

Assignment of master's thesis

Title:	Crossword Generator using Web Data
Student:	Bc. Adam Benda
Supervisor:	Ing. Jaroslav Kuchař, Ph.D.
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Web Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

The goal is to create a web application for generating Swedish-style crossword puzzles based on user preferences.

- Word List generating
 - Design and implement a tool obtaining words and their definitions from linked data.
 - Design and implement a tool fetching additional relevant information about the words: categorization, difficulty.
 - Discuss other possible word sources.
- Crossword generating
 - Research the NP-complete problem of crossword generating and existing solutions
 - Design and implement an algorithm for crossword generating from the obtained Word List. Allow parametrization by choice of preferred word categories.
- Web application
 - Design an architecture to interconnect all the parts.
 - Create a complete pipeline using the Word List and Crossword generating tools.
 - Implement and test the web application where a user requests a crossword and obtains the result.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Crossword Generator using Web Data

Bc. Adam Benda

Department of Software Engineering
Supervisor: Ing. Jaroslav Kuchař, Ph.D.

May 6, 2021

Acknowledgements

I would like to express my gratitude to the people around me who supported me during this project development and thesis writing - most of all to my fiancée and my family.

Thanks to Jan Beneš for the thorough spell check.

And thanks to my supervisor Jaroslav Kuchař for the candid feedback and willingness for frequent consultations.

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 accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 6, 2021

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2021 Adam Benda. 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

Benda, Adam. *Crossword Generator using Web Data*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021. Also available from: (<https://generator-krizovek.cz/>).

Abstrakt

V práci byl vyvinut software ke generování křížovek, který z propojených dat projektu Wikidata získává slova s legendami a sématickou kategorizací. Je implementována heuristika konstruuující křížovky s modifikací, jenž umožňuje upřednostňovat slova z kategorií, které si zvolil uživatel. V práci je také vytvořeno jednoduché webové uživatelské rozhraní a všechny části procesu výroby křížovky jsou propojeny.

Klíčová slova Propojená data, Znalostní graf, Generování slovníku, Generování křížovky, Křížovka, Konstrukce křížovky, Optimalizační kombinatorický problém

Abstract

This Thesis develops a Crossword Generation software, sourcing the list of words and semantic word's categorizations from Wikidata, the Linked Data Knowledge Base. A crossword constructing heuristic is implemented and modified to prefer user-defined categories of words. A simple web-based user interface is developed and the project parts are interconnected to form a full stack automated crossword generator.

Keywords Linked Data, Knowledge Base, Knowledge Graph, Wikidata, Word List, Classification, Word Puzzle Generator, Clue, Crossword Construction, Optimization Combinatorial Problem

Contents

Introduction	1
Work structure	1
1 Business Analysis	3
1.0.1 Crossword Classification	3
1.0.1.1 Interlocking	3
1.0.1.2 Clues	3
1.0.1.3 Grid Style	4
1.0.1.4 Secret	5
1.1 Problem Definition	5
1.1.1 Vision	5
1.1.2 Constrains of this thesis	5
1.1.3 Business	5
1.1.3.1 Competition	6
2 Word List Generation	7
2.1 Introduction	7
2.1.1 Problem Definition	7
2.1.1.1 List Of Words	7
2.1.1.2 Clues	7
2.1.1.3 Categorization and Difficulty	7
2.1.2 Internationalization	8
2.2 Analysis	8
2.2.1 Linked Data	8
2.2.2 Knowledge Graphs	8
2.2.2.1 DBpedia	9
2.2.2.2 Wikidata	9
2.2.2.3 YAGO	9
2.2.3 Previous Work	9

2.3	Data Sources	9
2.3.1	Linked Data: Wikidata	9
2.3.1.1	About the Choice	9
2.3.1.2	Local Replication	10
2.3.1.3	Clue Generation	11
2.3.1.4	Categorization Use	11
2.3.2	Wikipedia	11
2.3.2.1	Parsing Wikipedia Article Dump	12
2.3.3	Wikipedia Pageviews	13
2.3.4	Korpus.cz	13
2.3.5	Linked Data Benefit: Machine Learning	14
2.3.5.1	Key Nodes Selection	15
2.3.5.2	Machine Learning Problem Definition	15
2.3.5.3	Method Choice	16
2.3.5.4	Performance	17
2.3.5.5	Extensibility	18
2.4	Other Data Sources Proposals	19
2.4.1	Wiktionary	19
2.4.2	Corpus	19
2.4.3	Open Street Map	19
2.4.4	Abbreviation Generator	19
2.4.4.1	Generate Clues From Linked Data	20
2.5	Results	20
2.5.1	Exported Data Format	20
2.5.2	Obtained words	20
3	Crossword Generation	23
3.1	Analysis and Design	23
3.1.1	Formal Definition	23
3.1.2	Constructive Combinatorial Problem	24
3.1.2.1	NP Complete	24
3.1.3	State of the Art	24
3.1.3.1	Variation: Unconstrained Variant	25
3.1.3.2	Variation: Solving the Crossword	25
3.1.4	Optimization Combinatorial Problem	25
3.1.4.1	Problem is NPH	26
3.2	Realisation	26
3.2.1	Heuristics: Decision Problem	26
3.2.1.1	Heuristic Strategy	27
3.2.1.2	Best Option	28
3.2.1.3	Search Structure	28
3.2.2	Heuristic: Optimization Problem	28
3.2.2.1	Improvements Proposal	29
3.3	Results	29

3.3.1	Convergence	29
3.3.2	Performance	29
4	Web Application	31
4.1	User Interface	31
4.1.1	User Case Analysis	31
4.1.1.1	Target Audience Scenarios	31
4.1.1.2	One Consumer — Electronic	31
4.1.1.3	One Consumer — Pen & Paper	32
4.1.1.4	Multiple Consumers — Electronic	32
4.1.1.5	Multiple Consumers — Pen & Paper	32
4.1.1.6	Teacher	32
4.1.2	Design	33
4.1.2.1	Feature List	33
4.1.2.2	Frontend and Backend	33
4.1.2.3	Workers	34
4.1.2.4	Architecture	35
4.1.3	Realisation	35
4.1.3.1	Frontend	35
4.1.3.2	Backend	36
4.1.3.3	Backend Data Model	36
4.1.3.4	Working Queue	37
4.1.4	Screenshots	38
4.2	Application Architecture	40
4.2.1	Running Modes	40
4.2.1.1	Word List Generation	41
4.2.1.2	Crossword Generation	42
4.2.1.3	Frontend and Backend Interacting with User	42
4.2.2	Word List Data Structure	42
4.2.2.1	Crossword Generation Worker	43
4.2.3	Deployment	44
4.2.4	Improvements Proposal	45
	Conclusion	47
	Bibliography	49
	A Contents of enclosed data medium	59

List of Figures

2.1	KorpusDB genres example. Source: KorpusDB [1]	13
2.2	Score of Bayesian ridge method based on subset of most popular words	16
2.3	Score of various regressors in the KorpusDB categories	17
2.4	Example Knowledge Graph containing Keynodes K_1 and K_2 and wordnode W	18
2.5	Lengths of words obtained from Czech Wikidata Articles	20
2.6	Lengths of words obtained from Czech Wikidata	21
3.1	Crossword filling step example	27
3.2	Heuristic success rate on a 6x8 instance	30
3.3	Heuristic success rate on a 10x15 instance	30
4.1	User Interfacing App Architecture	35
4.2	Frontend Crossword list	36
4.3	Backend entity-relationship diagram	37
4.4	Frontend Grid Designing	38
4.5	Frontend Setting the Categorization	39
4.6	Frontend Example of the generated Crossword	40
4.7	Diagram of the application parts	41

List of Tables

Introduction

“A crossword is a word puzzle that usually takes the form of a square or a rectangular grid of white- and black-shaded squares. The game’s goal is to fill the white squares with letters, forming words or phrases, by solving clues, which lead to the answers.” [2]

Crosswords are often present in newspapers or magazines as a relaxing activity, sometimes combined with other puzzles. The most important property of the crossword is that the crossing words share letters, allowing the solver to uncover additional letters of the words he is unsure of. The letters are usually case-insensitive.

In this Thesis, I developed a set of programs automatizing the crossword generation process completely. The Crossword Generation can be parameterized with the preference of word categories. I believe this will allow the generated crossword to come closer to the crossword crafted by a human in terms of dramaturgical quality and playability — simply that the generated crossword will be more fun to solve.

The ability to generate attractive crosswords automatically will allow more people to enjoy the puzzle. It will allow teachers to create crosswords with words relevant to the subject, it will allow crosswords with custom words to be used as a gift or advertisement, and it will enable the crossword puzzle to reach new markets. Solutions developed in this work might be adapted to other languages with only a little effort.

Work structure

This Thesis is divided into three parts.

Word List Generation Chapter 2 discusses the possible word data sources. I design and implement a program to obtain the words with their definitions from Wikipedia dump and Wikidata database. Moreover, I categorize obtained words based on the publicly available data to allow the resulting crosswords to be customized based on given preferences.

Crossword generation Chapter 3 describes the problem of constructing the crossword — placing suitable words to the grid so that all letters match. The problem is NPC. I implement a well-known heuristic with a modification to respect the user preferences.

Web Application Chapter 4 focuses on development of a user interface. The user interface allows users to submit crossword requests and obtain results constructed crosswords. This chapter also discusses the overall architecture of the application, its scalability, and deployment.

Business Analysis

1.0.1 Crossword Classification

1.0.1.1 Interlocking

The highest possible interlocking means that every single letter in every word is shared with another word. This is a common property of the *American-style* crosswords. The crossword is easier for the solver as they are able to fully fill any unknown word if they are able to fill in the crossing words.

The *Swedish-style* and *British-style* grids are less dense than American style, with British style grids having “*roughly half of the letters in each word are crossed over with another word*” [3]. This property can be easily computed for a given grid and we may define

$$\text{interlocking} = \frac{\text{number of crossing letters}}{\text{number of letters}} \quad (1.1)$$

which makes $\text{interlocking} \in [0, 1]$.

The higher the interlocking, the easier is the task for a solver. With $\text{interlocking} = 1$ it is enough to get the words from one direction, and the crossword would be fully filled. The task is harder for the *crossword creator* because the words must fully match in the second direction. We will discuss this more in Chapter 3.

1.0.1.2 Clues

The clue should lead a Solver to fill in a proper word. Some of the common types observed are:

- Semantic Superclass (e.g. “means of transport” → “CAR”)
- Part of speech (e.g. “verb” → “RUN”)
- Synonym (e.g. “incredible” → “AMAZING”)

1. BUSINESS ANALYSIS

- Common algorithm (e.g. “2002 in latin” → “MMII”)
- Translation (e.g. “World in Spanish” → “MUNDO”)
- Obvious abbreviation (e.g. “United Nations, abbrev.” → “UN”)
- Hidden abbreviation (e.g. “Abbrev. of a football association” → “FIFA”)
- Knowledge fact (e.g. “Zola’s novel” → “NANA”)
- Knowledge fact about geography (e.g. “Lake in the Andes” → “TITICACA”)
- Name of a famous person (e.g. “Name of di Caprio” → “LEONARDO”)
- Fill-ins of common phrases (e.g. “Mamma ___” → “MIA”)
- Other Wordplays (e.g. “music instrument without leading P” → “IANO”)
- Compound clues (e.g. “Iran Iceland” → “IRIS” (using country codes))
- Image clues

The adoption of those clue types depends on the culture and current crossword puzzle trends. Some of those types are completely unseen or deprecated in some cultural contexts. Sometimes the usage of a specific clue type is seen as an entirely new crossword type — for example, there seems to be an explicit classification in British-style crosswords according to [4]:

- Quick crossword — You do not need any special knowledge or skills to attempt a quick crossword.
- Cryptic crossword — Cryptic clues are written in the form of code — they contain hidden instructions that tell the solver how to arrive at the answer.

Another good example would be the image clue crosswords popular in children’s magazines.

1.0.1.3 Grid Style

There are several common styles of the crossword grids and their visualization — colors and clue placement.

- American grid — every single letter in every word is crossed over by another word — *interlocking* = 1
- British grid — the grid is less dense, roughly half of the letters in each word are crossed over with another word, no two-letter words are allowed
- Swedish-style grid — *interlocking* = 1, the clues are written in the non-letter cells in the grid.

1.0.1.4 Secret

The crossword might contain a secret message that the solver’s task is to reveal. It might be placed into a crossword similarly to a normal word without a clue (which is common to Swedish crosswords), or it might be by numbering some of the letters in the crossword — after the crossword is filled, the secret message can be read as the sequence of those numbered letters.

Again the existence of a secret highly depends on the culture. In the Czech context, the placement of word-like secret phrases into Swedish crosswords is very common. Sometimes the secret message can be the point of a cartoon joke; sometimes, it can complete an attached cooking recipe. Contests of sending the revealed secret are also popular. It is common for a secret to be a multiword or a sentence with spaces and special characters omitted (“FOREXAMPLETHIS”).

1.1 Problem Definition

1.1.1 Vision

The goal of my work is to create a crossword puzzle generator accessible as a web application. The generator should not rely on an external or a user-provided dictionary. Instead, it should obtain words and clues from the open data sources — such as Wikipedia or Wikidata.

The use of Linked Data will allow us to generate clues from the knowledge graph itself. The clues might then be adapted for different target crossword styles and audiences. It should also allow for easy translations.

1.1.2 Constrains of this thesis

I will focus on getting to the minimum viable prototype of the whole pipeline. Design choices will be made to support the project’s extensibility. I will perform the experiments on a *Czech* language Word List.

1.1.3 Business

Some of the newspaper or magazine crosswords are still constructed by people, and it might produce the best results, as the joy for the solver relies heavily on a cultural context, shared knowledge, unwritten rules, and dramaturgy.

However, crossword constructing is such time-consuming activity that the professionals use at least specialized helper programs to find suitable combinations. As the demand for the crossword puzzles is still very high, many of the crosswords published seem to be generated fully automatically or with just a little contribution from the editor.

1.1.3.1 Competition

According to the New York Times puzzle makers [5] there are several usable programs suitable for augmented crossword design: commercial CrossFire [6] and Crossword Compiler [7] and open-source Phil [8].

There is also a lot of free online services, their limitation being a small or nonexistent dictionary. Usually they provide only low interlocking crosswords.

Word List Generation

2.1 Introduction

2.1.1 Problem Definition

2.1.1.1 List Of Words

A list of words is needed to construct a crossword. Words should contain the permitted alphabet only. For our case, we forbid multiword phrases and strings containing punctuation marks or numbers. Also, we look at the words case-insensitely. We show that a rich word list is of crucial importance for crossword generation as we measure in Chapter 3 so we will implement or propose multiple data sources here.

2.1.1.2 Clues

Although a crossword can be generated without clues, it is essential for the user experience and our business case to provide a well-defined clue to every placed word. We have listed some of the common crossword clue types in Section 1.0.1.2. For the prototype, only the Synonyms and Semantic Superclasses will be used as those are most easily found as readable text. I will also propose a way to generate clues out of linked data.

2.1.1.3 Categorization and Difficulty

This work's novel approach to crossword solving is to allow the user to choose the categories of words that they want to prioritize in the crossword construction process. I believe it is a way to imitate the human crossword maker's dramaturgy and consistency.

This can also enable us to create relevant classroom materials — crosswords with most of the words relevant to the taught subject. Of course, the ability to pick the word with the preferred category and following the crossword constraints intensify the need for a rich Word List.

2.1.2 Internationalization

Data sources that we use are multilingual. That does not mean that every word or sentence exists in every language, but that the data are structured consistently among the language versions, and data extraction can be parametrized by language.

The work uses *Czech* as the language for experiments. However, the implementation tries to stay language independent. The tools developed might be easily used for obtaining the word list in another language and produce crosswords in virtually any language.

2.2 Analysis

2.2.1 Linked Data

The term *Linked Data* is used for “structured data interlinked with other data so it becomes more useful through semantic queries”. [9] Linked Data use URIs for naming of things. Various relations between the data are published as (subject, predicate, object) triples. This data model is named RDF and specified by W3C [10].

Graph Database software such as Apache Jena, Blazegraph, Eclipse RDF4J, and OpenLink Virtuoso can be used to store and query millions of triples. Those databases might be queried using SPARQL [11].

Linked Data published under permissive licenses are called Linked Open Data (LOD). By reusing entity URIs from other LOD repositories or by providing information about identities (`owl:sameAs` predicate), data from multiple repositories can be interconnected. The term **Knowledge Graph** is sometimes used for a Linked Data repository.

2.2.2 Knowledge Graphs

Most of the words in a crossword should be known to the person solving. As we do not have any assumptions about user proficiency, the words should ideally be commonly known. That is why I considered the *crossdomain* knowledge graphs.

There are some proprietary Knowledge graphs — like Google Knowledge Graph, WolframAlpha, or Facebook Graph but data gathering from these is problematic, and the data license makes the data unusable.

Open knowledge graphs, on the other side, have permissive licenses — either a CC0 or a CC-BY-SA and usually provide accessible endpoints and even dumps of the whole database. The main active cross-domain open knowledge graphs are DBpedia, Wikidata, and YAGO.

2.2.2.1 DBpedia

DBpedia [12] was created in 2007, which makes it one of the oldest and most respectable Linked Data sources. DBpedia extracts the structured texts in the Wikipedia project into Linked Data.

2.2.2.2 Wikidata

Wikidata [13], established in 2012, is a collaboratively edited multilingual knowledge graph. The primary use for data is in Wikimedia projects. Wikipedia is becoming the source of the info-boxes found in Wikipedia articles, pushing the Wikipedia editors to edit facts structurally in the Wikidata database.

2.2.2.3 YAGO

YAGO4 [14] combines data from Wikidata with widely used vocabulary Schema.org [15]. Yago additionally filters the data enforcing logical consistency. The previous version, YAGO3, used DBpedia instead of Wikidata as a base.

2.2.3 Previous Work

The word clue list was obtained by [16] from Italian Wikipedia using Natural Language Processing. [17] uses DBPedia structured data to categorize the questions in DB-Quiz, a clone of the popular Czech TV game called AZ kvíz. Cloverquiz [18] developed a mobile game with multichoice questions based on DBPedia categorization. Ferdinand Mütsch [19] constructs the questions directly from the DBPedia facts but has a problem with latency as he generates the questions on the fly.

2.3 Data Sources

2.3.1 Linked Data: Wikidata

2.3.1.1 About the Choice

After an analysis of the current state, I have chosen the Wikidata as a source of the Linked Data used in this project for the following reasons:

- Wikidata contains more data than DBpedia — Wikidata 12bn [20], DBpedia 9.5bn [21] triples.
- Wikidata is the source of data for Wikipedia articles but it does not limit itself on Wikipedia articles.
- Wikidata provides a single multi lingual database dump while DBPedia can be replicated by running quite complicated Extraction process.

2. WORD LIST GENERATION

- Wikidata’s license is more permissive (CC0) — that mean using of the data does not need an attribution.
- The feedback of our users (incorrectness report or a label fix) might be used to improve Wikidata.

2.3.1.2 Local Replication

Public SPARQL endpoint of Wikidata enforces limitations [22]: long-running query is terminated 60s and one agent is allowed 60seconds of processing time each 60seconds. For this reason a local replica of the wikidata database has been created by following [23]. The system used was Ubuntu 20.10 [24] running on AMD Ryzen 5600X with 64GB RAM and 2TB M.2 NVMe SSD storage.

- Install Docker Engine [25]
- Pull the wikibase/wqds docker image [26, 27]
- Download the Wikidata TTL dump from the Wikimedia dump archive [28]
- Start the docker container with bind mount of the host machine directory containing the dump
- Access the docker container using `docker exec`
- Inside the docker container run the `munge.sh` script. The script runs for approximately 20 hours.
- Run the docker container with entrypoint `runBlazegraph.sh`
- Inside the container run `/wdqs/loadData.sh` script that would load the munged data. This runs for several days.

The process of Wikidata import is machine-time consuming. I have also measured the disk writes — 18.1TB was written, using 1.4% of that particular SSD TBW guarantee. The imported database uses 693GB of storage. The process, however, can be scripted and run periodically (i.e., monthly). Original author of [23] used Google cloud infrastructure, although no pricing estimate is presented.

After the Local Replication finishes, running the docker container will provide a sparql http service on `localhost:9999/bigdata/namespace/wdq/sparql` usable to work with the local instance of Wikidata. The endpoint is limited only by the available computer power and usable RAM. Tunning `HEAP_SIZE` environment variable and `analyticMaxMemoryPerQuery` might be necessary for long queries to run properly.

2.3.1.3 Clue Generation

For purposes of this work, the simple approach of using the strings obtained from `schema:description` triplets seems to be satisfactory. The descriptions are brief, syntactically correct, and often cover the most important features about the subject — usually notable categorization, such as “French Town”, “a kind of insect”, “sports contest”, etc. Another way the clues are obtained is by multiple `rdf:label` and `rdf:altLabel` on the same subject. This is the way to obtain the synonym clues, such as “bias”~“systematic error”.

2.3.1.4 Categorization Use

Categorization structure can be found in the Wikipedia Knowledge Graph by following triplets with following predicates:

- P1628 — Equivalent
- P31 — Instance of
- P279 — Subclass
- P361 — Part of
- P921 — Main subject

Yet, most of the category nodes cannot be used for this thesis objective as user relatable thematic areas. The categories are either too wide (Q5 Human has nine million instances linked) or too narrow. Many different categorization concepts overlay in the graph, making it impossible to find the concept root category. A machine learning solution for this problem using the Wikidata graph along with another dataset was implemented, and it is described in Section 2.3.5.

2.3.2 Wikipedia

Apart from Linked Data, Wikipedia itself has some use for this work:

- Wikidata does not contain links to Wikipedia articles, making it hard to provide a link to the Wikipedia article when needed in the application
- The Wikipedia page metadata might be usable.
- The pageview statistic of the Wikipedia article might be obtained.
- We may expand our synonym list by using Wikipedia article redirects.
- Roughly 25% of the articles in the Czech Wikipedia do not have the corresponding Wikidata entity. Hopefully, this ratio will decrease with the Wikidata project growth.

There are the downsides of fetching the Wikipedia articles:

- Probably it just duplicates DBpedia algorithms
- Wikitext parsing and NLP is needed to use any part of the article text. It is language-dependent, and current implementation does fail a lot — creating descriptions containing fragments of templates and thumbnails.

The number of suitable crossword words obtained by parsing Czech Wikipedia is 105003. However, most of them were already retrieved from Wikidata. The number of entirely new words is 16380, which is around 6% of the total word count.

2.3.2.1 Parsing Wikipedia Article Dump

The *pages-meta-current.xml* is an xml structure where the Wikipedia Article texts are saved. The file is distributed in the .bz2 archive, and the compressed size for Czech cswiki is approximately 1GB. Stream parsing is possible to avoid excessive RAM consumption.

An example function to count the xml articles using the bz2 and lxml python libraries follows. The code for the actual parsing is a bit more complicated — but basically it is a state automaton driven by the `start` and `end` events invoked when xml stream reaches the opening or closing of a tag.

```
def clear_element(element):
    element.clear()
    while element.getprevious() is not None:
        del element.getparent()[0]

pages_xml = bz2.open('cswiki-20210201-pages-meta-current.xml.bz2', 'r')
cnt = 0
for event, element in etree.iterparse(pages_xml, events=["end"]):
    if element.tag.endswith("page"):
        page_count += 1
    self.clear_lxml_element(element)
print(page_count)
```

Additional relevant dump files are:

- *redirect.sql.gz* — contains redirects between articles
- *page_props.sql.gz* — maps from Wikipedia articles to Wikidata entities (e.g. “Douglas Adams” → Q42)

Those are in gzipped sql format. After decompressing, I used a small transformation and was able to load them using Sqlite3 [29], which was then queried by Python.

2.3.3 Wikipedia Pageviews

The Wikimedia REST API [30] is used to fetch the number of page views of Wikipedia Articles found in the previous section. The numbers can be used to categorize words by the popularity. The problems here are that:

- This can be done only on the part of the words that were found as Wikipedia articles (37% — 105003 out of 283415)
- The number of page views might give the only estimate of the word popularity as a lot of other factors might affect the visitor rate — such as the encyclopedic activity of corresponding subcommunity, article quality (resulting in more search hits), etc.

2.3.4 Korpus.cz

The categorization is implemented by using KorpusDB [1]. Accessible via undocumented public API, KorpusDB finds a lemma for a given string (a crossword word) and provides:

- Frequency in whole corpus — usable to determine the word popularity
- Frequency distribution of the word among 34 genres

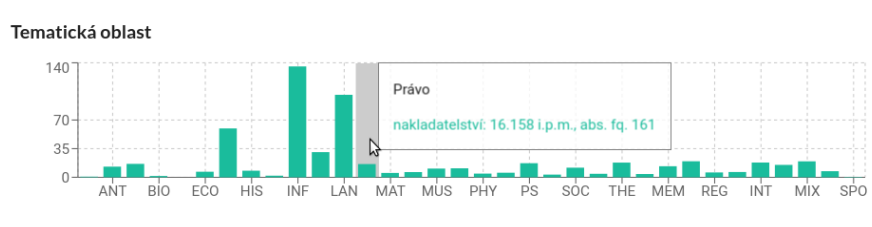


Figure 2.1: KorpusDB genres example. Source: KorpusDB [1]

The genre data are obtained using the corpus structure attributes [31]. That means every document in the corpus is tagged with the genres based on its source (e.g., focused magazine, newspaper section). A single word genre frequency distribution is then computed by averaging all word occurrences sources distribution. The categorization is noisy, but the maximums seem representative for words with enough (> 100) corpus occurrences. The main data source is the Czech corpus SYNv8 containing over 8 million lemmas.

There are downsides to this:

- My current implementation of the search is unable to distinguish between homographs by the word meaning. Corpus lemmas are capable of such distinction, so it might be possible at least to detect this problem.

2. WORD LIST GENERATION

- Lot of words is not recognized in the corpus, or the occurrence is so rare the categorization is not dependable. Out of the 283415 words, only 39223 was present in the corpus, 24740 of them with occurrence higher than 100.

The data incompleteness will be addressed in Section 2.3.5. For the Wikidata sourced words, we have corresponding multilingual Knowledge graph nodes, so this categorization might be used for other languages as well.

2.3.5 Linked Data Benefit: Machine Learning

We have discussed that the naive algorithm for the categorization Section 2.3.1.4 is not suitable for the needs of this project. In the previous section, we were able to fetch useful categorization information, but only about a fraction of the words needed. With the assumption that there lies some information in the Knowledge Graph structure, I performed an experiment in the machine learning field.

We would be using the presumed coincidence of position within the Wikidata Knowledge Graph structure with the categorization fetched from the KorpusDB. As we were able to obtain the KorpusDB categorization only for 10% of the words, the target of this is to derive the categorization for the rest 90% of words from the known position in the Wikidata graph.

I filter the edges only to the list of predicates stated in Section 2.3.1.4 — Equivalent, Instance of, Subclass, Part of, and Main subject relations. The distance between a node representing a word obtained from Wikidata and a certain set of key nodes will be examined. My assumption is that some key nodes will generally be nearer to the nodes categorized in a specific category — and further from nodes not categorized in the category.

2.3.5.1 Key Nodes Selection

```

Data: Words with known categorization and a graph node, List of
          Categories, K, N
Result: Set of key nodes
KeyNodes = empty set
for Cat in Categories do
  | Words = words categorized in Cat
  | Neighbours = empty list
  | for Word in random sample of Words do
  | | Node = graph node of Word
  | | NodeNeighbours = nodes reachable from Node in  $\leq K$  steps
  | | Append NodeNeighbours to Neighbours
  | end
  | CategoryKeyNodes = find N elements with maximal occurrence
  |   in Neighbours
  | Append CategoryKeyNodes to KeyNodes
end
return KeyNodes

```

Algorithm 1: Choosing the key nodes

Suitable values $K = 2$, $N = 3$. Algorithm would generate at most $N \cdot |\text{Categories}|$ key nodes.

2.3.5.2 Machine Learning Problem Definition

Categorization of a word is vector v of length = number of *Categories*. v_n quantifies the membership of the word into n th category. $v_n \in (0; 1)$. *categorization* is normalized, meaning $\sum_{n=1}^{|v|} v_n = 1$. Graph distance of a word is vector w of length = number of *KeyNodes*. w_n is the shortest path distance between the word node and n th key node.

I will use the scikit learn framework [32]. The problem is multi-class (classification into many categories), multioutput (as the output is fuzzy membership $\in (0; 1)$) [33] regression. The number of algorithms supporting this setup is very limited — I have tried the *DecisionTreeRegressor* with poor results. Instead, the following tricks can be used to reduce the problem:

1. Instead of Categorization vector use only maximal value. This changes the problem to classification — for a word with a given graph distance-vector, we want to find one category where it suits the most.
2. We use the regression on each of the classes separately.

The first solution was tempting for KorpusDB data at first. The maximum would filter out corpus noises — weak categories would be ignored. However, it seemed to fail on words with two and more strong categories — the winner

2. WORD LIST GENERATION

takes it all regardless of the closeness of the next competitor category. The learning process was seriously tainted by that problem. Probably the approach would work if there were less than 34 categories that KorpusDB provided. The second solution has some problems too:

- the independency of the regressors returns a vector that is not normalized
- the categories common among the words might get advantage
- learning result evaluating gets complicated

However, I like the versatility of the solution finally implemented.

2.3.5.3 Method Choice

Words with the known KorpusDB categorization were split into train and test set by half. The *Coefficient of determination R^2 of the prediction* provided by the scikit-learn `score` method was used to evaluate the regressor quality. I also experimented with selecting only the most popular words. This trick filtered out the noise caused by uncommon words being incorrectly categorized due to low occurrence in corpus. The sweet spot for my data was identified as

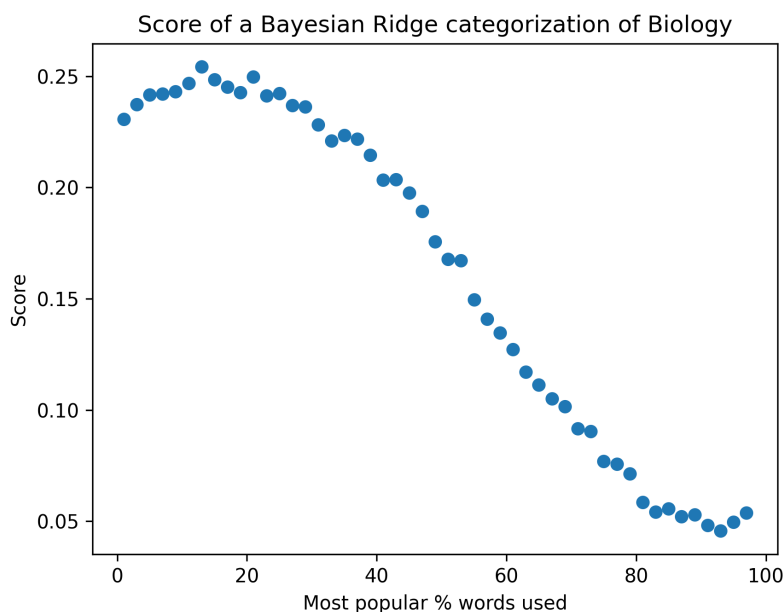


Figure 2.2: Score of Bayesian ridge method based on subset of most popular words

selecting the 20% most popular words — roughly 8000. With the 20% most popular words I did the comparison of the various regressors methods provided by scikit learn:

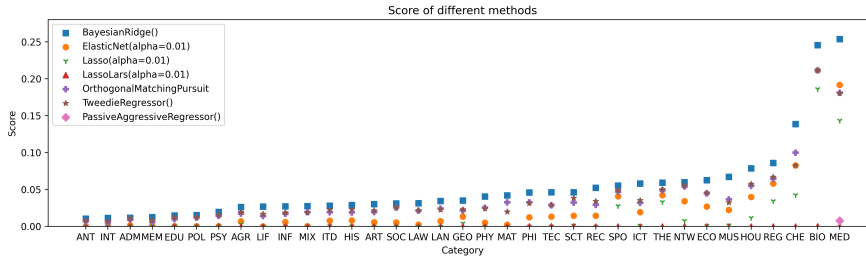


Figure 2.3: Score of various regressors in the KorpusDB categories

2.3.5.4 Performance

Theoretically, SPARQL might be used to query the node distances on the specified edge list — especially with Property Paths [11]. In the case of Blazegraph running the Wikidata Knowledge Graph, I have tried and failed: either the RAM was exhausted, or the computation did not finish in a reasonable time.

I developed the following tricks:

- SPARQL queries must be reasonably simple. When I tried to fetch all the connections in depth 2 the query failed for nodes close to nodes with high degree (e.g., Human(Q5) node has degree 9 million) — blazegraph consumed 64GB RAM and 200GB of the available swap, then crashed.
- SPARQL queries must not be too simple: querying the Knowledge Graph for every node’s close neighbors separately had a big performance overhead. I ended up setting to query the sparql endpoint to provide a list of close neighbors for 100 nodes in batch.
- I have resigned on long paths: limiting the path length discovered between the word node and key node to k and treating everything longer as a $k + 1$. The shorter the paths between the word node and key nodes, the more important the connection is.

The key nodes neighborhood limited with a certain depth(D) is fetched only once. The in-memory graph is created with the NetworkX library [34], then converted into a Pandas [35] Dataframe used as a graph distance matrix describing the distances between key nodes and all the neighboring nodes. This dataframe occupies only a fraction of the space needed by the graph representation.

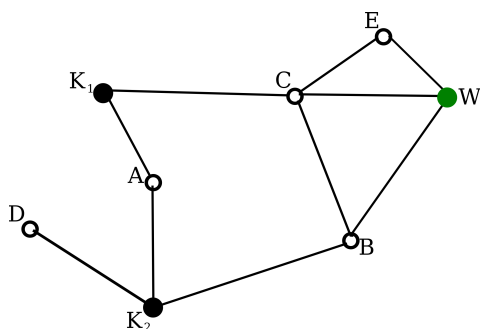


Figure 2.4: Example Knowledge Graph containing Keynodes K_1 and K_2 and wordnode W

Key nodes matrix for $D = 1, K = 1$:

	A	B	C
K_1	1	<i>nan</i>	1
K_2	1	1	<i>nan</i>

For every Word Node we perform the neighborhood discovery — gathering all nodes reachable in the distance K and turning them to a similar distance matrix.

	B	C	E
W	1	1	1

Now we select all the nodes found in both matrices and sum Word node distance with every Key Node row.

	B	C
K_1	<i>nan</i>	2
K_2	2	<i>nan</i>

The minimum value of a row is the shortest distance to the corresponding Key Node. The computation is performed by pandas, and it scales well. With this algorithm, I am able to compute the shortest paths in the Wikidata Knowledge Graph in 2s per word on average.

2.3.5.5 Extensibility

By this experiment, I have proved there is some information in the Wikidata Knowledge Graph structure usable to complement a partial categorization. I believe this process might be easily adapted and used with other sources

of categorization. One such source would be a manual annotation of several hundred or thousands of words. Such categorization might be costly in human time but might be more suitable for the goal.

2.4 Other Data Sources Proposals

Currently implemented data sources described in the previous section are Wikidata and Wikipedia Articles. Although Wikipedia is an excellent source of knowledge, the word suitability for the crossword puzzle generation is limited — there is a lot of named entities that tend to be quite obscure and unknown for the solver.

2.4.1 Wiktionary

In currently implemented word list verbs, adjectives and adverbs are missing almost completely. When the words are present, the clues are quite misleading. The enhancement will be possible using the open multilingual dictionary Wiktionary [36]. The words have a definition that might be used as a clue. In inflectional languages there is also a lot of word forms connected with their lemmas — base forms. However, Wiktionary is only loosely structured, so it will need a lot of parsing.

2.4.2 Corpus

We have used a Czech Corpus for the categorization data. But with proper tools, it might also be used as a word source - using the structure information about the syntax (word category as a clue), collocation (most used phrases containing the word), and synonyms. The downside of this data source is that it would be separately developed for the various platforms of national language corpora.

2.4.3 Open Street Map

Although a lot of the recognizable features are already obtained from Wikidata, acquiring the data from Open Street Map can benefit this project by additional metadata and also by using the geographical features near the user, like hometown streets or nearby lakes. The current project pipeline does not, however, allow such user Word List customization, so this might be the wrong way.

2.4.4 Abbreviation Generator

A lot of short words are needed for generating the crossword, so it is common there are the abbreviations present in the crosswords (e.g., United States of

2. WORD LIST GENERATION

America \Rightarrow USA). Some abbreviations are covered by the current implementation as they are present in the Wikidata labels. However, it would be good to provide a way to detect such cases (possibly adding (*abbr.*) hint to the clue). Custom abbreviations could also be generated from the multiword phrases, although the results might be surreal.

2.4.4.1 Generate Clues From Linked Data

I am not using the most exciting feature of Linked Data — the possibility to generate custom clues from the graph structure. Such generation will probably require templates mapping between SPARQL and natural language, and it will be a topic of my following research.

2.5 Results

2.5.1 Exported Data Format

The output format of the Word List Generator depends on the Crossword Generation application, so it will be discussed later in Section 4.2.2.

2.5.2 Obtained words

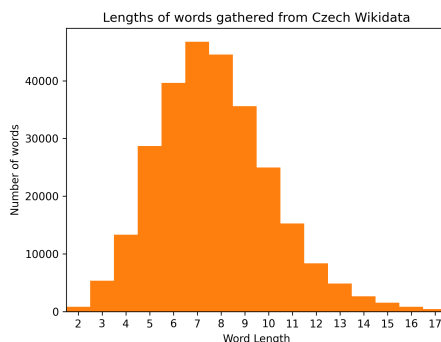


Figure 2.5: Lengths of words obtained from Czech Wikidata Articles

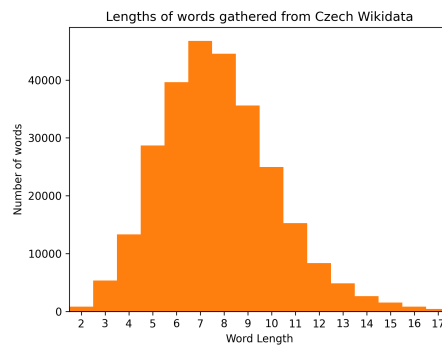


Figure 2.6: Lengths of words obtained from Czech Wikidata

Crossword Generation

3.1 Analysis and Design

3.1.1 Formal Definition

Grid A is a boolean matrix of width n and height m . *Crossword* C is a matrix of the same dimensions as its grid. The grid specifies where the letters from an alphabet Σ will be placed in C . Every cell in crossword is either a *letter cell* ($C_{i,j} \in \Sigma$) if $A_{i,j} = \text{true}$, or an *empty cell* ($C_{i,j} = \epsilon$) if $A_{i,j} = \text{false}$.

A word space describes an area of a crossword and spans for two or more cells of C either horizontally or vertically. Word space is constrained by an empty cell or the matrix border. A word space can be identified by the beginning (minimal) coordinates inside the crossword, direction (either horizontal or vertical), and its length (number of cells covered).

We say the cells inside the word space area *belong* to the word space. Cells belonging to the word space form a sequence starting at the beginning coordinates of the word space. We may address the cell on beginning coordinates as the first cell, next cell as the second cell, and so on. When a cell belongs to two word spaces W and V . The cell is the *shared* cell of W and V . Those word spaces are in relation *cross*: $W \boxplus V$

Constraints for a grid might be found in Engel2012 [37]. I have adapted them to suit our definitions:

1. Each letter cell is to be part of at least one-word space.
2. Each word space has to span over at least two letter cells.
3. Each word space is to be enclosed in between two empty cells.
4. No two horizontal or two vertical word spaces may overlap.

When a grid satisfies those constraints, it is considered *valid*. For a valid grid, the word spaces might be derived conclusively. The set of word spaces derived from G is $\text{WordSpaces}(G)$

Another common constraint of a Grid is that the crossword is a single component. That means for every word space W , word space V there exists some sequence of word spaces beginning with W and ending with V having \boxplus relation between every two consecutive elements of the sequence.

3.1.2 Constructive Combinatorial Problem

Word is a finite sequence of the alphabet elements $\text{Word} \in \Sigma^*$. Word can be *filled into* a word space if the word length equals word space length. *Filling* a word *into* a word space means the word letters are assigned to respective word space letter cells: The first cell of a word space gets assigned the first letter of the word, and so on.

When a valid grid G and a set of words ($\text{Words} \subseteq \Sigma^*$) is given, construct a function $M : \text{WordSpaces}(G) \mapsto \text{Words}$ so that the words can be filled into corresponding word spaces and every crossword letter cell has all its assigned letters equal.

M goes from a finite domain into a finite codomain. Therefore the number of possible functions is finite. That makes the problem combinatorial. The number of possible combinations is upper bounded by $|\text{Words}|^{|\text{WordSpaces}(G)|}$ [38] The G and Words form the problem *instance*. Function M is a *solution* of the problem when it satisfies the constraints. [39]

3.1.2.1 NP Complete

Engel et al., 2012 [37] proves that Crossword Construction is a NP-complete problem. The proof is done by “*reduction from the well-known NP-complete problem 3SAT*”. Another proof is in Peterson [40] by Exact Cover by 3-Sets to Crossword Puzzle Construction reduction.

3.1.3 State of the Art

One of the oldest works on the Crossword Construction problem is Mazlack, 1976 [41]. It provides interesting performance-optimizing details. The method he gets to work is a letter-by-letter filling. Ginsberg et al., 1990 [42] changes this to a word-by-word approach and introduces a *lookahead* technique: the list of possible word choices for every word space is maintained during the search algorithm run.

Berghel, 1987 [43] gave a purely logical solution to the problem written in Prolog. The solution is clear and elegant, however, the solution was not practically used much because of performance issues — the heuristic can be done only by the clause ordering and fails beyond the runtime adaptability (I have experimented with the Prolog solution too).

Constraint Programming Lessons Learned from Crossword Puzzles(2001) [44] gives a comprehensive comparison of seven data models used with eight back-tracking algorithms with the following conclusion:

1. the form of the CSP model is important
2. the choices of model, algorithm, and heuristic are interdependent, and making these choices sequentially or assuming a level of independence can lead to non-optimal choices
3. to solve a problem using constraint programming most efficiently, one must simultaneously explore the space of models, algorithms, and heuristics

Bonomo et al. [45] present a Genetic Algorithm enhanced with Wisdom of Artificial Crowds method. Using a Genetic algorithm or Simulated annealing to solve the Crossword Constructing problem was out of the scope of this thesis. However, I would very much like to expand the project in this direction.

Another interesting approach is the reduction of crossword generation to some other well-known NPC problem that can be solved by specialized optimized solvers. That's what popular [5] online program Phil [8] is doing. Phil is using a Glucose SAT solver [46] running inside the user's browser through the Web Assembly [47].

3.1.3.1 Variation: Unconstrained Variant

When the grid is not specified, and only the target crossword dimension is given, the constructing algorithm may enlarge or shorten some word spaces in order to be able to fill in the words. This is called the Unconstrained variant. The problem was first stated by Harris, 1990 [48] A heuristic algorithm is in Kuzma, 2009 [49]. Engel et al., 2012 [37] proves the variant is NPC as well and proposes a construction algorithm. Latest work in the field Agarwal et al., 2020 [50] provides an overview of the heuristic strategies on the Unconstrained Crossword, along with detailed measurements.

3.1.3.2 Variation: Solving the Crossword

There is also a few works focused on *solving* the Crosswords: Littman et al., 2002 [51] use a probabilistic method, Dr.Fill [52] works by converting the Crosswords to Singly Weighted CSPs.

3.1.4 Optimization Combinatorial Problem

I want to propose an optimization variant of the Crossword Constructing problem (CC): **Weighted Crossword Construction (WCC)**.

When a valid grid G and a set of words with weights ($\{(word, weight); word \in \Sigma^*, weight \in \mathbb{R}\}$) is given, construct a function $M : WordSpaces(G) \mapsto Words$

so that the words can be filled into corresponding word spaces, every crossword letter cell has all its assigned letters equal and the sum of weights of the words filled into the word spaces is maximal.

3.1.4.1 Problem is NPH

The weighted crossword construction is NP-hard. I will prove that using definitions from [53].

A polynomial-time Turing reduction can be constructed between CC and WCC: Given a WCC solver as an oraculum and any instance of CC problem, the trivial weight 0 may be added to all the Words of CC instance. Provided with that modified instance, the WCC oraculum returns an optimal solution that is also an acceptable solution to CC.

That proves $CC \leq_T^P WCC$. Given that the Crossword Construction is NP-complete, the Weighted Crossword Construction is NP-hard.

To my knowledge, this optimization problem has not been presented among previous works on the crossword construction topic.

3.2 Realisation

3.2.1 Heuristics: Decision Problem

I have adapted a well known heuristic summarized by [54]. The algorithm uses the lookahead technique described in [42]. The algorithm is implemented in Python3.

```
Data: WordSpaces, Words
Result: WordSpaces filled in with Words
Fill in a random WordSpace with a random Word of the same length
for unfilled WordSpace from WordSpaces do
  | Rebuild the Search Structure for WordSpace
end
for unfilled WordSpace from WordSpaces sorted according to the
  heuristic strategy do
  | if no best_option for the WordSpace then
  | | Backtrack = remove the Word fill from the WordSpace filled
  | | previously
  | end
  | BestOption = best_option for the WordSpace
  | Fill BestOption into the WordSpace
  | Rebuild the Search Structure for WordSpaces crossing
  | WordSpace
end
return filled WordSpaces
```

Algorithm 2: Heuristic for Constructing the Crossword Puzzle

3.2.1.1 Heuristic Strategy

The sorting priority determines the sort order of the word spaces. The sort order is crucial for the success of the heuristic. It must adapt to the fill situation during the runtime as the word spaces constraints change based on their crossing word spaces filling status.

But the criteria should be evaluated quickly. It might be better to go with an approximation instead of recounting the possibilities.

I implemented a compromise. Every word space has counts of words that can be filled in the neighbours after a specific letter is inserted. I have named this structure a `possibility_matrix`.

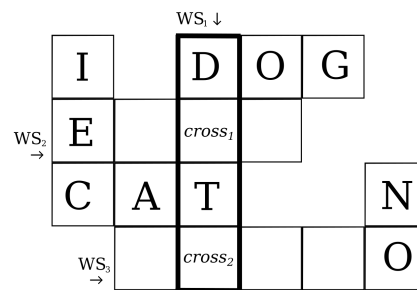


Figure 3.1: Crossword filling step example

Example for the situation shown in 3.1: We have WS_1 , WS_2 , WS_3 unfilled, however some of their crossing letters are already set. Possibility matrix of WS_1 :

	A	B	C	...	Z
<i>cross1</i>	30	42	60	...	0
<i>cross2</i>	560	60	80	...	20

That means that inserting a 'A' into *crossing1* would leave 30 possibilities for WS_2 - satisfied by four letter words with mask E.A.. Inserting 'Z' into *crossing1* would lead us to unfeasible solution branch.

The possibility matrix has to be recounted every time the word space is filled for the crossing word spaces - as their mask change. This causes the backtracking of my algorithm to be quite slow.

I have reached the optimal heuristic strategies, using the different aggregations on the `possibility_matrix`. I have started with a simple 6x8 instance, filtered out 8 promising strategies based on the ratio of successful/unsuccessful algorithm runs. The best strategies measured on 10x14 instance:

- Sort the word spaces by the minimal value of the mean candidate letter (min-mean) ascending (58% success)

3. CROSSWORD GENERATION

- Sort the word spaces by the sum of the sum of all candidate letters (sum-sum) in descending order (34% success)
- Sort the word spaces by the minimal value of the maximal candidate letter (min-max) ascending (50% success)

The last round on 11x15 instance confirmed the results with min-max 10% success rate, sum-sum 2%, min-mean 8%.

3.2.1.2 Best Option

`find_best_options` is called when the specific Word space is to be filled. It does a similar computation as the `possibility_matrix` recount, however does return the actual set of words with the maximal value of the heuristic strategy criterion. One word of this set is picked and returned as the `best_option`.

3.2.1.3 Search Structure

To allow fast possibility matrix recount and `find_best_option` search a data structure finding the set of words according to provided mask. Example of the query: Find all words fitting the `.A.D` pattern.

A multi level dictionary is used with sets inside the nodes:

$$words_structure[word_length][index_of_letter][letter] \quad (3.1)$$

The set intersection operation is used when performing a search. $.A.D \in words_structure[4][2][A] \cap words_structure[4][4][D]$.

A single retrieval from the dict is $\mathcal{O}(1)$. Retrieval count can be bounded by a maximum word length and we may consider that a constant by not allowing longer words than 20. The set intersection $s \cap t$ is average case $\mathcal{O}(\min(\text{len}(s), \text{len}(t)))$ according to ???. The set size is upper bounded by the word list size - but actually the word list is divided into disjunct parts by the word length observable in 2.5 and the letter frequency ???. That means the worst-case scenario of the operation might be $\mathcal{O}(|Words|)$, but the average case would be $\mathcal{O}(1)$.

Native Python structures `dict` and `set` are performing well in the terms of speed. There is a room of improvement for the memory consumed as the data structure has clearly big overhead - consumed memory is 25MiB per 10'000 words Section 3.3.2

3.2.2 Heuristic: Optimization Problem

I have implemented an extension of the heuristic to prioritize the words based on their weights during the search:

- Adjust the algorithm first step: Fill in a random *WordSpace* with a *Word* of the same length and maximal weight.
- Allow some randomization of the 3.2.1.1: the second best word space is picked instead of the first with some small probability.
- Adjusting the `best_option` : if more words are found sharing the same maximal value of the heuristic strategic criterium pick the one with the maximal weight.

The search algorithm is executed repeatedly for a constant amount of tries. The best found solution is picked as a resulting crossword.

There are few advantages to this approach: The multiple search runs may be trivially paralelized and the original heuristics was modified just a slightly. Additionally, the most time consuming process of generating the data structure 3.3.2 might run only once for all the search runs.

3.2.2.1 Improvements Proposal

The current heuristic might be improved by providing a feedback about the problem instance difficulty. It might choose between the current suitable word picking strategy and the strategy focused on maximalizing the weight. There also exist advanced probabilistic techniques such as Genetic Algorithm or Simulated Annealing that I would like to experiment with on this problem.

3.3 Results

3.3.1 Convergence

The heuristic backtracking is not very effective once the unfeasible branch is entered and it is necessary to restart the search in that case. In 3.2 and 3.3 the sucess ratio $\frac{\text{number of solutions found}}{\text{number of heuristic runs}}$ is observed. After reaching a word list size treshold, the sucess ratio is increasing almost linearly. It seems that for the 10x15 instance, the optimal word list size is more than 250'000 words.

3.3.2 Performance

The search structure described in construction time scales with the number of words linearly; I have measured 25*MiB* of RAM used per 10'000 words — using 604*MiB* for an entire word list. The time to build the search structure scaled with the number of words as well - I have measured roughly 1*second* per 10000 words for the data structure to be built.

3. CROSSWORD GENERATION

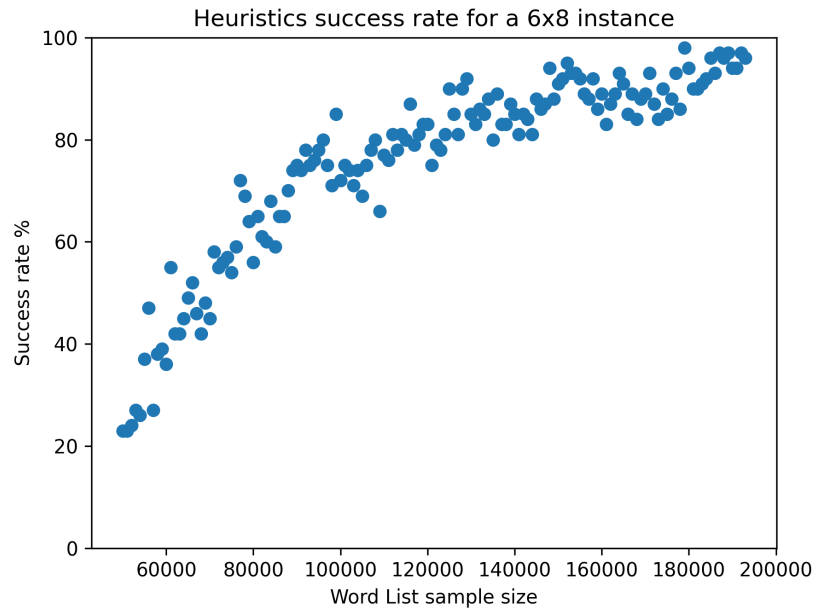


Figure 3.2: Heuristic success rate on a 6x8 instance

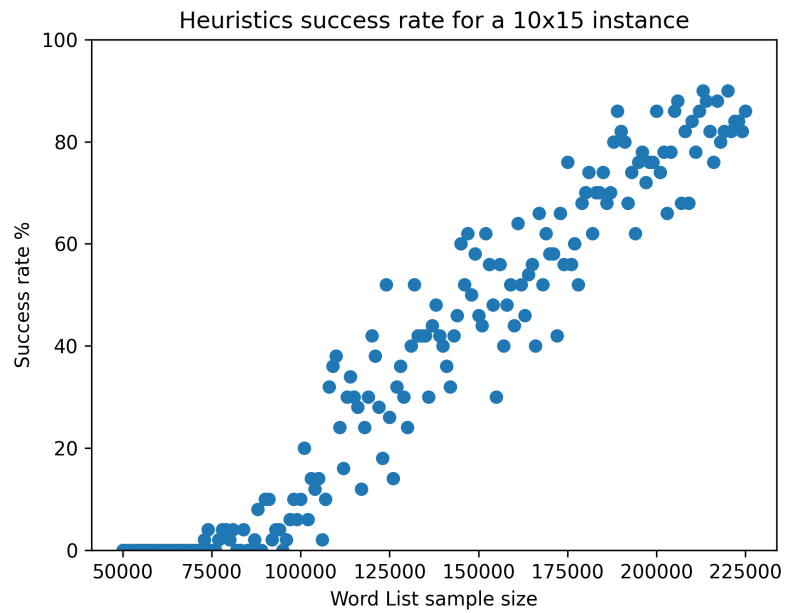


Figure 3.3: Heuristic success rate on a 10x15 instance

Web Application

In this chapter, I design a Web Application to interface the users. I also describe the architectural perspective of the whole system, interconnecting developed parts into a functional system.

4.1 User Interface

4.1.1 User Case Analysis

Solving Crossword puzzles is a popular leisure-time activity in Czech culture — according to [55], the three most popular magazines have each reached 200'000 people, which counts for 2% of the population.

However, it is important to realize the demography of those potential users: most of them are of senior age (older than 65 years). According to [56] 18% of 242 seniors likes to “solve crosswords, read books and magazines”. As per [57] 37.5% of 40 surveyed seniors in Brno states crosswords are a significant part of their activities.

4.1.1.1 Target Audience Scenarios

We can classify the potential users by the Reach of one Crossword and the Medium type.

	Single Consumer	Multiple Consumers
Electronic	Casual Web Game	Embedded on Blog, Distant Teaching
Pen & Paper	Printed on a home printer	Published in a magazine

4.1.1.2 One Consumer — Electronic

In this scenario, the frontend has to provide **interactive solving** of the Crossword. That means letter input optimized for the main mobile gaming platforms. This approach should focus on attractiveness and usability.

It would be possible to use many techniques known from casual mobile and web games — adjusting difficulty, achievements, a narrative story, interplays. Main competitors would be casual mobile and web word games. The business model of such project might be freemium. The demographics can negatively affect this case’s userbase size. The seniors are typically not a good target audience for the web and mobile gaming, often lacking the skills to use the application and preferring analog over electronic devices.

4.1.1.3 One Consumer — Pen & Paper

The User in this scenario visits the frontend application to obtain a Crossword to **print on a printer** at home or at work. The printing would occur through the browser printing dialog. It is necessary to test the application against the possible platforms and common printer types. The app should hint about the print-out possibility to attract users. Printing on paper might be viewed as unecological, yet it is still the most comfortable medium for Crosswords for many users.

4.1.1.4 Multiple Consumers — Electronic

In this scenario, the User is creating a Crossword to share via electronic media. The sharing may be through a link or via an embedded component (our html code that the User inserts into his website). Many of this sharing will occur on social networks. The need here is to optimize the sharing metadata (e.g., *og:image*) to make the social network previews attractive. This scenario also depends on the **interactive solving** capabilities of the application.

4.1.1.5 Multiple Consumers — Pen & Paper

The User is a publisher of paper media, e.g., a local newspaper or a magazine. In this scenario, perfect **export to common print formats** (PDF, PostScript) is needed, as well as the ability to edit and fix the word descriptions and inserting custom words relevant to the magazine scope. There might be a benefit in providing an API for the publishers, as they can integrate our product into their own pipelines. The obvious advantage of attracting this kind of customer is their willingness to pay significant money if the application meets their requirements.

4.1.1.6 Teacher

Outside of the reach/medium classification lies the group of users that are educating someone and want to provide their pupils a fun way to remember certain words. For success by this group, the main need is the ability to **own import words** as well as good electronic or print sharing capabilities discussed in previous scenarios.

4.1.2 Design

4.1.2.1 Feature List

Based on those user case scenarios, we can create the application feature list. Proposed feature groups are segmented into a sequence of *milestones*. The first milestone in each feature group represents the minimum viable product. The last milestone is usually the perfection far beyond this work scope. The **thick milestone** is the current state of the presented application.

Feature Group	Milestones
Word List Customization	<ul style="list-style-type: none"> • One static Word List • User preferences have effect • Custom words
Print to paper	<ul style="list-style-type: none"> • Default • Printed page styled via CSS • Optimized for multiple platforms and printer setups
Interactive solving	<ul style="list-style-type: none"> • No interactivity • Simple letters input • Input optimized for different platforms
Game Elements	<ul style="list-style-type: none"> • No game elements • Leveling • Story
Graphic Design	<ul style="list-style-type: none"> • Basic Layout • Carefully designed Visual identity (layout and colors) • Custom Artwork
Sharing capabilities	<ul style="list-style-type: none"> • None • Sharing a crossword through link • Sharing link optimized for social networks • Embeddable HTML for use on external webs • Custom mobile and social network application
Publisher integration	<ul style="list-style-type: none"> • None • Export to common print formats • Fully editable crossword • Schedule periodic crossword generation • Well documented API

4.1.2.2 Frontend and Backend

The User interfacing web application features

- Visitors can Sign up to become Users
- Users are handled separately and securely

- User can specify the size and shape of a new Crossword (the Grid)
- User can specify the word categorization preferences of a new Crossword
- User can request one or more Crosswords to be created
- User can see if the solution was found yet or the solution unfeasible
- User can see the Crossword as grid with placed word descriptions once it is created

The user interfacing application seems pretty standard — comparable to a simple blog system or a tiny e-shop. There exist many frameworks in which the Backend of this application might be created — Laravel (PHP), Rails (Ruby), Express(Javascript), Flask(Python), ASP.NET(C#), or Spring(Java). Usually, an abstract layer of ORM is provided with connectors to the common relational databases (MySQL, Postgres, MSSQL, Oracle, etc.). Such frameworks usually come with some module ecosystem and development conventions. Maintainability and extensibility is the goal of all these frameworks so we may do the pick based on developer preferences.

The interesting part would be interfacing with the crossword-generating workers. That will be discussed in the next section.

4.1.2.3 Workers

The process of Crossword Generating may be covered by the Consumer/Producent pattern:

- Backend — the Producent of task that users place
- Crossword Generating worker — the Consumer of tasks that creates the crossword

The workers might be accessing the Backend directly. But there are advantages of adding another node in this chain: a separate queue service. The advantages of such dissociation are:

- Simplifying the code of Backend
- Easing the load on Backend that would be caused by workers polling for new tasks
- Out-of-the-box features such as multiple queues or task priority

We need the specific application choice to support both our Backend application and the Workers programming language.

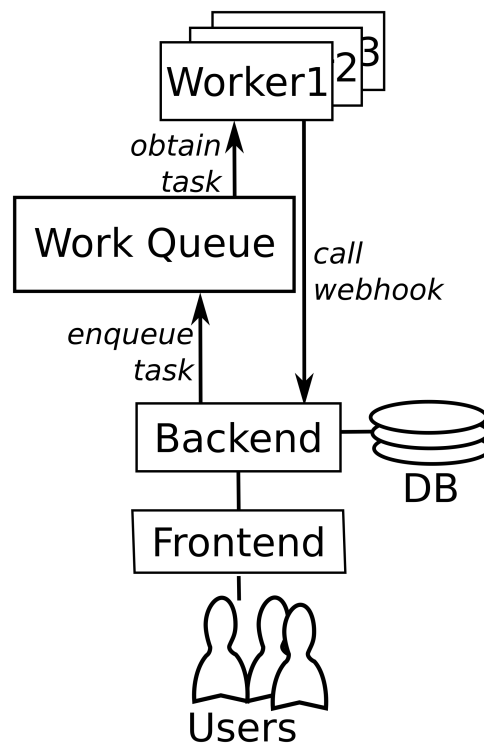


Figure 4.1: User Interfacing App Architecture

4.1.2.4 Architecture


4.1.3 Realisation

4.1.3.1 Frontend

The Frontend application is written in Javascript, using the Vue framework. The Frontend is a Single Page Application, which means it operates in the client's web browser, seamlessly updating the application state based on the interaction with the User and communication with the backend application by asynchronous requests on REST API endpoints. The main objective of this is decoupling of Backend and Frontend — an alternative client may be written (e.g., a native mobile application) and it allows the API to be exposed for the publishers (after it is mature and properly documented). The Publishers can then integrate without using the original Frontend application.

I have selected Vue framework because of its popularity and the reactive concepts, which in my opinion, evolved from the older javascript frontend frameworks (AngularJS, Angular, React, and others) to the state in which they are quite handy and intuitive to use. [58] provides an in-depth comparison.

KŘÍŽOVKY MŘÍŽKY ODHLÁSIT SE







































	Apr 21, 2021 8:39 PM		5x5		
	Apr 21, 2021 8:39 PM		7x7	ART	
	Apr 21, 2021 8:55 PM		7x7	CHE	
	Apr 21, 2021 9:10 PM		7x7	MAT, PHY, TEC	
	Apr 21, 2021 10:49 PM		7x7	ART	
	Apr 22, 2021 8:22 AM		5x5	BIO	
	Apr 22, 2021 10:10 AM		5x5	AGR	
	Apr 22, 2021 2:17 PM		5x5	AGR	
	Apr 24, 2021 1:08 PM		7x7	GEO, ICT, MAT, PHY, TEC	
	Apr 24, 2021 5:13 PM		9x9	AGR, ART, EDU	
	Apr 29, 2021 9:09 AM		7x7	BIO	
	May 4, 2021 10:29 AM		7x7	ART, EDU	

Figure 4.2: Frontend Crossword list

4.1.3.2 Backend

The Backend is written in Javascript using the Express [59] framework. It uses Sequelize [60] for ORM. Backend authenticates and authorizes the user session and provides the API endpoints to fetch and manipulate the data in the manner of CRUD

4.1.3.3 Backend Data Model

- **User** — the application user authenticated by email and password (stored hashed in database)

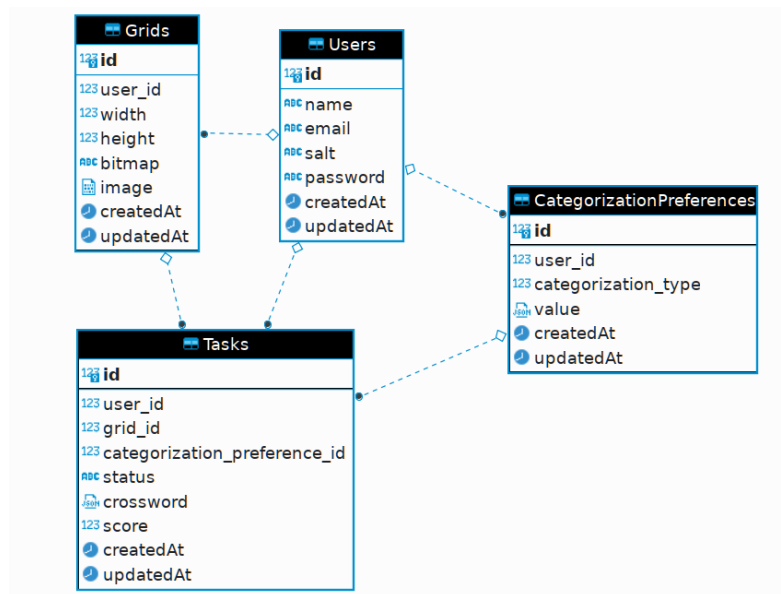


Figure 4.3: Backend entity-relationship diagram

- **Grid** — the user-editable Crossword grid: specifying size of crossword and word limits
- **CategorizationPreference** — user-specified vector of their interests (target words categories)
- **Task** — In the *created* state it is the crossword demand specification. After the Task is fulfilled by the Worker (the Crossword is created), the Task contains the created crossword (list of words and descriptions with their coordinates in the grid)

4.1.3.4 Working Queue

I have chosen the working server Faktory [61]. Other found solutions such as Bull [62] have no Python bindings, so creating a wrapper would be needed. After the User creates a Task through the Backend API, the Task is serialized into JSON and uploaded into Faktory server. There it waits for being fetched by the Worker program.

4.1.4 Screenshots





Figure 4.4: Frontend Grid Designing


KŘÍŽOVKY


MŘÍŽKY


ODHLÁSIT SE



Nová křížovka


Mřížky









ZEMĚDĚLSTVÍ, CHOVATELSTVÍ	<input type="checkbox"/>
ANTROPOLOGIE, ENTOGRAFIE	<input type="checkbox"/>
UMĚNÍ, ARCHITEKTURA	<input type="checkbox"/>
BIOLOGIE	<input type="checkbox"/>
CHEMIE	<input type="checkbox"/>
EKONOMIE, OBCHOD, LOGISTIKA	<input type="checkbox"/>
PEDAGOGIKA	<input type="checkbox"/>
HISTORIE, BIOGRAFIE	<input type="checkbox"/>
VYPOČETNÍ TECHNIKA	<input type="checkbox"/>
KNIHOVNICTVÍ, INFORMATIKA	<input type="checkbox"/>
INTERDISCIPLINÁRNÍ	<input type="checkbox"/>
FILOLOGIE	<input type="checkbox"/>
PRÁVO	<input type="checkbox"/>
MATEMATIKA	<input type="checkbox"/>
LÉKAŘSTVÍ	<input type="checkbox"/>
HUDBA	<input type="checkbox"/>
FILOZOFIE NABOZENSTVÍ	<input type="checkbox"/>
FYZIKA	<input type="checkbox"/>
POLITIKA VOJENSTVÍ	<input type="checkbox"/>
PSYCHOLOGIE	<input type="checkbox"/>
SPORT REKREACE HOBBY	<input type="checkbox"/>

Figure 4.5: Frontend Setting the Categorization

4. WEB APPLICATION

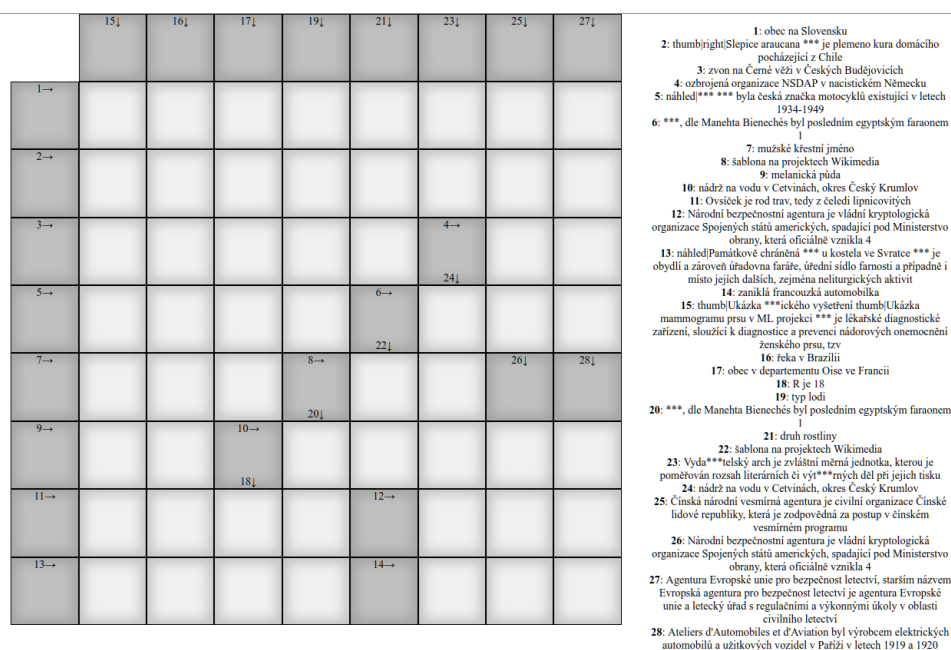


Figure 4.6: Frontend Example of the generated Crossword

- Figure 4.2 shows the list of the Crosswords already generated, waiting to be generated or failed.
- Figure 4.4 is the Grid design screen — user specifies the letter cell / empty cell positions by clicking. The Grid size might be adjusted by the sliders.
- On Figure 4.5 the user specifies the Crossword to be generated: chooses a Grid and inputs categorization preferences by sliders.
- Figure 4.6 shows the generated crossword in the Czech language.

4.2 Application Architecture

4.2.1 Running Modes

Recaping the parts on the project:

Part	Complexity	Interactivity
Word List Generation	High	None
Crossword Generation	Medium	Medium
Frontend and Backend interacting with User	Small	High

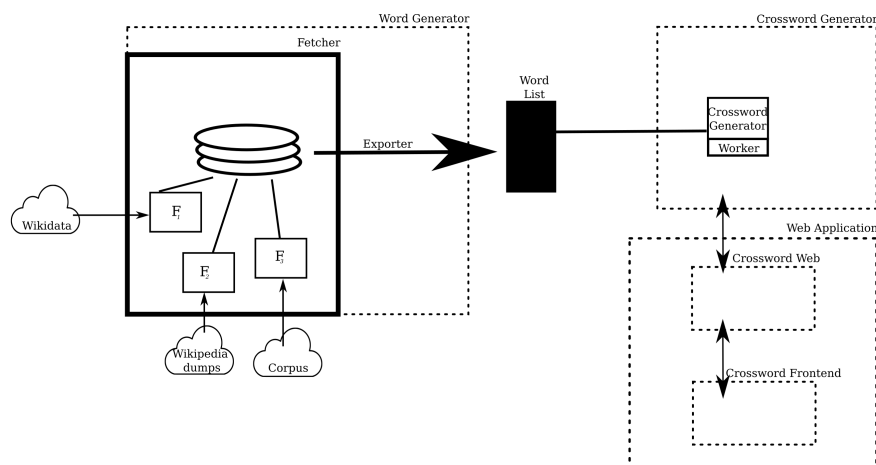


Figure 4.7: Diagram of the application parts

4.2.1.1 Word List Generation

Word list Generation can run at arbitrary times with intervals ranging from days to months. Of course, the occasional words refreshing is beneficial for our app — as more words can be produced and errors can be fixed in the primary sources. But given the maturity of Wikidata and Wikipedia project, it is unlikely that the word improvement per month would exceed 1% of our vocabulary.

The cost for such refreshing is (if you consult Chapter 2) very costly in terms of time, computing, and network resources — it means downloading and parsing the whole Wikipedia dump and re-creating Wikidata database (the process that took days). The current machine learning algorithm also consumes a lot of memory (it was tested with 64GB RAM and failed on less). The upside is there is no need for internet connection stability (apart from downloading tens of gigabytes of data), and the process might be parallelized and even paused. Running this process manually on a desktop computer (with a fragment of the expense compared to any server hosting) is feasible. Probably it can also benefit from services similar to Amazon EC2 Spot [63] that uses the Cloud capacities when the capacities can be spared (off-hours) instead of on-demand.

4.2.1.2 Crossword Generation

This is the true friction point. As the problem is NPC, we are trading off the generated crossword quality against the resources used. The amount of resources needed varies greatly with different task specifications.

Also, there is a time constraint. We should deliver the tasks in a reasonable time — failing to do so will cause the Users leave. The upside here is that the tasks are completely independent, making them perfect adept for parallelization and horizontal scaling - that means we can use multiple worker machines in the case of higher demand and quickly free up the resources when the demand drops. This makes it perfect adept for deploying on cloud services similar to Amazon Lambda [64].

4.2.1.3 Frontend and Backend Interacting with User

The availability is critical for this part. Luckily the resource demands are not so high; it is comparable with usual web applications. The needs are stacking with the number of users and it is good to have it deployed on a scalable infrastructure to be ready for the Slashdot effect.

4.2.2 Word List Data Structure

The Word List data structure contains list of words along with their clues and categorizations. It is produced by the Word List Generator, and it is consumed by the Crossword Generator. The *production* of the Word List happens once a month or less frequently, as the source data does not change rapidly. Therefore it is acceptable to have the process time- and resource-consuming.

The Word List gets *consumed* much more often — every time the crossword is generated. This asymmetry causes my design of the Word List structure to make it as close to the Crossword Generator needs as possible. I have considered three solutions:

1. The Word List as a view to the Word List Generator database. I have dismissed this possibility immediately as it would not account for the asymmetry at all. Every single crossword generated would mean the Crossword Generator would need to load a huge amount of data from some remote database.
2. The Word List as a serialized Crossword Generator internal structure Section 3.3.2. I have dismissed this because it would cause make the development lot less flexible - after updating the Crossword Generator internals, I would have to update the Word List.
3. The Word List as structured files — I considered the simplistic approach of a CSV file, but ended up with a serialized Pandas [35] dataframe called “pickle”.

I wanted to keep the categorization expandable — so the information about the words from different sources might be added easily. Therefore I decided to put the categorization into separate data files joinable through shared `word_id`. Those files are serialized dataframes.

This separation also allows the weights of the words based on the user preference vector to be computed effectively as a matrix multiplication as described in ?? 4.2.2.1.1.

4.2.2.1 Crossword Generation Worker

I have implemented the Worker wrapper to the Crossword Generation program. It gets tasks from a queue as discussed in Section 4.1.2.3.

```

Load the WordList
Load the Categorization
while true do
  | Wait for Task
  | TaskCategorizationScore = Matrix multiply
  |   Categorization · Task.categorizationVector
  | WordListWeighted = join(WordList,
  |   TaskCategorizationScore) over wordid
  | Run WCC(Task.Grid, WordListWeighted )
end

```

Algorithm 3: Crossword Generation Worker

The Word List and Categorization is loaded, and the search structure from 3.3.2 is generated only when the Worker starts. With the Task, a custom User Preference vector comes. The *individual word weights* for this Task must be computed. This can be expressed as matrix multiplication, and experiments show it finishes in 0.1s even for our 230k Word List and 34 categories.

4.2.2.1.1 Example of the Individual Word Weights Computation

A Word List of *length* = 2:

word_id	word_label_text	word_description_text
1	Acoustics	science that deals with the study of sound
2	Earth	third planet from the Sun in the Solar System

A simplified example of the KorpusDB categorization — MUS is the music category, PHY physics, and BIO biology:

<i>word_id</i>	<i>MUS</i>	<i>PHY</i>	<i>BIO</i>
1	0.5	0.45	0.05
2	0.1	0.7	0.2

Task 1 comes with a User Preference vector:

$$\begin{array}{l|l} MUS & 0.8 \\ PHY & 0.1 \\ BIO & 0.1 \end{array}$$

The WordListWeighted gets computed:

$$\begin{pmatrix} 0.5 & 0.45 & 0.05 \\ 0.1 & 0.7 & 0.2 \end{pmatrix} \cdot \begin{pmatrix} 0.8 \\ 0.1 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.45 \\ 0.17 \end{pmatrix}$$

The resulting matrix is joined with the word list again:

word_id	word_label_text	word_description_text	word_weight
1	Acoustics	science that deals with the study of sound	0.45
2	Earth	third planet from the Sun in the Solar System	0.17

It means that Acoustics will be preferred during the crossword filling WCC algorithm run.

4.2.3 Deployment

As the application consists of interconnected services, I have provided a *Dockerfile* for Frontend, Backend, and Crossword Generator, making it possible to build a docker [25] container.

After the containers are build, it is possible to run them using the `docker-compose` tool that uses a configuration written in a YAML file to establish the connections between the services and their dependencies. Excerpt from my *docker-compose.yml*:

```
version: '3.6'
services:
  factory:
    image: contribsys/factory:latest
    environment:
      - FACTORY_PASSWORD
    command: ./factory -b :7419 -w :7420
    ports:
      - "7419:7419"
      - "127.0.0.1:7420:7420"
  postgres:
    image: postgres:13
    environment:
      - POSTGRES_USER: ${PGUSER:-postgres}
```

```
    - POSTGRES_PASSWORD: ${PGPASSWORD:-changeme}
  expose:
    - '5432'
  volumes:
    - ./postgres-data:/var/lib/postgresql/data
crossword-web:
  image: crossword-crossword-web:0.0.1
  depends_on:
    - postgres
    - faktory
  ports:
    - '127.0.0.1:3111:3111' # crossword-web API
  command: bash -c "yarn sequelize db:migrate && yarn start"
crossword-front:
  image: crossword-crossword-front:0.0.1
  ports:
    - "127.0.0.1:3112:80"
```

Specifying the deployment schema with code allows it to be versioned with a version control system such as GIT. Also, it enables the application to be deployed automatically, making the development faster.

4.2.4 Improvements Proposal

Given the wide scope of this work, the software presented is a working prototype, and there is a lot of interesting areas to focus on next.

- Proper Usability Testing
- Focusing on the target audience

Conclusion

As my Master's work, I have created a fully automated Crossword Puzzle Generator. It uses words, clues, and word categorization gathered from public data sources to form a rich word list. Users can access the program through a comfortable web interface that allows them to set their word categorization preferences. User preference influences the crossword generation process. The result is a Crossword adjusted to the User. This is a novel approach to the Crossword Constructing problem.

In this Thesis, I analyzed the problem and divided it into three parts: Word List Generator, Crossword Generator, and User Interface. I implemented those three parts and structured them to interface with each other. The result is a fully functional application deployed in the Czech language on generator-krizovek.cz

In this Thesis, I formulated and analyzed the problems and described my solutions and reasoning behind them. Also, I have proposed several directions to take in this project evolution and possibilities of the following research.

Bibliography

- [1] Vondříčka, P. KorpusDB: Databáze slovních tvarů a lemmat doložených v korpusech ČNK. 2020. Available from: <http://db.korpus.cz/>
- [2] Wikipedia contributors. Crossword. 2021. Available from: <https://en.wikipedia.org/wiki/Crossword>
- [3] SUTHERLAND, Denise. American vs British crosswords. 2013. Available from: <https://alwayspuzzling.blogspot.com/2013/01/american-vs-british-crosswords.html>
- [4] Henry Howarth. Learning how to solve cryptic crosswords. 2018. Available from: http://www.czdcrosswords.co.uk/images/Advice/LH2SCC_white_paper.pdf
- [5] Last, Natan and Steinberg, David. How to Make a Crossword Puzzle. 2018. Available from: <https://www.nytimes.com/2018/05/11/crosswords/how-to-make-crossword-puzzle-grid.html>
- [6] Beekeeper Labs. Available from: <http://beekeeperlabs.com/crossfire/>
- [7] . Available from: <https://www.crossword-compiler.com/>
- [8] KING Keiran. Phil. Available from: <https://github.com/keiranking/Phil>
- [9] Wikipedia contributors. Linked Data. 2021. Available from: <https://en.wikipedia.org/wiki/Crossword>
- [10] CYGANIAK Richard, W. D.; Markus, L. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2014. Available from: <https://www.w3.org/TR/rdf11-concepts/>

- [11] CYGANIAK Richard, W. D.; Markus, L. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2013. Available from: <https://www.w3.org/TR/sparql11-query/>
- [12] Association, D. DBpedia. Available from: <https://wiki.dbpedia.org/>
- [13] The Wikimedia Foundation, I. Wikidata. Available from: https://www.wikidata.org/wiki/Wikidata:Main_Page
- [14] Pellissier Tanon, T.; Weikum, G.; et al. YAGO 4: A Reasonable Knowledge Base. In *The Semantic Web*, Cham: Springer International Publishing, 2020, ISBN 978-3-030-49461-2, pp. 583–596. Available from: <https://suchanek.name/work/publications/eswc-2020-yago.pdf>
- [15] Schema.org. Schema.org. 2021. Available from: <https://schema.org>
- [16] Rigutini, L.; Diligenti, M.; et al. A Fully Automatic Crossword Generator. In *2008 Seventh International Conference on Machine Learning and Applications*, 2008, pp. 362–367, doi:10.1109/ICMLA.2008.104.
- [17] Mynarz, J.; Zeman, V. DB-Quiz: A DBpedia-Backed Knowledge Game. In *Proceedings of the 12th International Conference on Semantic Systems*, SEMANTiCS 2016, New York, NY, USA: Association for Computing Machinery, 2016, ISBN 9781450347525, p. 121–124, doi:10.1145/2993318.2993339. Available from: <https://doi.org/10.1145/2993318.2993339>
- [18] Vega-gorgojo, G. Clover Quiz: A trivia game powered by DBpedia. *Semantic Web*, volume 10, 06 2018, doi:10.3233/SW-180326.
- [19] Mütsch, Ferdinand. Auto-generated trivia questions based on DBpedia data. Available from: <https://github.com/n1try/linkedata-trivia>
- [20] Cloud, T. L. O. D. Wikidata: about the dataset. Available from: <https://lod-cloud.net/dataset/wikidata>
- [21] Cloud, T. L. O. D. DBpedia: about the dataset. Available from: <https://lod-cloud.net/dataset/dbpedia>
- [22] The Wikimedia Foundation, I. Wikidata Query Service/User Manual. Available from: https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual
- [23] Addshore. Your own Wikidata Query Service, with no limits. Available from: <https://addshore.com/2019/10/your-own-wikidata-query-service-with-no-limits/>
- [24] Ltd., C. Ubuntu. Available from: <https://ubuntu.com/download/desktop/thank-you?version=20.10&architecture=amd64>

-
- [25] Docker, I. Docker Engine. Available from: <https://www.docker.com/>
- [26] Wikibase. Wikibase Query Service (Blazegraph) docker image. Available from: <https://hub.docker.com/r/wikibase/wdqs>
- [27] Wikibase. Wikibase Query Service docker image source code. Available from: <https://github.com/wmde/wikibase-docker/tree/master/wdqs>
- [28] The Wikimedia Foundation, I. Wikidata dumps. Available from: <https://dumps.wikimedia.org/wikidatawiki/entities/>
- [29] Consortium, S. Sqlite3. Available from: <https://www.sqlite.org/index.html>
- [30] The Wikimedia Foundation, I. Wikimedia REST API. Available from: https://wikimedia.org/api/rest_v1/
- [31] Cvrček, V. Genre (tematická oblast) – seznam hodnot. 2015. Available from: <https://wiki.korpus.cz/doku.php/seznamy:genre>
- [32] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.
- [33] scikit-learn developers. Multiclass and multioutput algorithms. 2021. Available from: <https://scikit-learn.org/stable/modules/multiclass.html>
- [34] Hagberg, A. A.; Schult, D. A.; et al. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, edited by G. Varoquaux; T. Vaught; J. Millman, Pasadena, CA USA, 2008, pp. 11 – 15.
- [35] Reback, J.; McKinney, W.; et al. pandas-dev/pandas: Pandas 1.2.4. Apr 2021, doi:10.5281/zenodo.4681666.
- [36] The Wikimedia Foundation, I. Wiktionary. 2021. Available from: <https://www.wiktionary.org/>
- [37] Engel, J.; Holzer, M.; et al. On Computer Integrated Rationalized Crossword Puzzle Manufacturing. 06 2012, pp. 131–141, doi:10.1007/978-3-642-30347-0_15.
- [38] GeeksforGeeks. Number of possible Functions. Available from: <https://www.geeksforgeeks.org/number-of-possible-functions/>

- [39] Fišer, P. Combinatorial problems and algorithms, complexity. Lecture, 10 2020. Available from: <https://moodle-vyuka.cvut.cz/mod/page/view.php?id=89314>
- [40] Peterson, O.; Wehar, M. Automatic Crossword Puzzle Construction. Presented at NCUR 2021, 2021, conference name. Available from: <https://www.crosswordconstruction.com/>
- [41] Mazlack, L. J. Computer construction of crossword puzzles using precedence relationships. *Artificial Intelligence*, volume 7, no. 1, 1976: pp. 1–19, ISSN 0004-3702, doi:[https://doi.org/10.1016/0004-3702\(76\)90019-9](https://doi.org/10.1016/0004-3702(76)90019-9). Available from: <https://www.sciencedirect.com/science/article/pii/0004370276900199>
- [42] Ginsberg, M.; Frank, M.; et al. Search Lessons Learned from Crossword Puzzles. In *AAAI*, 1990.
- [43] Berghel, H. Crossword Compilation with Horn Clauses. *Comput. J.*, volume 30, no. 2, Apr. 1987: p. 183–188, ISSN 0010-4620, doi:[10.1093/comjnl/30.2.183](https://doi.org/10.1093/comjnl/30.2.183). Available from: <https://doi.org/10.1093/comjnl/30.2.183>
- [44] Beacham, A.; Chen, X.; et al. Constraint Programming Lessons Learned from Crossword Puzzles. In *Canadian Conference on AI*, 2001.
- [45] Bonomo, D.; Lauf, A. P.; et al. A crossword puzzle generator using genetic algorithms with Wisdom of Artificial Crowds. In *2015 Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES)*, 2015, pp. 44–49, doi:[10.1109/CGames.2015.7272960](https://doi.org/10.1109/CGames.2015.7272960).
- [46] Audemard, G.; Simon, L. On the Glucose SAT Solver. *International Journal on Artificial Intelligence Tools*, volume 27, no. 01, 2018: p. 1840001, doi:[10.1142/S0218213018400018](https://doi.org/10.1142/S0218213018400018). Available from: <https://doi.org/10.1142/S0218213018400018>
- [47] W3C. Web Assembly. 2021. Available from: <https://www.w3.org/wasm/>
- [48] Harris, G. Generation of solution sets for unconstrained crossword puzzles. In *Proceedings of the 1990 Symposium on Applied Computing*, United States: IEEE Computer Society, 1990, pp. 214–219, doi:[10.1109/SOAC.1990.82171](https://doi.org/10.1109/SOAC.1990.82171).
- [49] Kuzma, T. Heuristic Crossword Puzzle Generation. 2009. Available from: http://eprints.fri.uni-lj.si/799/1/KuzmaT_UN.pdf
- [50] Agarwal, C.; Joshi, R. Automation Strategies for Unconstrained Crossword Puzzle Generation. 07 2020.

-
- [51] Littman, M. L.; Keim, G. A.; et al. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, volume 134, no. 1, 2002: pp. 23–55, ISSN 0004-3702, doi:[https://doi.org/10.1016/S0004-3702\(01\)00114-X](https://doi.org/10.1016/S0004-3702(01)00114-X). Available from: <https://www.sciencedirect.com/science/article/pii/S000437020100114X>
- [52] Ginsberg, M. Dr.Fill: Crosswords and an Implemented Solver for Singly Weighted CSPs. *J. Artif. Intell. Res. (JAIR)*, volume 42, 09 2011: pp. 851–886, doi:10.1613/jair.3437.
- [53] Fišer, P. NPC and NPH problems, Karp reduction, Turing reduction. Lecture, 10 2020. Available from: <https://moodle-vyuka.cvut.cz/mod/page/view.php?id=89312>
- [54] Nikos, M. Algorithm to generate a crossword. 2014. Available from: <https://stackoverflow.com/a/23435654>
- [55] Vojtěchovská, Martina. Čtenost časopisů zůstává stabilní, některé skupiny si polepšily. 2020. Available from: <https://www.mediaguru.cz/clanky/2020/11/ctenost-casopisu-zustava-stabilni-nektere-skupiny-si-polepsily/>
- [56] Pošustová, Kateřina. Senioři a volný čas. 2008. Available from: <https://dspace.cuni.cz/bitstream/handle/20.500.11956/100050/120297475.pdf>
- [57] CHÝLKOVÁ, LENKA. Životní styl seniorů. 2011. Available from: https://theses.cz/id/wteyds/BP_2011_Chylkova.pdf
- [58] Vue.js. Comparison with Other Frameworks. Available from: <https://vuejs.org/v2/guide/comparison.html>
- [59] OpenJS Foundation. Express. Available from: <https://expressjs.com/>
- [60] Sequelize. Sequelize. Available from: <https://sequelize.org/>
- [61] Contributed Systems. Factory. Available from: <https://github.com/contribsys/factory>
- [62] OptimalBits. Bull. Available from: <https://github.com/OptimalBits/bull>
- [63] Amazon. Amazon EC2 Spot Instances Pricing. Available from: <https://aws.amazon.com/ec2/spot/pricing/>
- [64] Amazon. AWS Lambda. 2021. Available from: <https://aws.amazon.com/lambda/>

Glossary

API Application Programming Interface . 13, 35, 36

CC-BY-SA Creative Commons license with Share-Alike and Attribution terms . 8

CC0 Creative Commons CC0 license - license that waives the copyright so the work is placed in the public domain as completely as possible . 8

CRUD create, read, update, and delete (CRUD) is a software architectural style regarding the four basic operations of persistent storage. (wikipedia) . 36

CSP Constraint Satisfaction Problem, . 25

CSV Comma Separated Values - a delimited text file that uses a comma to separate values. (wikipedia) . 42

freemium Offering basic services for free while charging a premium for advanced or special features. (wiktionary) . 32

Genetic Algorithm Genetic Algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. (wikipedia). 29

GIT GIT is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development (wikipedia). 45

homograph A word that is spelled the same as another word, usually having a different etymology. (wiktionary) . 13

- inflectional languages** Languages in which words are derived from a base form(lemma) by prefixes and suffixes. . 19
- JSON** JavaScript Object Notation; a data format used to represent structured data (wiktionary) . 37
- NLP** Nature Language Processing . 9, 12
- NPC** NP-complete problem, any of a class of computational problems for which no efficient solution algorithm has been found. (britannica). 2, 24-26
- NPH** NP-hard problem - the class of problems at least as hard as the hardest problems in NP. (Fišer) . 26
- ORM** Object-Relational Mapping; a higher layer way to work with relational database. . 34, 36
- RDF** Resource Description Framework, a W3C specification . 8
- REST** Representational state transfer; a software architectural style which uses a subset of HTTP and a predefined set of stateless operations. . 13, 35
- Simulated Annealing** Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. (wikipedia). 29
- Slashdot effect** The Slashdot effect, also known as slashdotting, occurs when a popular website links to a smaller website, causing a massive increase in traffic (wikipedia) . 42
- SPARQL** SPARQL Protocol and RDF Query Language . 8, 17, 20
- SSD** Solid State Drive . 10
- TBW** Terabytes Written (or Total Bytes Written) - endurance metrics in Solid State Drives . 10
- URI** Uniform Resource Identifier . 8
- userbase** The established group of users for a particular computer program, technology, etc. (wiktionary) . 32
- Web Assembly** A binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications. (w3.org/wasm/). 25

YAML "YAML Ain't Markup Language", a human-readable data-serialization language. (wikipedia) . 44

Contents of enclosed data medium

crossword-gen.....	Crossword Generator in Python
crossword-compose.....	docker-compose files
plot_scripts.....	scripts to create plots
thesis.....	all source files of this thesis
└ DP_Benda_Adam_2021.pdf	thesis in PDF format
└ DP_Benda_Adam_2021.tex	thesis source
crossword-front.....	Frontend in Javascript with Vue
crossword-web.....	Backend in Javascript with Express
wordgen	Word List Generator in Python