visualized, which is limited to a range from 10 to 200.

In this function is, as well, composed the link for Sketch Engine’s query. The link is stored in a variable inputLink and is then encoded and connected with a string “proxy2.php?url=“ in another variable called inputData. To set this variable (inputData) is the purpose of this function.
4.4.2 Obtaining data

The data is obtained through `d3.json()` function with the assistance of the external “proxy2.php” script. The php script decodes the link and creates a query for corresponding Word Sketch in Sketch Engine. This technique of creating the particular query prevents the `d3.json()` to be executed asynchronously and limits possible cross-domain problems, while the `urlencode()` and `urldecode()` functions make sure, there isn’t a problem with processing letters from Czech alphabet.

4.4.3 Sorting the data

The acquired data is in JSON format and its hierarchy is quite complicated for easy use in other functions. The original data represent an object of arrays of objects that contain another arrays. This is shown in figure 4-6.
To make the access easier, the data is stored in variable `gramrel`. Later is created a new variable called `words` with simpler hierarchy, as it consists of an array of objects. An object is pushed into this array only if it has a `word` key. Every object then contains the same set of keys and they are:

a) `gramrel` – signifying grammatical relation of the word  
b) `color` – colour belonging to the specific word based on its grammatical relation (How are the colours chosen will be explained later.)  
c) `word` – word participating in the collocation  
d) `score` – salience of the word  
e) `seek` – necessary to create the query for concordance [see more at the end of section 4.4.4]

The colours are manually chosen and entered in an array named `colors` in hexadecimal code. On this array is then applied function `gramrelColor()`
which uses d3’s `d3.scale.ordinal` and as an input domain takes an array of grammatical relations (gramrels) and as an output range takes the array of colours.

The final visualization needs to visualize only the first n values with the highest score (“n” being the maximum number of words to be visualized in the form at the beginning), so these objects are then sorted, based on their score. Beginning with the highest score to the lowest.

As the final part of sorting the data, is created yet another new variable to store the data that will be used in the visualization. It is called `newWords` and takes only the first n words from the sorted array `words`.

### 4.4.4 Visualizing the data

As first, to ensure the new visualization would always replace the old one, is used a function `remove()`. The visualization is based on the example posted on GitHub¹. After that, new SVG is appended to the “visualization” element, already existing in the DOM. It has set width and height and its centre is moved to the middle of the SVG.

Then a rectangle is appended to the SVG, serving as a background for the visualization. It has the same properties as the SVG. After that, the text of the headword (lemma) is printed in the middle. The size of font for the headword is calculated in a function `lemmaFontSizeScale()`. It takes as an input the smallest possible size of the inner radius and its possible biggest size and as an output the range from 15 to 40. This range represents the font size of the lemma based on the size of inner radius.

Next follows the function to set the inner and outer radius of the final ring. For calculation of the size of the radius is used the function `radiusScale()`. This function uses d3’s linear scaling and as an input is given the minimum and maximum possible number of the words that are to be visualized. Then this scale is applied to `outerRadius` from which is later computed `innerRadius`. The inner radius is computed as 4/5 of outer radius. Once the radius

---

¹ Simple D3 Pie Chart with Magnitudes in Arcs and Legends Outside and Along Arcs < http://bl.ocks.org/Guerino1/2295263>
is set, the pie layout is applied its aspects. This particular layout has its wedges divided into even pieces. Their count depends on the number of words given to the visualization. This is assured by an anonymous function that returns value of one for every component of the data. The “sort” attribute is set to null which, therefore, defaults to sorting the order in the pie clock-wise with the biggest value at the top.

In another part are prepared the wedges (the arcs of the pie). By using data(pie) the layout is given the “piefied” data, where each arc contains its stratAngle, endAngle and value. Each arc is appended a g element of the DOM and every slice has a class wedge. In d3, anything that doesn’t have ordinary shape (such as rectangle or circle) can be drawn using path. This is now used to draw the actual arcs. After that, each arc is assigned its own colour, depending on the grammatical relation.

The main part of the visualization consists of the text appended to the SVG. The words are written around the ring, in an angle corresponding the angle of the arcs. This is assured by several attributes. At first there is attribute text-anchor telling the visualization on which side of the ring the words is situated. This function is taken from a question on stackOverflow\(^1\). For this part to work, it is necessary to calculate the length of the arc. This is based on the Pythagorean Theorem. This makes sure, that each word starts at the end of an arc and keeps its angle and is still readable. The words are then rotated by function angle(d), which. This function is used to convert from radians to degrees as well.

What follows next is the fill attribute given the colour from the data. Attribute font-size changes its value due to the salience of the lemmas. It is computed and stored in variable fontSize, which is later used in fontSizeScale to scale the size, making it largest at the top of the ring and continuously dropping off to half of the original size. The scale function is based on the count of words (objects) and in visualization is represented as the index \(i\) of the word \(\text{fontSizeScale}(i)\). The basic idea is that it calculates the circumference of the ring based on its outerRadius that is then divided by

---

1 Rotating text labels around pie <http://stackoverflow.com/questions/13958929/how-to-properly-rotate-text-labels-in-a-d3-sunburst-diagram>
the count of the words. To make sure the word wouldn’t be bigger than the place reserved for it, on this result is applied function `Math.floor`, helping in case of non-integral division. The 4px are added just to make the words more readable.

As means of interactivity there is couple features used. The words enlarge while hovering over them with mouse’s cursor. When a word in the visualization is clicked on, the link to its concordance is displayed above the visualization. If the headword is clicked on, the displayed link leads to Sketch Engine’s Word sketch. Example can be seen in *figure 4-7*, where the word “widowed” is clicked on and link to its concordance is displayed above the visualization.

![Figure 4-7. Word Sketch visualization with “clicked” word preview](image)

What follows are three pictures of Word Sketch visualization with different number of words to be visualized. (figure 4-8, figure 4-9 and figure 4-10)
Figure 4-8. Word Sketch visualization with 10 words preview
Figure 4-9. Word Sketch visualization with 60 words preview

Figure 4-10. Word Sketch visualization with 150 words preview
4.5 Putting it all together

Both visualizations are connected placed on a website. This website allows the onlooker to read a little about these visualization, while it provides a possibility to try out the particular visualizations. The layout of the web isn’t complicated and consist of a head, body and footer. Under the head is a simple horizontal navigation bar with links to:

a) “home” - home page containing some basic information about corpus data
b) “about” - displaying the information about these particular visualizations
c) “contact” – providing the contact email address of the author and links to Sketch Engine’s and NLP Centre’s websites
d) “Sketch-diff” – providing the Sketch-diff visualization
e) “Word sketch” – providing Word sketch visualization.

The actual web consists of 6 html files, because to the main Sketch-diff belongs an additional Sketch-diff which can be opened separately in different tab. The web also uses external CSS file and a “reset” file generated at meyer-web.

4.5.1 Sketch-diff page

The first thing seen on the Sketch-diff page is a little information explaining how to manipulate with the visualization and how to fill in the form at the beginning. This includes some basic rules that need to be complied, otherwise it won’t be possible to create the visualization. Next follows the actual form, where user selects corpus, enters first and second headword, selects the part of

---

2 CSS Tools, Reset CSS <http://meyerweb.com/eric/tools/css/reset/>
speech for chosen headwords and writes in the maximum number of items in one grammatical relation. For drawing the visualization based on these parameters serves the button “visualize”.

On the page, under the form, is also a space reserved for link to the concordance or Sketch Engine’s Sketch-diff, which is written out after clicking on the visualized word. Here will, after executing the visualization, also appear the button allowing to show this visualization in a new tab in its original size. This can be observed in figure 4-11 and figure 4-10.

Figure 4-11. Sketch-diff web page form preview
Figure 4-12. Sketch-diff web page visualization preview
4.5.2 Word Sketch page

First thing that can be seen at this page is also a bit of information, explaining how to handle the visualization and what the particular parts of the visualization mean. Here are, as well, written few simple rules for the visualization.

Then, there is the form where user can select the corpus, enter the headword, select its lexical category, and choose the number of words they want to be visualized. This all is executed by clicking on the button “visualize”.

Under the form is a reserved space for a link to the concordance or Sketch Engine’s Word Sketch. This link will appear once the visualization is executed and the user clicks on a word in it. This can be seen in figure 4-13 and figure 4-14.

![Figure 4-13. Word Sketch web page form preview](image)
Figure 4-14. Word Sketch web page visualization preview
5 Conclusion

The final result of this work is a website. This site allows users to use two of Sketch Engine’s features, namely Sketch-diff and Word Sketch. The manipulation with these features is more intuitive than on Sketch Engine and the result is easier to look at. Even though it doesn’t display so much information at first sight, this data is included in the visualization itself. For Sketch-diff it means that the size of each word mirrors its score, and the position of the word itself can tell with which headword the lemma is associated more often. Considering the Word Sketch visualization, the most outstanding aspect is that words that are visualized are not from one grammatical relation, but are selected as the words with highest score from all the relations on Sketch Engine’s Word Sketch for this particular query. Their score is also visible in their size. If a user requires to see it in more ordinary way, they can easily get to the original feature on Sketch Engine through the headword.

These visualizations use 2 different language corpora, but it is not difficult to add more corpora in the future or change the conditions on which the words are visualized. The same way can be changed the information displayed after clicking on the word, or while hovering a mouse over it.
Bibliography


Electronic appendixes

Part of this thesis is also this list of attachments:

about.html - web page with info about this thesis
contact.html - web page with contact
index.html – home page
wordSketch.html – web page with Word Sketch visualization
sketchDiff.html - web page with Sketch-diff visualization
sketchDiff2.html – web page with “zoomable” Sketch-diff visualization
myFormSd.js – JavaScript file to process the Sketch-diff visualization
myFormSD2.js – JavaScript file to process the “zoomable” Sketch-diff visualization
myFormWS.js – JavaScript file to process Word Sketch visualization
proxy2.php – php script file
d3.v3.js - D3 JavaScript library (version 3)
css/newStyleSheet.css – style sheet for the website
css/reset.css – reset style sheet for the website