



Norwegian University of  
Science and Technology

# Extracting and Exploring Examples from semi-structured Text

**Espen Hellerud**

Master of Science in Informatics

Submission date: May 2016

Supervisor: Pinar Öztürk, IDI

Norwegian University of Science and Technology  
Department of Computer and Information Science



---

# Abstract

Active Learning has been a highly promoted form of learning the last decades. One of the exercises Active Learning makes use of is task and problem solving. Utilizing examples solving similar problems, can be very helpful when performing these exercises. Therefore this project will create a searchable database of examples, to help users finding relevant examples which can aid them in exercises involving solving tasks and problems. We will use Wikipedia as a source of examples. The system will extract examples found in an XML dump of Wikipedia, transform them into example objects, which will be inserted into a database. A user interface will be created for searching the database and displaying the examples for the user. A software pipeline will be used as the system's main architectural pattern. The independent processes in a software pipeline is very beneficial in the time consuming task of parsing the XML dump of the entire Wikipedia.

The work of the thesis resulted in a system that can parse the XML dump of Wikipedia and create a database of examples. A search interface lets the user enter keywords and displays the returned examples. The system is able to find relevant examples to a satisfactory degree. Since the final implemented system acts as a minimum viable product, a number of propositions for future improvements are also included in the thesis.

---

---

# Preface

This document and its associated source-code is the end product of the Masters Thesis of Espen Hellerud at the Norwegian University of Science and Technology (NTNU).

# Acknowledgements

The thesis work have been conducted with guidance of Pinar Øzturk and Kerstin Bach at the Department of Computer and Information Science(IDI). They have given me valuable guidance, assistance and feedback throughout the whole process of completing my Masters Thesis.

I would also like to thank John Arne Øye for reading through my thesis, giving me feedback on spelling errors and sentence improvements.

---

---

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>Table of Contents</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Research Goals . . . . .	3
1.3 Structure of thesis . . . . .	4

---

<b>2</b>	<b>Literature Review and Background</b>	<b>7</b>
2.1	Related work . . . . .	7
2.1.1	Semi-structured text . . . . .	7
2.1.2	Wikimedia . . . . .	8
2.1.3	Text data mining . . . . .	9
2.1.4	SMILA . . . . .	11
2.2	Methods . . . . .	12
2.2.1	Pipeline . . . . .	12
2.2.2	TF-IDF . . . . .	13
<b>3</b>	<b>Conceptual design</b>	<b>15</b>
3.1	Overall Approach . . . . .	15
3.2	Pipeline . . . . .	16
3.2.1	Creating the pipeline . . . . .	16
3.2.2	Extracting data from XML-dump . . . . .	17
3.2.3	Building an index of examples from SQL . . . . .	18
3.3	Search examples . . . . .	18
3.3.1	General idea . . . . .	18
3.3.2	Querying by keywords . . . . .	19
3.3.3	Querying by examples . . . . .	20
3.4	Analysis of examples . . . . .	20
3.4.1	An example . . . . .	20



---

3.4.2	Comparing examples . . . . .	21
3.4.3	Structure of a good example . . . . .	21
<b>4</b>	<b>Implementation</b>	<b>25</b>
4.1	Pipeline . . . . .	25
4.1.1	Overview . . . . .	25
4.1.2	Wikidump Parser . . . . .	27
4.1.3	SQL Database . . . . .	30
4.1.4	Example Indexer and Whitelist . . . . .	31
4.1.5	Index . . . . .	35
4.2	Example Search . . . . .	36
4.3	Tools . . . . .	38
4.3.1	Elasticsearch . . . . .	38
4.3.2	NPM and Node.js . . . . .	39
<b>5</b>	<b>Experiments, Results and Discussion</b>	<b>41</b>
5.1	Resulting Pipeline . . . . .	42
5.2	Describing a good example . . . . .	43
5.3	Collection of examples . . . . .	44
5.4	Search interface . . . . .	45
5.5	Querying by keywords . . . . .	47
5.5.1	Experiment I: Search examples . . . . .	47

---

5.5.2	Experiment II: Whitelist . . . . .	51
5.5.3	Experiment III: Evaluation by F-score . . . . .	57
5.5.4	Discussion of results . . . . .	58
5.6	Querying by examples . . . . .	64
5.6.1	Experiment IV: Finding related examples . . . . .	64
5.6.2	Discussion of results . . . . .	66
<b>6</b>	<b>Conclusion and Future Work</b>	<b>69</b>
6.1	Conclusion . . . . .	69
6.2	Future Work . . . . .	71
	<b>Bibliography</b>	<b>73</b>
	<b>Appendices</b>	<b>75</b>
<b>A</b>	<b>Database statistics</b>	<b>77</b>
<b>B</b>	<b>Example samples</b>	<b>79</b>
<b>C</b>	<b>Example white lists</b>	<b>91</b>
<b>D</b>	<b>Charts from Experiment II</b>	<b>97</b>

# List of Tables

3.1	Properties of an example and a description of them. . . . .	22
4.1	Statistics for examples and categories included when applying different whitelists for filtering of the example collection. . . . .	33
5.1	Mapping of which sections discusses the different research goals. . . . .	42
5.2	Results from processing the XML dump through the pipeline	43
5.3	Statistics from the index after original implementation. . .	45
5.4	The precision of the queries evaluated by a set of keywords and the top ten and top five results . . . . .	48
5.5	The precision of the queries when all examples are included.	52
5.6	The precision of the queries when whitelist <i>Edu</i> is used to filter the collection beforehand . . . . .	53
5.7	The precision of the queries when whitelist <i>Top200Edu</i> is used to filter the collection beforehand . . . . .	54

---

5.8	The precision of the queries when whitelist <i>MathTech</i> is used to filter the collection beforehand . . . . .	55
5.9	The precision of the queries when whitelist <i>MathTechWiki</i> is used to filter the collection beforehand . . . . .	56
5.10	Results of how well the system retrieves example from a manually inspected set. . . . .	58
5.11	Statistics when querying by examples with <i>java</i> as keyword	65
5.12	Statistics when querying by examples with <i>nash equilibrium</i> as keyword . . . . .	66
A.1	Statistics over most popular categories . . . . .	78
C.1	Removes categories not related to educational topics within science. For instance topics regarding history is removed. Includes only the top 200 most popular categories in the list.	96

# List of Figures

1.1	A typical representation of the learning pyramid, showing passive and active learning techniques and how much knowledge retained. . . . .	2
2.1	An overall overview of the SMILA architecture . . . . .	12
3.1	A conceptual overview of the pipeline used to extract and index examples . . . . .	16
3.2	A simple overview of the user's interaction with the examples	19
4.1	Architectural overview of the pipeline . . . . .	26
4.2	Architectural overview of Wikidump Parser . . . . .	27
4.3	The XML structure used in the XML dump of Wikipedias database . . . . .	28
4.4	A short outline from an article's wiki markup, with important aspects highlighted and marked. . . . .	29
4.5	A simple overview of the database tables and their attributes	31

---

4.6	A venn diagram showing how whitelists act as subsets of all categories . . . . .	32
4.7	Mapping used to define an example in Elasticsearch . . . . .	34
4.8	A sequence diagram showing the interactions between the actors when searching for examples . . . . .	36
5.1	A screenshot of the search interface before a search is performed. . . . .	45
5.2	A screenshot of the search interface after a search is performed, showing the returned results. . . . .	46
5.3	A screenshot of the search interface when a specific example has been chosen. . . . .	47
5.4	Average precision for the three different groups . . . . .	49
5.5	Average precision for all whitelists . . . . .	58
5.6	Precision score when the keyword <i>Java</i> is used as a search phrase, with the different whitelists applied. . . . .	59
5.7	Precision score when the keyword <i>Derivation</i> is used as a search phrase, with the different whitelists applied. . . . .	60
5.8	Precision score when the keyword <i>Zero Sum</i> is used as a search phrase, with the different whitelists applied. . . . .	62
D.1	Precision score when the keyword <i>Logic</i> is used as a search phrase, with the different white lists applied. . . . .	97
D.2	Precision score when the keyword <i>Programming</i> is used as a search phrase, with the different white lists applied. . . . .	98
D.3	Precision score when the keyword <i>Heuristic</i> is used as a search phrase, with the different white lists applied. . . . .	98

---

---

D.4	Precision score when the keyword <i>Algebra</i> is used as a search phrase, with the different white lists applied. . . . .	99
D.5	Precision score when the keyword <i>Game Theory</i> is used as a search phrase, with the different white lists applied. . . . .	99
D.6	Precision score when the keyword <i>Fuzzy Logic</i> is used as a search phrase, with the different white lists applied. . . . .	100
D.7	Precision score when the keyword <i>Bayes Network</i> is used as a search phrase, with the different white lists applied. . . . .	100
D.8	Precision score when the keyword <i>Chain Rule</i> is used as a search phrase, with the different white lists applied. . . . .	101
D.9	Precision score when the keyword <i>Prisoners Dilemma</i> is used as a search phrase, with the different white lists applied.	101
D.10	Precision score when the keyword <i>Nash Equilibrium</i> is used as a search phrase, with the different white lists applied. . . . .	102
D.11	Precision score when the keyword <i>Cartesian Product</i> is used as a search phrase, with the different white lists applied. . . . .	102
D.12	Precision score when the keyword <i>Parrondos Paradox</i> is used as a search phrase, with the different white lists applied.	103

---

---

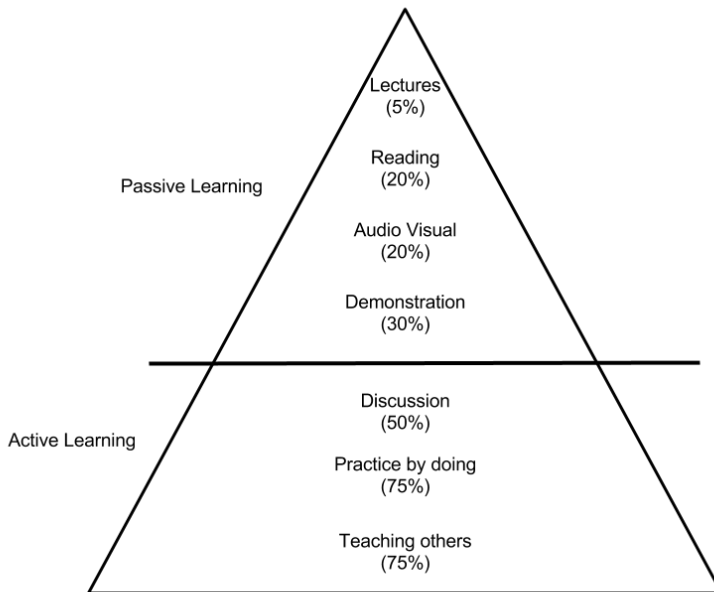


# Introduction

## 1.1 Motivation

Learning has always been, and will most likely always be one of the central pillars of human success. Humans are very passionate about learning, and therefore continually improve on techniques for acquiring knowledge. The traditional way of learning has been *Passive Learning*, but *Active Learning* has been more prevailing, especially for schools and universities [1]. Although the classical passive learning technique, lectures, are still heavily used, discussions, problem solving, tasks and student presentations have become a big part of a normal school day. J. P. Lalley and R. H. Miller discusses the learning pyramid produced by Edgar Dale [2]. Figure 1.1 depicts the learning pyramid displaying different learning techniques, and how much knowledge you retain after using them. Edgar Dale's research has been disputed since the original data, regarding retaining knowledge, was not recorded. Despite the retention values may be inaccurate, it gives an overview of different learning techniques.

Almost at the bottom of the pyramid, the technique *Practice by doing* can be found. Practice by doing is a very popular technique, often in the form of assignments based on solving tasks or problems. From personal experience in solving tasks, making use of examples that solves similar



**Figure 1.1:** A typical representation of the learning pyramid, showing passive and active learning techniques and how much knowledge retained.

tasks can be of great help. Finding helpful examples among the massive information accessible through the web is sometimes challenging though.

Educational Technology could be applied to improve the process of finding helpful examples. The article *Facilitating Learning* [3] uses AECT's<sup>1</sup> definition of Educational Technology, "Educational technology is the study and ethical practice of facilitating learning and improving performance by creating, using, and managing appropriate technological processes and resources." The article highlights the use of the term *facilitation* in the definition. They argue that Educational technology's primary purpose is to **help** people learn, not control or manage it, which older definitions leaned towards. This project will facilitate the users' learning by creating a technological process that will gather and manage a collection of examples. The

---

<sup>1</sup>The Association for Educational Communications and Technology

---

collection can be used as a tool for learning new subjects or solving tasks.

To accomplish this, the thesis will look closer into the examples themselves. We want to discover what separates a good example from a bad one, and if there is a particular structure reflecting the quality of the example. With this knowledge, we will explore the possibility of creating a database consisting purely of examples. Publicly available examples found in Wikipedia will be transformed into structured example objects, and then inserted into a database that users can access through a search interface.

## **1.2 Research Goals**

The overall goal of our work is to build a platform where users can find examples, which will aid them in learning. To help aim the work towards the thesis' overall goal, four research questions have been defined. Completion of these goals will ensure good quality of the implemented system.

### **1 - Set up a pipeline**

The first goal of this study will be to setup a software pipeline in the most beneficial way. The pipeline should extract examples from Wikipedia. The output of the pipeline should be a searchable database of the examples.

### **2 - Define a good example by using their structure and content**

To be able to create a high quality database consisting of examples, a deeper understanding of the examples themselves is needed. Therefore the second research goal in this study will focus on the nature of the examples. The study will look into how the examples are structured, what properties are preferable in certain circumstances and what the content of an example should be in regards to its domain.

---

### **3 - Create and populate a database of examples**

- To be able to populate the database, we first have to create it. Which database management system to use is essential, since it will directly affect the process of inserting and retrieving examples. The modeling of the database in terms of possible entities, relations and properties will also have to be considered.
- When the database is created the next step will be to populate it with examples. Based on the models decided for the database, queries will be created that insert all relevant examples.

### **4 - Implement a user interface for searching examples**

- Showing how the created database of examples can be used to be queried and present the relevant example information to the user. This includes different search queries facilitating different objectives that can be served by the example database.
- An interface is needed to make the user able to execute the queries. The interface also has to present the result of the queries to the user.

## **1.3 Structure of thesis**

This thesis will describe the process of creating a searchable database consisting of examples. In order to accomplish the end results, four research goals have been formulated. The project starts with a large dump of source data from Wikipedia, for this data to be fully used, it first has to be filtered and structured by using a pipeline (Research Goal 1). To better understand examples, and thus improving search results, we will also look into what defines a good example (Research Goal 2). The end result of the pipeline is a database populated with examples. The database has to store the examples in a way that makes them easy to retrieve (Research Goal 3). Finally, a user interface is needed to search for examples from the database. Queries with keywords as user input will be used to retrieve relevant examples.

---

Chapter 2 will explore the related work in field of information retrieval and extraction. In particular, it will look for data extraction from wiki sites, and data retrieval of semi-structured text. In addition, the chapter will also explore an alternative approach, SMILA, and look at some techniques used in this project.

Chapter 3 will elaborate on the conceptual design of the project. The main reasons for using a pipeline and how the pipeline refines the source data, will be explained. The chapter will also explain how the project intend to search for examples. Finally a thorough analysis of examples, reflecting Research Goal 2, will be presented.

Chapter 4 explains the implementation of the system. The chapter will start by stepping through the pipeline, explaining each subprocess used to refine the data. Meanwhile it will elaborate on the role of different files in the project, what libraries used to help and storage systems used. The search interface created for fetching the examples will be described, including its interaction towards the rest of the system. Finally a more detailed explanation will be given for the main tools used by the system.

Chapter 5 will discusses whether and how the Research Goals are accomplished. It will try to answer for each one of them in order, either by elaborating in how the goal is achieved already by the system or in the thesis, or by conducting experiments to verify that the system produces the intended results.

Chapter 6 will conclude our thesis. With the results of the experiments in Chapter 5 and the following discussion, the chapter will try to answer whether the research goals have been reached or not. As an ending for the chapter and the thesis, the future of the system will be discussed.

---

# Literature Review and Background

This chapter consists of two parts, Related work and Methods. Section 2.1 attempts to place our work among already existing work in the field of information retrieval and information extraction. The subsections will explore work done by other researchers, and discuss whether aspects of their work can be applied to our work. Section 2.2 will briefly explain two key methods used in our project, and why they are used.

## **2.1 Related work**

### **2.1.1 Semi-structured text**

When extracting information from a textual document, it is important to know how structured the text is. There are two extremes that the document most likely falls between, *structured data* and *free text* [4] also called unstructured data. Structured data is using data models for organizing and standardizing data elements, also including their relations. The other extreme, free text, is unstructured as a newspaper article. Wikipedia's articles exists somewhere between these two extremes. We define a Wikipedia article as semi-structured text. We define it as semi-structured because the ar-

---

ticles from Wikipedia are created using a comprehensive markup language. The articles are also stored in the form of this markup language. Because of that we can extract the information from the article's markup and not from the page presented to the readers at the Wikipedia page.

The fact that the articles are semi-structured is an advantage when extracting information. In Wikipedia's case all sections has a header, which lets us not only know the subject of the sections content, but it also shows its position in the hierarchy of all the article's sections. These added tags to the text makes it self-describing, making it possible for us to find the semantics of the text easier. It also defines the hierarchy of different parts in the text, like sections, records or fields. In free text subtle inferences are required based on grammar to create domain objects, which in turn will describe the semantics of the text.

### **2.1.2 Wikimedia**

Wikimedia [5] is an organization that supports and manages many different knowledge projects. Their goal is to make free knowledge accessible anywhere and on any platform. Their income comes primarily from donations. They do not make us of ads, because they believe it could jeopardize their reliability as a neutral source of information.

Wikimedia is mostly known for Wikipedia [6]. Wikipedia is the largest collection of free, collaborative knowledge that exists. Wikipedia can be found in over 290 languages and across those, contains more than 35 million articles. Our project will make use of the English Wikipedia, which contains more than 5 million articles. Wikipedia uses a software called MediaWiki [7]. It is a server side software used for hosting wiki sites. Many of the wiki sites under the Wikimedia umbrella makes use of MediaWiki. A part of the MediaWiki software is the markup language used for writing articles. This markup, written by contributors, will then later be translated into HTML when displayed for users in their web browser. This markup is used to create elements such as tables, equations, lists and links. Wikimedia regularly backs up Wikipedia and makes it publicly available as a large XML file. The XML file provides access to the raw data of articles,



---

which is the article's metadata and the content of the article in MediaWiki markup. The metadata gives access to useful information such as id, title and revision. The revision data includes author and timestamp. The meta-data also let us determine if the article contains content itself or if it just redirects to another article.

Public access to this information gives a lot of opportunities. Alberto Montero-Asenjo and Carlos A. Iglesias used a XML dump from the Spanish Wikipedia for language research [8]. They created a piece of software that processes the raw source data from the XML dump. It starts by converting the Wiki markup into plain text and then use further operations on the plain text to make the end result of useful data. While they use only plain text as data for the language research, we need to extract information by treating the source data as semi-structured text. Information can be extracted as semi-structured text by looking at the markup. Section headers, references, code tags and categories, are examples of extra information the markup makes available. To gain the extra information made available by the markup, the syntax and the semantics behind it has to be interpreted, instead of filtering it away. How to interpret the semantics that the markup reveals, has been touched on and discussed in section 2.1.3 and a couple of articles [9] [10]. There is though a surprisingly lack of work about the interpretation of the markups syntax itself. Because of this, we have had to build this from the ground up, with help from a Wikipedia page created for assisting people writing the articles<sup>1</sup>. You can read more about our work on this issue in sections 3 and 4.

### **2.1.3 Text data mining**

This projects aims at extracting example sections from articles on Wikipedia and then finding relations among them. Therefore this project is related to two research fields. One of the research fields is information retrieval (IR), which is mainly about helping users finding documents of their needs [11]. The other research field is text data mining (TDM), where the goal is to discover useful information from textual data. Hearst discusses methods that can help us find and extract example sections from Wikipedia in her

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Help:Wiki\\_markup](https://en.wikipedia.org/wiki/Help:Wiki_markup)

---

work, *Untangling text data mining* [12]. She explains how to find patterns across data sets and finding relevant information among a collection of mostly irrelevant data. She achieves this by focusing on a mixture between computationally-driven and user-guided analysis, rather than a fully artificial intelligent text analysis.

To be able to discover relevant data from the Wikipedia articles, Wiki-media's markup language has to be interpreted. To interpret the articles, we will focus on the structure that we can derive from its markup. The articles' structure mainly consists of section headers followed by the textual content of the section. There is also other information that can be extracted. To find relations between articles we would also need to look at the semantics from this information, such as lists, categories and references, amongst others. YAWN<sup>2</sup>, is a project that created an XML version of Wikipedia with focus on semantic information [13]. The XML corpus produced by YAWN is general for the whole Wikipedia and the entire article collection. Our project focuses only on the articles that are relevant for educational purpose and comprising examples to explain its content. Although YAWN does more than what is needed for our purpose, we can use key concepts and techniques for exploiting the Wiki markup to classify examples, in our work. The most interesting for this project is how it finds semantic annotations for Wikipedia pages. In order to do so, it uses a combination of exploiting information about categories assigned to articles and deriving information from the structure of an article. The categories are added by the author to place the article among related articles in specific domains.

Making heavy use of categories is a straight forward method to discover relations between examples. However making use of other information in the articles, would further increase the accuracy of the relations discovered. Internal links between articles is another way we can mine data about relations between articles. Links also force us to consider the difference between a link going into an article, and a link going from one. Evgeniy Gabrilovich and Shaul Markovitch used Wikipedia as a knowledge repository in an effort to enhance text categorization [9]. They made use of links and their sometimes differing *anchor texts*<sup>3</sup> to improve their text categorization. A concept they took advantage of were that different anchor

---

<sup>2</sup>Yet Another Wikipedia Annotation project

<sup>3</sup>Display text of the link

---

texts for the same link can indicate that the links are used in different contexts in the articles. They also looked at number of incoming links to an article. In contrast, our project aims to look deeper into the relationships between articles. We aim to utilize the information about how one article links to another and if that makes them related. David Milne and Ian H. Witten look more into this connection between articles in their approach, Wikipedia Link-based Measure (WLM) [10]. Here they used mainly two methods to measure the relatedness between two articles, based on links. The first one is an approach similar to TF-IDF<sup>4</sup>, a information retrieval algorithm explained in section 2.2.2, where they count links instead of terms. They then create vectors according to the vector space model and then find the similarity between two articles based on the angle of their vectors. The second method they use are modeled after the Normalized Google Distance [14]. Here they simply assume that two articles' containing the same link indicates relatedness. On the other hand, if one article contain a specific link which the other do not, it indicates they are not related.

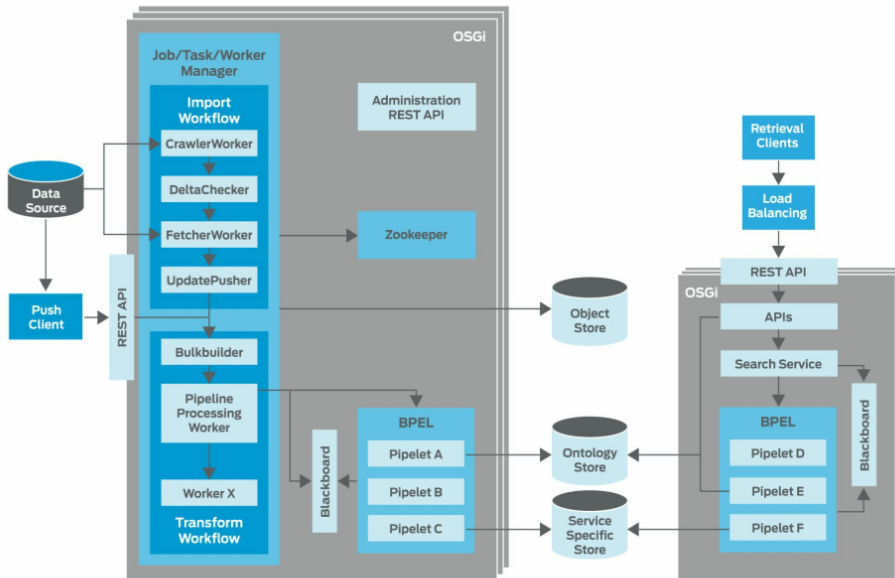
#### 2.1.4 SMILA

In the early stages of our project, an existing tool called SMILA [15] was explored. SMILA is a system with its first release in 2010, it is used to search and access unstructured information. SMILA crawls the web to extract information and then indexes and stores that information. It has a REST API to control the system and for searching the index. The SMILA architecture is also based on the pipeline architecture containing the following processes; jobs, crawling, storage, indexing and querying. Since 2010, 6 new versions has been released adding more features to SMILA. With SMILA being very complex, it gains asynchronicity as its biggest benefit from the pipeline architecture. The SMILA pipeline also allows custom made pipelets to be inserted into the pipeline. A pipelet is a subprocess inside a pipeline. By creating pipelets, the behaviour of SMILA could be tailored into extracting relevant information from Wikipedia.

The releases for SMILA has been dwindling the last three years with only 1 release in 2015. If you add that to the fact that all the different fea-

---

<sup>4</sup>term frequency inverse document frequency



**Figure 2.1:** An overall overview of the SMILA architecture

tures of SMILA makes it very complex, the usability and stability suffers. This was experienced during testing of the program during this project. Nevertheless the utility that the SMILA system offers could be taken advantage of in this project. Either by utilizing SMILA itself, or look at how SMILA retrieves data from the web, processes it and produces a data set as a result.

## 2.2 Methods

### 2.2.1 Pipeline

A software pipeline is a chain of processes where the output of one process is fed as input into another. If these processes are arranged correctly, the

---

result is a pipeline. A software pipeline is actually a design pattern, where it is better known as *pipes and filters* [16]. The two big advantages of a software pipeline is modularity and parallelity [17]. The reason for parallelity is that the data can be spread across several processes. This means that the data can be processed in different instances of the program. Another case is more like a conveyor belt, where data is propagated through the pipeline. So while some of the data are being inputted into the first stages, another part of the data is finalized at the end. The other advantage, modularity, simply means that it is easy to replace a part of the pipeline. The reason for this is that the subprocesses are *loosely coupled*, so changing one part of the system will not affect the rest in any way.

The first mentioned advantage, parallelity, is not used to a significant extent in this project. This is mostly because performance has not been an important quality attribute. Instead the focus has been more on functionality. For this reason the modularity part has been very valuable. It has allowed different parts of the system to be easily replaced or altered in conjunction with the changes of the requirements.

### 2.2.2 TF-IDF

TF-IDF is a statistical measure which evaluates how important a word is to a document based on the document itself and the collection it is part of. Based on this it is possible to decide how likely it is for the document to be relevant. TF-IDF is used as the standard similarity measure in Elastic-Search, see section 4.3.1.

TF-IDF can be divided into two parts, *term frequency* and *inverse document frequency*. *Term frequency* is how often a term appears in a document. The more instances the document has of the word, the higher is the chance of the document being relevant. *Inverse document frequency* looks at how often a term appears in the whole collection of documents. The more often a term appears, the less relevant is the term. This means the common terms will have less weight than rare ones, when calculating the likelihood of the documents relevance.

To calculate the similarity the formula needs a term  $t$ , a document  $d$  and

---

a document collection  $D$ . TF-IDF is then calculated as

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

Both the term frequency and the inverse document frequency can use various ways to determine the exact values, following is two simple variants:

$$tf(t, d) = f_{t,d}$$

$$idf(t, D) = \frac{|D|}{|\{d \in D : t \in d\}|}$$

Here  $f_{t,d}$  denotes the raw term frequency of  $t$  and  $|\{d \in D : t \in d\}|$  is how many documents  $t$  appears in. A high measure weight for the term will then be given by it having a high frequency in the document, but a low frequency in the overall collection.

## Conceptual design

This chapter will explain how the system will extract examples, add them to the database and make them searchable for an end user, on a conceptual level. It will start by briefly explaining the overall approach, before going more into details regarding the pipeline, and how the examples produced by the pipeline are managed for search purposes. The chapter will finish with an analysis of examples, to help us better understand them, which in turn can improve the performance of the system when it is implemented.

### 3.1 Overall Approach

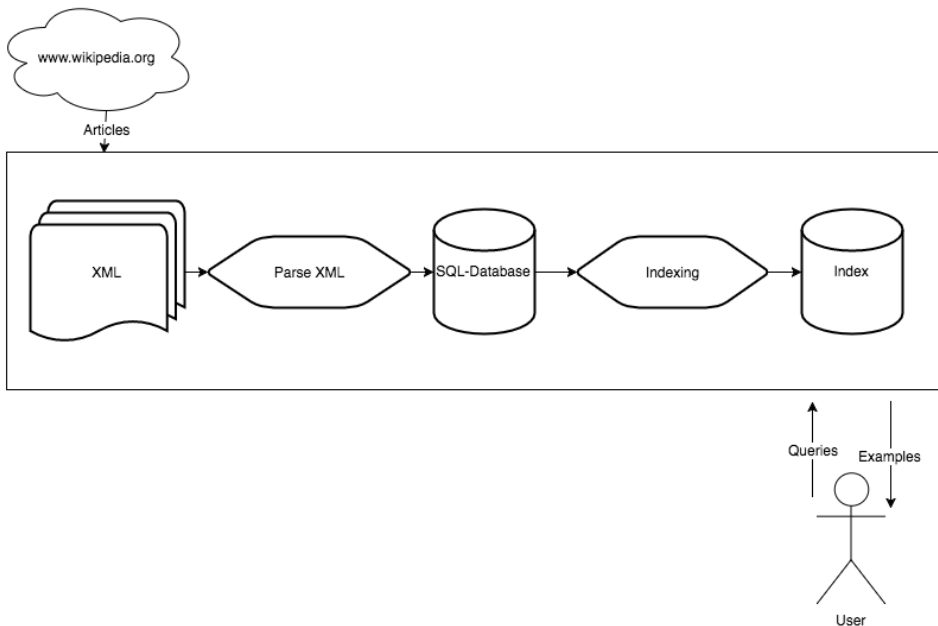
The goal of our approach is to create a searchable index for a collection of examples. The first step is to find examples and store them in a database. Wikipedia was chosen as a resource for extraction of examples. Wikipedia's articles have a reliable and consistent format. It is also the largest collection of open information, which gives us a sufficient amount of source data to work with. The source data is acquired from a XML dump that Wikimedia publishes regularly. Except for the markup itself, the articles contain no structure. From the dump we obtain the article's markup and additional meta data. Therefore a considerable amount of processing is required to

---

locate and extract relevant examples that can be inserted into the index. To achieve this we created a software pipeline where several independent processes work together by using the output from one process as input to the next process. We start by feeding the pipeline with the raw XML file. The pipeline parses the XML and markup of each article. The relevant articles are stored in a relational database. A new process creates examples based on the data in the database. Last, an index is built from these examples.

## 3.2 Pipeline

### 3.2.1 Creating the pipeline



**Figure 3.1:** A conceptual overview of the pipeline used to extract and index examples

Based on what was discussed in section 2.1.4, it was decided not to use the SMILA system and its utilities itself, but instead look at how SMILA approaches the problem and take inspiration from that. Therefore other



---

methods were considered, which mainly consisted of building the pipeline from scratch. Tailoring the different sub-processes for our project and organizing them in a pipeline, was decided to be the best course of action. The first issue to be settled regards the format of the source data. SMILA fetches information by crawling the web, in our case Wikipedia<sup>1</sup>, but using a web-crawler is a very general method. Since our project will only use Wikipedia, the extraction method could be more specific. An XML-dump from a snapshot of Wikipedia's database fits perfectly in terms of simplicity and stability for our project. The dump contains all the articles of Wikipedia in their newest version.

To transform the XML-dump into useful data, several sub-processes will be utilized. All the sub-processes in the pipeline are loosely coupled, which means that they are fully independent given the correct input. Being fully independent allows the pipeline to easily swap out or alter sub-processes without affecting the rest of the system. The SMILA pipeline also delivered a high degree of parallelity. Parallelity is a property the current version of the pipeline created from scratch does not have. The pipeline implemented will not focus on parallelity because functionality will be prioritized higher than efficiency and performance.

### 3.2.2 Extracting data from XML-dump

Parsing the source data and then extracting examples from it, is the first task the software pipeline performs. The source data used for input to the pipeline uses the XML format. On the top level of the XML documents there exists article elements marked with an *article* tag. Each article element represents a page on Wikipedia and thus contains metadata about the page as well as the content itself. All the data forms an XML document with a size of 50 gigabyte. Because of the document's huge size, the process reads the XML-document as a stream, buffering line by line, and then identifies article elements and saves the relevant data from it to an object in memory. For each article, the process looks at the sections and selects the ones that contains an example. It parses those articles into an object with the relevant data. A relational database is created to store information

---

<sup>1</sup>[www.wikipedia.org](http://www.wikipedia.org)

---

about the examples. When iterating over the articles, the process populates the relational database with data from relevant sections and articles. The database stores meta data from the articles, content from specific sections and references in the sections. Relations between articles, sections and references are also stored, and are used in the next step of the pipeline, when the data will be used to form example entities.

### **3.2.3 Building an index of examples from SQL**

The first step in building the index is fetching data from the database to form examples. Virtual tables called *views*, are used in the database to normalize the data on a format which the index will be using.

Elasticsearch was chosen as the tool for building the index. Section 4.3.1 explains why it is a good choice for this project. A simple Java process fetches the views from the database and creates objects representing examples. References to other examples and which categories the examples are also added here. The example model has a one-to-one relationship to the examples represented in the Elasticsearch index, so the objects are directly converted to JSON and inserted into the index. To configure and query the index, Elasticsearch serves a HTTP API. The API is used before building of the index to specify behaviour of different fields, and after for searching the complete index.

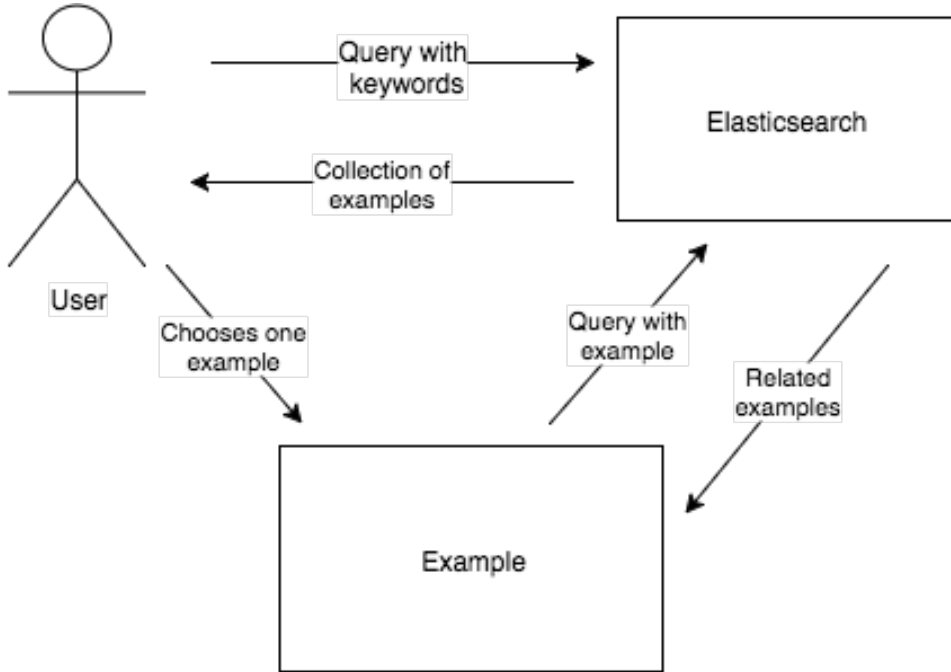
## **3.3 Search examples**

### **3.3.1 General idea**

When the source data consisting of Wikipedia pages has been processed through the pipeline described in section 3.2, the final result is an index built up of examples. Having just an index is not very useful without an easy way of interacting with it though. Therefore a user friendly, web based search interface has been created. Figure 3.2 displays the interaction between the

---

user and Elasticsearch with the two methods for querying, which are by using keywords or a selected example.



**Figure 3.2:** A simple overview of the user’s interaction with the examples

The main objective of the interface is to assist the user in finding relevant examples and to present these examples to the user in a helpful manner. To present the examples, the interface use HTML and CSS from the live version of Wikipedia from its web server. To find relevant examples, the interface helps the user in two ways. The first one is simply using keywords entered in a search field by the user. The second is when the user has already selected an example that is relevant, the search interface will then use this example to find similar examples.

### 3.3.2 Querying by keywords

The user’s first interaction with the interface is by interacting with the search field. In this field the user will type in keywords which will be used

---

in a query sent to Elasticsearch. Elasticsearch will then use the TF-IDF algorithm to match the search phrase to fields contained in the example. The fields used for matching the search terms are introduction, content, title and categories. By boosting the importance of some fields, the query can be fine tuned into delivering a more relevant collection of examples based on the keywords in the search phrase.

### **3.3.3 Querying by examples**

The collection of examples returned by the user's first query is presented in a list. The list allows the user to browse through the returned examples and select an example. The selection of an example triggers a new query, which is automatically sent to Elasticsearch and executed. The query uses data from the selected example and the previous search. By comparing the categories of the selected example, and categories from the search, the most popular category is identified. The most popular category is a category from the selected example's set of categories, and appears most times among the best results of the initial search. The most popular category is then used by Elasticsearch to retrieve related examples. The set of returned examples are then rated after how good a match they are to the selected example's set of categories, and then ordered in two lists based on their rating. One list contains examples with a perfect match, while the other contains the ten best examples that partially match.

## **3.4 Analysis of examples**

### **3.4.1 An example**

An example is used as a tool to better understand a topic. It is usually used together with an explanatory text, where the example is a minor part of it. Examples are rarely presented standalone since they often required a certain degree of context. If this context is fused into the example, it often tends to make the example very complex and too troublesome to use efficiently.

---

Because of this, an example can often be looked at as an appendix to a text regarding a subject.

People who want to learn about a certain topic, but already know the basic context may prefer to skip the explanatory text and only look at the examples. This project tries to exploit the fact that examples act as an appendix to a subject. Therewith, we attempt to give persons trying to expand their knowledge within a topic a more preferable way of doing so. In order to develop a better tool helping these persons, Research Goal 2 focuses on reaching a deeper understanding of examples. We hope that a better understanding of examples will lead to more success when finding search results and recommending related examples.

### **3.4.2 Comparing examples**

Finding common properties of examples is very difficult and nearly impossible if compared across different domains. When comparing two examples within the same domain and topic, there will still be different approaches and techniques of explaining, which still makes it hard to directly compare them. As a consequence, a highly qualitative and subjective analysis has been made for examples within the Game Theory domain.

Different key properties have been identified and examined across different topics. Some properties are connected to the structure and presentation of the example, while others are based on the contents. Through examining topics within Game Theory, a list of properties of an example were designed. Table 3.1 displays these properties with a brief explanation.

### **3.4.3 Structure of a good example**

The structure of an example is mostly defined by how it utilizes the properties mentioned in table 3.1. Simply checking which examples has the most properties is inaccurate and misleading. A good example does not need to have all the properties and counting them will not result in finding the best. Instead the combination of the values these properties contain are sig-

Name	Description
Figures	Graphics and drawings referenced in the example.
Length	The length of the example, can be both words and characters.
Domain	Which domain the example belongs to.
Source code	If source code is used for explaining principles.
Equations	If equations are used to explain the example
Analogies	When an real world example is used to explain a theoretical one.
Subsections	An example that is divided up in smaller parts or sections that offer different angles of explanations for the example.
Walkthrough	A step by step guide that solves a problem.
Iterations	Several iterations that gradually increase the complexity of the explanation.
References	How often the example refers to figures or other examples and articles.

**Table 3.1:** Properties of an example and a description of them.

nificant, especially the property describing the domain, as some domains could generally benefit from a different composition of properties than others. This leads to a more narrow approach domain-wise, when looking into different examples. When trying to identify the structure of a good example, there have been selected a few topics within the Game Theory domain. Then for a certain topic, for instance *Pareto Efficiency*, two or three examples have been compared to each other.

For the topic *Pareto efficiency* two examples were examined. Pareto efficiency is about reaching a state of allocation of resources where it is not possible to make one part better of without making another part worse. The first example is named *Examples and exercises on Pareto efficiency*. This example builds up its explanation step by step with several iterations. Each iteration is a bit more complex than the previous. This enables the example to cover the topic with a certain depth. The second example is named *Robinson Crusoe example*. This example is very long and detailed. In addition to a large amount of text, it also uses equations to a substantial degree. Initially the first example seems better, it definitely makes it easier

---

to grasp the concept of *Pareto efficiency*. The second example focuses more on the mathematical part, and there will most likely exist some persons who prefer this. See appendix B for both examples.

---



# Implementation

This chapter will explain in detail how the final system was implemented. Based on the concept described in section 3.2, a pipeline will transform Wikipedia's source data into structured example objects. The system's architecture and the format of the source data will be displayed in figures, which will explain how the system creates examples from the source data. The last part of the chapter will mention the tools used in our project, that were of great significance.

## 4.1 Pipeline

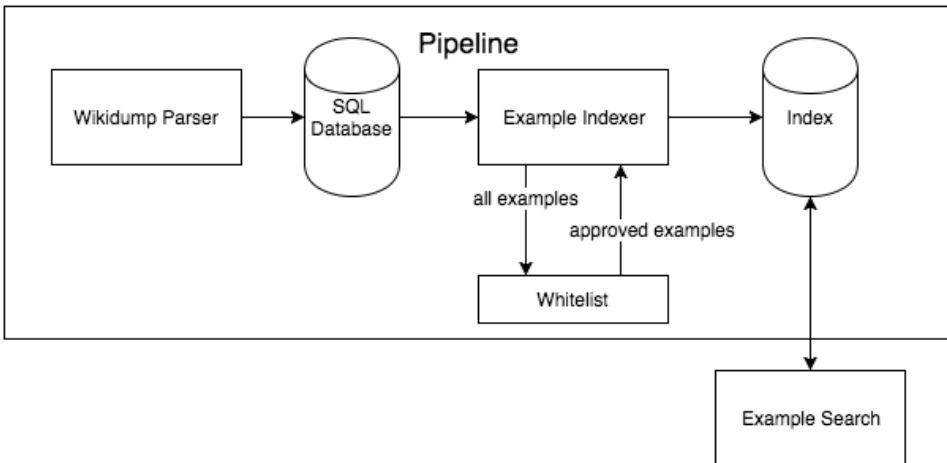
### 4.1.1 Overview

*Wikidump Parser* is the first part of the pipeline. It transforms the XML dump of the whole Wikipedia into a database with relevant data from articles. The Wikidump Parser is written in JavaScript and executed by a Node.js process. To insert the relevant data from the XML into a database, it parses the XML, then parses the Wiki markup and lastly inserts relevant articles into the SQL database. These separate processes are managed by a master process executed from the main file `index.js`.

---

The second part of the pipeline is called *Example Indexer*. Example Indexer fetches data from article, section and category tables in the SQL database and combines the data into examples. To fill the index with examples related to specific domains, a whitelist is used to include examples based on their categories before it builds an index in Elasticsearch with these examples.

*Example Search* is the last part of the pipeline. When the pipeline is finished with processing the data into examples, Example Search can fetch the examples from the index, and display them for the user. Example Search is a web based search interface that queries the index over HTTP and displays the results.



**Figure 4.1:** Architectural overview of the pipeline

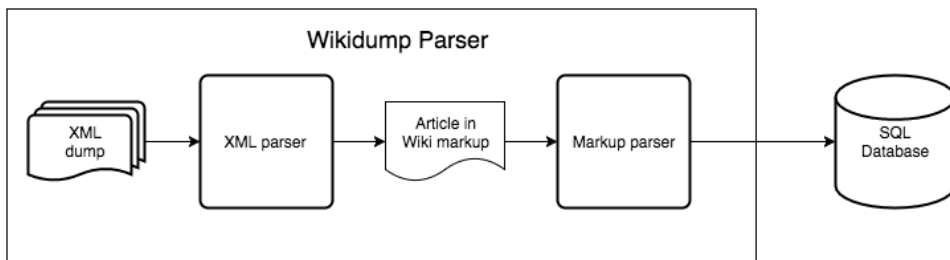
Figure 4.1 displays these three parts of the whole system, where arrows indicates the data flow. The different parts of the system does not communicate with each other directly, but by inserting and fetching data from permanent storage.

---

## 4.1.2 Wikidump Parser

### 4.1.2.1 Architecture

Wikidump Parser has a very simple architecture. As shown in table 4.2, two main processes are responsible for inserting relevant articles and sections from the XML dump into the SQL database. XMLParser uses a stream to parse the XML and uses an event pattern to notify every time an article has been parsed. Markup Parser then parses the article received from each event. If the article contains an example, Wikidump Parser inserts it into the SQL database.



**Figure 4.2:** Architectural overview of Wikidump Parser

### 4.1.2.2 XML parser

The data dump that is fed into the beginning of the pipeline is on the XML format. Figure 4.3 shows how the XML is structured. The file is composed of page elements at the top level with the page tag. Each *page* element represents an article in Wikipedia. The page has several sub elements, but the most interesting element, is the one with the revision tag. Wikipedia saves several revisions of a page, but in the XML dump used, only the newest revision is included. It is inside this element that we find the article's content in the form of Wiki markup. Line number 18 in figure 4.3 contains a comment marking where the whole article's markup is found.

In our project the JavaScript class `XMLParser.js` handles the parsing of XML. It detects where an article in the XML document starts and

---

```

1 ▾ <page>
2   <title>Anarchism</title>
3   <ns>0</ns>
4   <id>12</id>
5 ▾   <revision>
6     <id>703037144</id>
7     <parentid>702960614</parentid>
8     <timestamp>2016-02-03T02:56:04Z</timestamp>
9 ▾     <contributor>
10       <username>Graham11</username>
11       <id>14835592</id>
12     </contributor>
13     <minor/>
14     <comment>Rm line break</comment>
15     <model>wikitext</model>
16     <format>text/x-wiki</format>
17     <text xml:space="preserve">
18       ##CONTENT IN WIKIMEDIA MARKUP
19     </text>
20     <sha1>jffzqlqrr78pbadoelp50j78re1kjf7</sha1>
21   </revision>
22 </page>

```

**Figure 4.3:** The XML structure used in the XML dump of Wikipedias database

where it ends by looking at the element tags, then proceeds to send the content to `index.js`. The document is read as a stream, which gives the advantage of only keeping a small bit of the document in memory. The library `sax`<sup>1</sup> is used to read the document as a stream. Sax then creates events indicating where in the document it is currently reading, and what elements it is entering or exiting. Our project listens to the events for an open tag, close tag and content. By saving the state of which element Sax currently is inside we extract content, title, id and timestamp from the article element. When we detect the end of an article element, the data extracted is passed to `index.js`.

---

<sup>1</sup><https://www.npmjs.com/package/sax>

---

### 4.1.2.3 Markup parser

When extracting data from Wikipedia articles, the article's markup is parsed. The markup is called *wiki markup*, figure 4.4 demonstrates how the markup might look like for an article. When looking at the XML in figure 4.3, the entire wiki markup for an article is found at line 18.

```
...Now consider the game in which we alternate the two profiles while judiciously
choosing the time between alternating from one profile to the other
[[Image:Parrandos Paradox Biased Games.PNG|right|thumb|Figure 2]] A
...
===The coin-tossing example=== B
A second example of Parrondo's paradox is drawn from the field of gambling. Consider
playing two games, ''Game A'' and ''Game B'' with the following rules. For
convenience, define  $C_t$  to be our capital at time 't', immediately
before we play a game.
# Winning a game earns us $1 and losing requires us to surrender $1. It follows that
 $C_{t+1} = C_t + 1$  if we win at step 't' and  $C_{t+1} = C_t - 1$ 
if we lose at step 't'. C
# In ''Game A'', we toss a biased coin, Coin 1, with probability of winning  $P_1 = (1/2) - \epsilon$ . If  $\epsilon > 0$ , this is clearly a losing
game in the long run.
...
D
It is clear that by playing Game A, we will almost surely lose in the long run.
Harmer and Abbott<ref>G. P. Harmer and [[Derek Abbott|D. Abbott]], "Losing
strategies can win by Parrondo's paradox", 'Nature' '402' (1999), 864</ref>
show via simulation that if  $M=3$  and  $\epsilon = 0.005$ ...
```

**Figure 4.4:** A short outline from an article's wiki markup, with important aspects highlighted and marked.

Figure 4.4 points out some important aspects of the markup language. For a complete guide, Wikipedia has its own help page<sup>2</sup>. Mark A shows how to refer to an image or other graphic. The first piece of information is about where to find it in Wikipedia's database, while the rest is for the visual presentation of the image. Mark B is the most important markup when dividing the article into sections. The equal signs at each side indicates a header for a section. The number of equal signs range from one to six. One is the Article title, two is section, three is subsection and so on. By using the hierarchy, we can both separate out sections from each other and find each sections parent section.

---

<sup>2</sup>Help:wiki markup - [https://en.wikipedia.org/wiki/Help:Wiki\\_markup](https://en.wikipedia.org/wiki/Help:Wiki_markup)

---

The markup by Mark C is used for special formatting between the tags. In the case of figure 4.4, mathematical expressions are shown. Code is another format that is often included in examples, it is tagged in the following way; `<source lang="java">`, where the value of `lang` is the programming language. Mark D shows how one Wikipedia article refers to another. When parsing the sections, all these references are stored. These references are later used for finding relations and relevance between two articles. The words right of the pipe is the reference name, while the ones on the left are the names of the articles that are linked to.

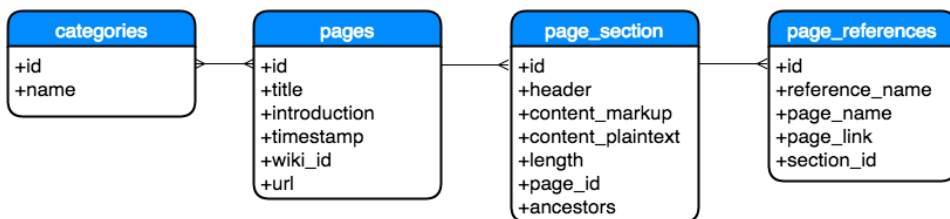
Each article sent to `index.js` from `XMLParser.js` is assessed on whether it contains examples or not. Articles not containing any examples, are discarded completely from the final collection. In order to assess the articles, the markup is parsed into objects. This is the task of `MarkupParser.js`. The parser uses several steps to extract the desired information from the markup. The first step parses the entire article's markup. The result of the first step is objects for each section, including the introduction. The sections contain a header, what level they belong to in the section hierarchy and the content of the section.

The next step iterates through the list of sections to determine if some of them is an example. Sections that are not an example is discarded. If no examples are found, the whole article is discarded at this point. References to other articles and categories are then extracted from the markup. The data is finally inserted into the database pictured in figure 4.5.

### 4.1.3 SQL Database

Section 3.2.1 explains that the source data is fed into the pipeline as a snapshot of Wikipedia at a particular time. The size of the source data is an approximately 50 GB XML-document. Most of the data has no link or relation to examples, making it uninteresting for this project. Therefore Markup parser excludes most of it before it is organized in the SQL database.

Figure 4.5 displays the tables of the database and the relations between them. If an article with a relevant section is discovered, the article is in-



**Figure 4.5:** A simple overview of the database tables and their attributes

serted into the *pages* table in the database, with only metadata and the article’s introduction. The relevant sections are then extracted from the article and stored in the database with relation to the article. The categories of the article is also kept, because it will be helpful when searching among examples in the finished index. At the end of the process of inserting one example into the database, the references are stored with a connection to the section they are used in.

The database acts as a temporary buffer for the data going into the index. Having a buffer separates the parsing, from the building of the index. Therewith changes made in the parsing, will not affect the rest, so the database acts as an interface between the parser and the indexer. The data is stored on disk, so also the point in time when the programs are executed can happen independently. This is preferred since the parsing is a very time consuming process, using about nine hours completing a full parsing run. Having it structured in SQL with its metadata helps showing how successful the parsing was. Some of the metadata is also irrelevant for the index, so only keeping it in the database makes the end result less complex.

#### 4.1.4 Example Indexer and Whitelist

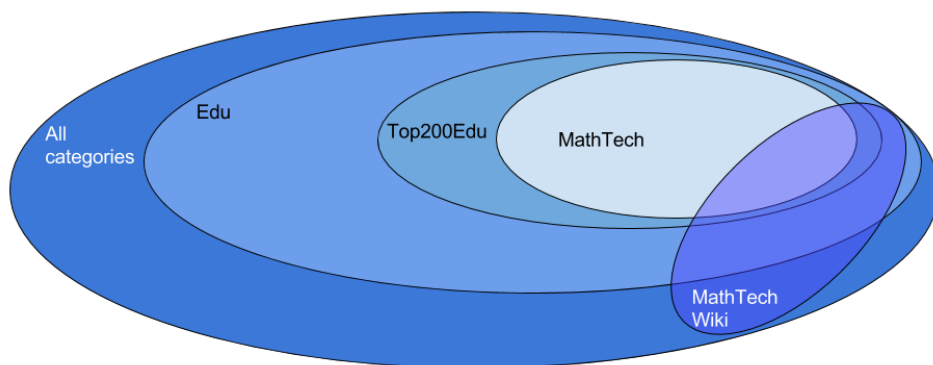
A Java program queries the database for all sections. The relations and data from the other tables are incorporated into the section. Now the section is an object with all the data needed to independently represent an example.

Although all the sections extracted from Wikipedia articles are examples, not all are relevant for a specific search purpose. Therefore the ex-

---

amples can be filtered through a *whitelist* before they are inserted into the index. By filtering the examples before adding them to the index, we can assure that only examples of interesting domains are added. This can then improve the results returned from the index after a search. Appendix C contains an example of a whitelist used to keep desired examples. A total of 4 whitelists have been created, to be used in different combinations. The whitelists can be found in the Example Indexer resource folder, at `src/main/resources`. The whitelists were created by extracting the complete list of all categories from the SQL database. Next, the categories were sorted by how many articles that had a relation to them. Finally, the list was manually altered into three different versions. A fourth version was created based on Wikipedia's category hierarchy. If one of the categories linked to a specific example also exists in the applied whitelist, that example will be included in the index's corpus.

The purpose of using whitelists was to reduce the amount of examples in the index. The whitelist can then reflect a chosen domain by including only examples related to predetermined domains, consequently giving the corpus a specific scope. A more specific scope for the corpus, should improve the quality of the results returned.



**Figure 4.6:** A venn diagram showing how whitelists act as subsets of all categories

The four different whitelists were given the names *Edu*, *Top200Edu*, *MathTech* and *MathTechWiki*. The names reflect the set of categories included for each whitelist. *Edu* and *Top200Edu* contains the categories which are most popular and also educational. *Edu* contains all educational



topics, while *Top200Edu* is restricted to the top 200 categories. *MathTech* narrows the selection further by only including categories related to mathematics and technology. The last list, *MathTechWiki*, is based on the main categories from Wikipedia’s main category overview<sup>3</sup>. Mathematics, Logic, Mathematical Sciences, Computing and all of their immediate sub-categories are included. Figure 4.6 demonstrates how one whitelist is a subset of another category list.

Whitelist	Categories	Examples
No whitelist	17 170	28 110
Edu	17 132	22 977
Top200Edu	200	6 364
MathTech	157	5 037
MathTechWiki	64	1 003

**Table 4.1:** Statistics for examples and categories included when applying different whitelists for filtering of the example collection.

Table 4.1 display the statistics after each whitelist has been used on its own to filter examples. Table 4.1 shows total number of categories included in each list and how many examples included in the final corpus. When building our index, *Top200Edu* and *MathTechWiki* are being used, which keeps examples related to the most popular and important categories. Examples only related to less popular categories or irrelevant ones, are as a result discarded.

Before the examples are inserted into the index, the index is created with the Elasticsearch java API<sup>4</sup>. Our project uses the default settings, so only the cluster name is needed to be defined beforehand, the rest happens automatically by Elasticsearch. Lastly a mapping for the example object is passed to Elasticsearch. Figure 4.7 shows the mapping passed. It defines all the fields with its types. The mapping sets *categories* to *not\_analyzed*, so it will match the exact category names.

<sup>3</sup><https://en.wikipedia.org/wiki/Portal:Contents/Categories>  
(Last visited 7. Mars 2016)

<sup>4</sup><https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/index.html>

---

```
1 PUT wikipedia/
2 {
3   "mappings": {
4     "example": {
5       "properties": {
6         "categories": {
7           "type": "string",
8           "index": "not_analyzed"
9         },
10        "content": {
11          "type": "string"
12        },
13        "header": {
14          "type": "string"
15        },
16        "id": {
17          "type": "long"
18        },
19        "introduction": {
20          "type": "string"
21        },
22        "markup": {
23          "type": "string"
24        },
25        "pageId": {
26          "type": "long"
27        },
28        "title": {
29          "type": "string"
30        },
31        "url": {
32          "type": "string"
33        }
34      }
35    }
36  }
```

**Figure 4.7:** Mapping used to define an example in Elasticsearch

The examples needs to be converted into JSON format first in order to put them into the index. A simple method maps all fields over to their equivalent attributes. Attributes in JSON are key-value pairs, key name and the field's value. The Java API sends each example to Elasticsearch as a document on the JSON format, which then indexes it.

---

## 4.1.5 Index

As already mentioned, Elasticsearch is the chosen tool for indexing the examples. By using its simple RESTful API, examples can be stored as JSON documents, which Elasticsearch then indexes automatically by detecting the example's data structure and type. Hence, we can focus more on the queries used to retrieve examples, instead of building the index. Elasticsearch manages the whole index, and leaves a simple web based interface for communication.

During this project, Elasticsearch was run on the same computer as both Example Indexer and Example Search. Therefore the communication between these programs happens with HTTP request over localhost. Localhost is used when communicating between different processes on same computer. By resolving to the IP address 127.0.0.1, which is a loopback address, only the right port is needed. Consequently, Elasticsearch, Example Search and Example Indexer, which all are web applications, communicates with each other without an internet connection. During the project, having them all on same computer was the easiest option. For possible future use, hosting the index on a separate server, only the ip address would have to be changed.

Although Elasticsearch mostly handles the customization for us, we have chosen the cluster and node setup. All the data is gathered in one cluster, running on one node, named `wiki_cluster`. There is also only one index, `wikipedia`. To make the index as simple as possible, only one document model exist to represent the examples, which naturally is named `example`. A query for a specific example will then look like this:

```
GET http://localhost:9200/wikipedia/example/1
```

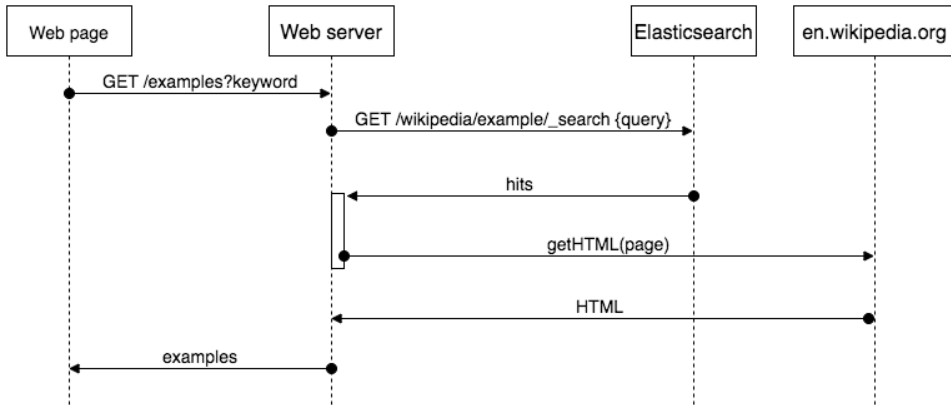
The request above specifies 1 at the end of the line, which makes Elasticsearch return the example with id 1. The web server mostly uses queries for its searches, here is a possible example of how to perform a query:

```
GET
'http://localhost:9200/wikipedia/example/_search' -d '{
  "query" : {
    "term" : { "title" : "Prisoners Dilemma" }
  }
}
```

---

The above search only makes use of the examples title field, while the different queries performed by the search interface utilizes most of the fields of an example.

## 4.2 Example Search



**Figure 4.8:** A sequence diagram showing the interactions between the actors when searching for examples

Example Search is responsible for managing queries passed from the user. It serves a web page that allows the user to enter queries for examples. Express<sup>5</sup> is used as a framework to effortlessly set up a web application with a HTTP API.

For the web page, Jade<sup>6</sup> is used as a template engine to build the HTML structure. Using a template engine allows easy reusing of markup code. Also control structures and programming statements can be used directly in the file. Jade offers its own syntax that is faster to write than the standard HTML syntax. For simplicity, the web page is mainly styled by Wikipedia's style sheets, with a few adjustments. When the user wants to query for

---

<sup>5</sup><http://expressjs.com/>

<sup>6</sup><http://jade-lang.com/>

---

examples, the web page offers a search box. The search box sends a HTTP request to the web server with the query.

For the web page to return examples after a search, a series of requests are made to different actors, figure 4.8 gives a quick overview of the actors and their interaction. The interaction starts when the server receives the HTTP request, Express routes the request to the right function. The routing functions can be found in `routes/index.js`. When the request is received, a new function in `elasticsearch/api.js` is called. `api.js` queries Elasticsearch directly by using the official Elasticsearch JavaScript library<sup>7</sup>. Elasticsearch divides the keyword into terms, then it matches the terms with the *inverted index* created from all the examples. Finally Elasticsearch uses TF-IDF to measure the relevance of the examples. The response is sent back, ordered by relevance, to the routing function asynchronously using a callback.

To display the page in the same way that Wikipedia does, the article's link is used to scrape the page from `en.wikipedia.org`. By scraping the page, we get access to the complete HTML structure of the actual page. From the HTML structure the example section is extracted. A list of all the relevant examples with their corresponding HTML is sent back to the web page in JSON format.

The web page uses AJAX<sup>8</sup> to handle the response from the server and displaying the results in a table. In the table a link for each example is created. This link leads to a view where only that example is displayed. In addition, related examples are shown. The related examples are found by sending a new request to the web server. The web server forms a new query to Elasticsearch. By using the selected example and results from the previous search, the web server tries to return the most related examples. First an algorithm detects the most popular category based on the first search. Then the most popular category is used to retrieve all examples that is related to it. Finally, a formula is used to measure how related the examples actually are. The formula used to sort the relatedness of examples based on categories are:  $\frac{|M|}{|C|}$  where M is matching categories one related example has with the selected example and C is all the categories for the selected

---

<sup>7</sup><https://www.npmjs.com/package/elasticsearch>

<sup>8</sup>AJAX - Asynchronous JavaScript and XML

---

example. With the selected example and its related examples returned to the web page, the user can browse between examples by using the links created for each example.

## 4.3 Tools

### 4.3.1 Elasticsearch

Elasticsearch is a tool used in this project for indexing the examples. Elasticsearch is built on top of Apache Lucene<sup>9</sup>, which is a information retrieval library, written in Java. Internally in Elasticsearch, data is stored as structured JSON<sup>10</sup> documents. The API<sup>11</sup> for communicating with Elasticsearch is a RESTful<sup>12</sup> API using JSON over HTTP. The API can be used for configuring Elasticsearch, building the index and querying it.

Elasticsearch is built for scalability. Being scalable means handling growth of both the dataset and interactions on it. Elasticsearch scales by having a cluster of many nodes. If the system needs to scale, new nodes can easily be added, and Elasticsearch will distribute resources to the newly added nodes. Different nodes can exist on different servers. However this project does not need or take advantage of this scaling, and will only run on one node.

There are two ways of implementing a search in Elasticsearch, *filter* and *query*. The *filter* is utilizing *terms* to decide whether a document should be returned or not. Searching with a *term* is very similar to how one would use SQL. Searches can for instance consist of text strings, numbers, ranges or dates, and Elasticsearch will return everything that matches. It also allows for boolean operators and nesting of these. Using a *filter* is very quick and should be used if the relevance of the documents is insignificant.

---

<sup>9</sup><https://lucene.apache.org>

<sup>10</sup>JSON - JavaScript Object Notation

<sup>11</sup>API - Application Programming Interface

<sup>12</sup>REST - Representational State Transfer

---

If relevance score is important then the second option, *query*, should be chosen. If a *query* is combined with a *term*, Elasticsearch is looking for the exact value in its index. A score is then returned based on the documents TF-IDF relevance to the term. If a *match* is used instead, an analysis will be performed, creating a list of terms from the query, and then executing low-level queries for each of the terms. The results are combined to produce the final relevance score. These two methods can also be combined or extended with other methods to customize the search further.

### 4.3.2 NPM and Node.js

In our project, both the programs Wikidump Parser and Example Search are written in JavaScript, which is normally executed by browsers. Node.js[18] allows the code to be executed from a terminal instead, which enables JavaScript to be used on servers. Node.js is an asynchronous event driven framework. By embracing to event loop in this manner, Node.js avoids thread management and blocking of those. Instead callbacks are pushed to the event loop and Node.js runs until there are no more callbacks to perform. This makes Node.js ideal for simpler and less complex systems, and is therefore chosen for this project.

Node.js also comes with a packet manager called Node Packet Manager(NPM). NPM allows any Node.js project, to include libraries and other JavaScript projects published to their *Open Source Registry*<sup>13</sup>. This is done by a simple API call to the NPM executable in the terminal.

Using code from open source libraries saves a lot of time during development while still having full control and overview of the code executed. Hence we decided to include several libraries. Examples of libraries used are *sax* for reading the XML file line by line as a stream. *wtf\_wikipedia*<sup>14</sup> to parse some of the Wikipedia markup; *request*<sup>15</sup> to fetch a HTML file from a server with a GET call; *cheerio*<sup>16</sup> for iterating through an HTML structure while supporting filtering, reading and editing of the HTML. Using

---

<sup>13</sup><https://www.npmjs.com/npm/open-source>

<sup>14</sup>[https://www.npmjs.com/package/wtf\\_wikipedia](https://www.npmjs.com/package/wtf_wikipedia)

<sup>15</sup><https://www.npmjs.com/package/request>

<sup>16</sup><https://www.npmjs.com/package/cheerio>

---

libraries results in the system being more modular, which makes it easier to alter during development. This has been highly advantageous for this project, since requirements have been continually changed.



## Experiments, Results and Discussion

Chapter 3 discussed the different concepts regarding the pipeline and the examples data managed by it. Chapter 4 explained how the pipeline is implemented. In this chapter we will use the information made available in Chapter 3 and 4 and explore to which degree the research goals have been accomplished.

First, for Research Goal 1 we will decide whether the pipeline created is satisfactory or not. Regarding Research Goal 2, we will conclude the analysis conducted in Chapter 3. Next, we will investigate if the database chosen to manage the collection of examples is satisfying the purpose of Research Goal 3. Finally, a series of experiments will be performed to evaluate the search results produced by the search interface. A comparison of the results will help to determine if Research Goal 4 is fulfilled.

Each section in Chapter 5 will discuss on of the research goals. Table 5.1 gives a quick overview of which research goals are being discussed by the different sections.

---

Goals	Section
1	5.1
2	5.2
3	5.3
4	5.4, 5.5 and 5.6

**Table 5.1:** Mapping of which sections discusses the different research goals.

## 5.1 Resulting Pipeline

We tested the final pipeline by using the English Wikipedia’s database dump from February 4, 2016. With a size of 56.3 GB, the XML file contains over five million articles. The process of extracting the relevant data and inserting it into the SQL database lasted nine hours. After the last section was added, the database contained 28 110 example sections deemed as relevant sections. The process was run on a mid end MacBook Pro from late 2013 with four cores, each with a processor speed of 2 GHz and 2x4 GB of memory with a speed of 1600 MHz.

When the process that extracts relevant sections were finished, a new fully independent process was started for the next step. First, the process queries the SQL database for sections<sup>1</sup>. By using the relations from the sections stored in the SQL database, the process builds examples. Next, a whitelist is used to filter the examples based on their categories. By filtering the examples we can avoid irrelevant categories, for instance examples concerning history. Finally, the collection of examples left are used to build an index with Elasticsearch. The Elasticsearch index offers an HTTP API that can be used to query for examples.

---

<sup>1</sup>A section is a part of an Wikipedia article.

---

	Value
XML file size in Gigabytes	56.3
Number of articles	5 100 000
Extraction duration in hours	9
CPU speed in GHz	2.0
Memory size in GB	8

**Table 5.2:** Results from processing the XML dump through the pipeline

## 5.2 Describing a good example

Finding examples that later can be presented to the user is the overall goal of our work. To make the system we have created perform well, we want to fill the database with useful data in the form of good examples. This project uses Wikipedia as a source to automatically identify and extract examples. To make the best out of the examples extracted, we performed an analysis on what differentiates a good example from a bad one. A more detailed account of the analysis can be found in section 3.4. This section will use the main points from section 3.4 to draw a conclusion.

The analysis was performed based on examples from Wikipedia, but to manually compare examples for the same topic, examples from a normal Google search were also used. Game Theory is used as an overall domain for the analysis. The examples were chosen to walk through the unimplemented system. The analysis let us discover how the examples would affect the system, and depending on the system's input, the structure of the examples could be fitted to the system's needs. The topics were chosen to reveal strengths and weaknesses of our approach. The following topics were chosen to find examples: Prisoner's Dilemma, Nash Equilibrium, Pareto Optimality, Zero sum, Parrondo's Paradox. Appendix B contains samples of two examples chosen for the subject Pareto Optimality.

Based on these subjects, a list of properties that examples might possess was compiled, table 3.1 contains these properties with a short description of each. The properties listed in table 3.1 are all favourable, but we experienced that when an example contains too many of them, the content of the

---

example becomes very complex. A complex example is not necessarily a bad thing. Although a more complex example requires more prerequisite knowledge from the reader, which may exclude some readers. The extent of use for each property is also a considerable factor for determining complexity. Although more properties often make the content of the example more complicated, some are always needed. A simple example will have a hard time explaining the more complicated aspects of a subject. Therefore an example should fall in between a golden mean of complexity. This golden mean is hard to define, but ought to be quite large. There is some techniques though, that lets an example explain more complicated subjects, without increasing complexity. Pictures or equations that are methodically referenced to from a descriptive text, is one way. Also the combination of the properties analogies, walk-through and iterations improve examples without making them more complex.

The analysis in section 3.4 gave a better understanding of examples. Consequently rating and ordering of examples can be improved based on the analysis. The improvement can result in the collection of examples having a higher quality and also lead to a better user experience. An instance were better understanding of examples can lead to better user experience is when the user is browsing through examples. The list of related examples can then be arranged in fashion that gives the user a natural feeling of progress when learning about a subject.

## **5.3 Collection of examples**

Research Goal 3 aims to obtain a collection of examples that we can populate a database with. The database accepts all kinds of examples, consequently the examples are represented in a generic way. We are using Wikipedia as source for all the examples, therefore we have to strip away a large amount of the information from the articles used.

After an article is extracted from the XML document, data is stored in a relational database. To form an example, data is fetched from the tables by using the relations defined in the SQL database. In addition to the content, categories and references associated with the example are included.

---

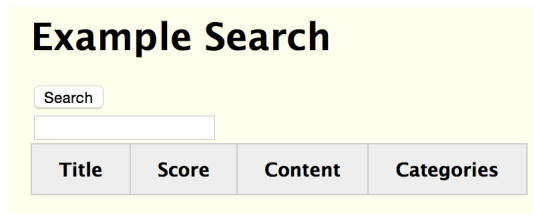
These properties are needed for an example in our system, so we later can perform queries on the collection. With Wikipedia having no limits for its domains, whitelists are used to narrow the examples down into only relevant domains for our project. During implementation whitelist *Top200Edu* and *MathTechWiki* were used.

After the processing, the collection now contains relevant elements, with a structure that represents a general example. This collection can then be indexed by Elasticsearch, which is the database containing all examples. Other examples from different sources can separately be included to Elasticsearch's example index, as long as the examples are structured correctly according to the mapping described in section 4.1.4. Table 5.3 shows how many examples were extracted from Wikipedia and how many examples that ended up in the final index.

	Value
Number of section after extraction	28110
Number of examples after filtering	6593

**Table 5.3:** Statistics from the index after original implementation.

## 5.4 Search interface

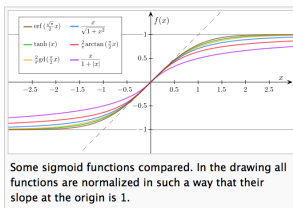


**Figure 5.1:** A screenshot of the search interface before a search is performed.

Figure 5.1 shows the search page of Example Search, before a search is performed. The design is very minimalistic with main focus on highlighting functionality. When a search is performed the now empty table will be

filled with results. The columns *Score* and *Categories* are mainly for help to evaluate and debug the system by using the returned results. (Would not be necessary in a finished version.) Meanwhile *Title* displays the name of the Wikipedia article and *Content* contains the actual example.

### Example Search

Title	Score	Content	Categories
<a href="#">Fuzzy logic</a>	1.0742466	<a href="#">Explore Example 2205</a> <b>Fuzzy logic: Define with sigmoid</b> $\text{sigmoid}(x) = 1 / (1 + e^{-x})$ $\text{sigmoid}(x) + \text{sigmoid}(-x) = 1$ $(\text{sigmoid}(x) + \text{sigmoid}(-x)) * (\text{sigmoid}(y) + \text{sigmoid}(-y)) * (\text{sigmoid}(z) + \text{sigmoid}(-z)) = 1$	Fuzzy logic, Artificial intelligence, Logic in computer science, Non-classical logic, Probability interpretations, Azerbaijani inventions
<a href="#">Sigmoid function</a>	0.9353394	<a href="#">Explore Example 3026</a> <b>Sigmoid function: Examples</b> <p>Many natural processes, such as those of complex system <a href="#">learning curves</a>, exhibit a progression from small beginnings that accelerates and approaches a climax over time. When a detailed description is lacking, a sigmoid function is often used<sup>[4]</sup>.</p>  <p>Besides the <a href="#">logistic function</a>, sigmoid functions include the ordinary <a href="#">arctangent</a>, the <a href="#">hyperbolic tangent</a>, the <a href="#">Gudermannian function</a>, and the <a href="#">error function</a>, but also the <a href="#">generalised logistic function</a> and <a href="#">algebraic functions</a> like <math>f(x) = \frac{x}{\sqrt{1+x^2}}</math>.</p> <p>The <a href="#">integral</a> of any smooth, positive, "bump-shaped" function will be sigmoidal, thus the <a href="#">cumulative distribution functions</a> for many common <a href="#">probability distributions</a> are sigmoidal. The most famous such example is the <a href="#">error function</a>, which is related to the <a href="#">cumulative distribution function (CDF)</a> of a <a href="#">normal distribution</a>.</p>	Elementary special functions, Artificial neural networks, Probability distributions
<a href="#">Gompertz function</a>	0.3973256	<a href="#">Explore Example 9638</a> <b>Gompertz function: Gomp-ex law of growth</b> <p>Based on the above considerations, Wheldon<sup>[6]</sup> proposed a mathematical model of tumor growth, called the Gomp-Ex model, that slightly modifies the Gompertz law. In the Gomp-Ex model it is assumed that initially there is no competition for resources, so that the cellular population expands following the exponential law. However, there is a critical size threshold <math>X_C</math> such that for <math>X &gt; X_C</math> the growth follows the Gompertz Law:</p> $F(X) = \max \left( a, \alpha \log \left( \frac{K}{X} \right) \right)$	Curves, Demography, Statistical terminology, Time series analysis

**Figure 5.2:** A screenshot of the search interface after a search is performed, showing the returned results.

When a search has been entered, the table expands and displays the examples returned. Figure 5.2 demonstrates how the interface looks after the search term *sigmoid* has been used. Every example has a link as its first line, which leads to a page specific for that example. When the link is clicked, the web server starts the process of finding related examples by sending a query to Elasticsearch and evaluating the returned results. The

selected example and its related examples are then sent to the web page and displayed there.

This page can be seen in figure 5.3. The page has the main example displayed in the middle, while the columns at each side of it contains a list of related examples. The two lists contain the name of examples and a link to their own page. The *Total matches* list on the left side contains related examples with a total match, while on the right side is the *Partial matches* list, which contains top ten of the examples that partly match. A search based on the category sets from the original search and the main example is used to find and order the relevant examples.

<p><b>Total matches:</b></p> <p><a href="#">Gompertz function: Gomp-ex law of growth</a> - 1</p> <p><a href="#">Gompertz function: Example uses</a> - 1</p> <p><a href="#">Gompertz function: Growth of tumors</a> - 1</p>	<p><b>Gompertz function: Gomp-ex law of growth</b></p> <p>Based on the above considerations, Wheldon<sup>[6]</sup> proposed a mathematical model of tumor growth, called the Gomp-Ex model, that slightly modifies the Gompertz law. In the Gomp-Ex model it is assumed that initially there is no competition for resources, so that the cellular population expands following the exponential law. However, there is a critical size threshold <math>X_C</math> such that for <math>X &gt; X_C</math> the growth follows the Gompertz Law:</p> $F(X) = \max \left( a, \alpha \log \left( \frac{K}{X} \right) \right)$ <p>so that:</p> $X_C = K \exp \left( -\frac{a}{\alpha} \right).$ <p>Here there are some numerical estimates<sup>[6]</sup> for <math>X_C</math>:</p> <p><math>X_C \approx 10^9</math> for human tumors</p> <p><math>X_C \approx 10^6</math> for <a href="#">murine</a> (mouse) tumors</p>	<p><b>Partial matches:</b></p> <p><a href="#">Lemniscate: Lemniscate of Gerono</a> - 0.25</p> <p><a href="#">Lemniscate: History and examples</a> - 0.25</p> <p><a href="#">Lemniscate: Lemniscate of Booth</a> - 0.25</p> <p><a href="#">Lemniscate: Lemniscate of Bernoulli</a> - 0.25</p> <p><a href="#">Lemniscate: Others</a> - 0.25</p>
<p>• Curves • Demography • Statistical terminology • Time series analysis</p>		

**Figure 5.3:** A screenshot of the search interface when a specific example has been chosen.

## 5.5 Querying by keywords

### 5.5.1 Experiment I: Search examples

In this project we have filled a database with examples, indexed those examples and created an interface that we can use for searching them. In order to

---

perform searches, we have created queries that Elasticsearch executes. To evaluate how well these queries perform we have created a list of keywords. We evaluate the accuracy of the queries by measuring the precision of the results returned from a search with the selected keywords. Precision is the fraction of returned examples that are relevant. It is calculated as follows:

$$\frac{|R \cap E|}{|E|}$$

In the equation above  $R$  is the set of all relevant examples and  $E$  is the set of all the retrieved examples. In our experiments  $E$ , representing all retrieved examples, is limited to the first five results in the top 5 tests, and similarly first ten in the top 10 tests. By limiting the size of  $E$ , manual inspection can be used to check whether an example is actually relevant, which would also place it in the set  $R$ .

Keyword	Total hits	Top 5	Top 10
Logic	362	1	0.9
Programming	884	1	0.9
Heuristic	44	0.8	0.7
Algebra	1046	0.8	0.7
Game Theory	2757	1	0.9
Fuzzy Logic	365	1	0.7
Java	269	1	0.9
Bayes Network	345	1	0.9
Derivation	67	1	0.8
Chain Rule	530	1	0.6
Prisoner's Dilemma	39	1	0.6
Nash Equilibrium	101	1	0.8
Cartesian Product	789	0.8	0.6
Parrondo's Paradox	49	0.6	0.4
Zero Sum	1017	0	0

**Table 5.4:** The precision of the queries evaluated by a set of keywords and the top ten and top five results

Table 5.4 contains the keywords and the results of the experiment. The keywords were chosen from the domains mathematics, artificial intelligence and programming. Those retrieved examples that are faulty<sup>2</sup> or are

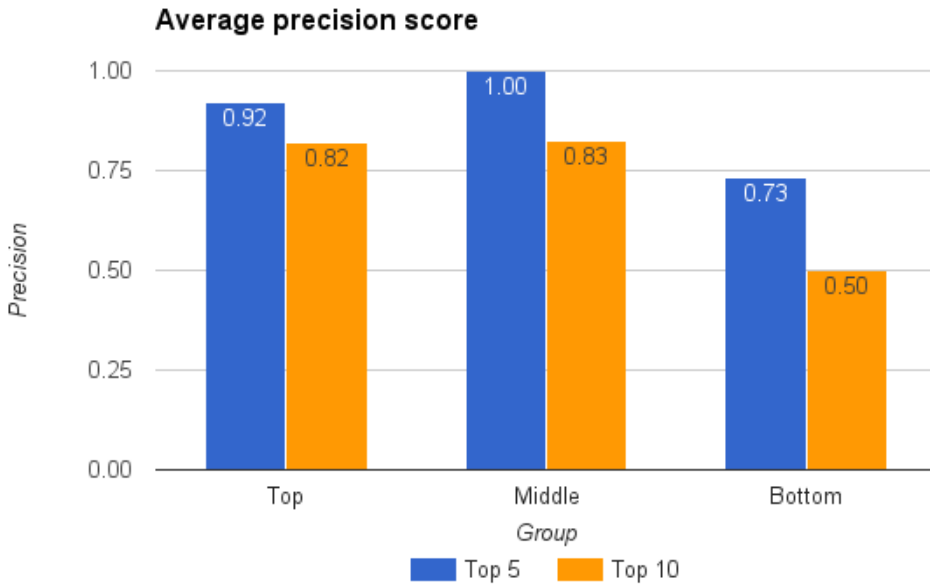
---

<sup>2</sup>Faulty examples can be caused by a bug in the pipeline or XML dump out of sync



---

examples for another subject are judged as an irrelevant result. Total hits reflects the amount of examples that the system found relevant to any degree. The different keywords have been divided into three different groups. The groups are listed in descending order by the degree of how general the keyword is perceived. The horizontal line in the table separates the groups. There is no specific order among the keywords within a group.



**Figure 5.4:** Average precision for the three different groups

Figure 5.4 shows how the three different groups' score in the experiment. The top and middle group has a very good score, both for top 5 and top 10, while the bottom group score a bit lower. The average score of top 5 and top 10 for all three groups is 0.8, which entails that only 2 out of 10 examples are not relevant. This indicates that the system does a good job retrieving relevant examples.

The score in the top group for top 5 and top 10 are very similar, with a 0.1 difference. In contrast the bottom group where the subjects are more specific, the score for top 10 results is remarkably lower with a difference

with the live Wikipedia page

---

of 0.23. Having a relatively small database of examples for specific topics is assumed to be the explanation, since the set of examples that can be recalled is not big enough. A richer selection of examples for the different topics could increase the precision for the bottom group.

Another interesting remark to note is that the top 5 results are consistently better than the top 10 results. Top 5 being better than top 10, is caused by at least half of the relevant examples which are returned, are found among the first five results. Therefore we can conclude that the ranking of the examples functions as intended.

There exist one substantial irregularity in table 5.4 though, the keyword *Zero Sum* achieve a score of zero on both measures, meaning all of the ten first results returned is deemed irrelevant. The fact that no relevant results are found among the first ten retrieved examples indicate that either there is a fault in the system or there is no relevant example in the system.

While performing the experiment, the error occurred when searching with the keyword *zero sum*. Upon closer inspection, we found out that there is a Wikipedia article called *Zero-sum game*<sup>3</sup> which the system is expected to retrieve. An error analysis will help determine why that did not happen, and in case of a system error, where in the system the error could have originated.

To find the potential error we start at the beginning of the pipeline, to explore whether the example is retrieved from the XML dump. We find the answer in the SQL database. First we find a row in the `pages` table with the name *zero sum game*. Next, using the primary key of this row, we can find all related rows in the table `page_sections`. A query on the column `page_id` in `page_sections` returns one result. This result is the same section found in Wikipedia's article, thus we can conclude that Wikidump parser successfully inserts the section into the SQL database.

The next step is to verify that the section is transformed to an example and indexed by Elasticsearch. The examples in Elasticsearch shares the same id as the sections primary key in the SQL database. Querying

---

<sup>3</sup>[https://en.wikipedia.org/wiki/Zero-sum\\_game#Example](https://en.wikipedia.org/wiki/Zero-sum_game#Example) (Last visited 28. April 2016)

---

Elasticsearch by id returns no results, therefore we know that the zero sum example never reaches Elasticsearch. Between the SQL database and Elasticsearch a whitelist filters the examples. If the zero-sum game article does not contain a category that exists in the whitelist, the example will be discarded. The article contains the categories *Non-cooperative games* and *International relations theory*. Matching the article's categories with the complete category list reveals that there exist only 4 occurrences of articles with Non-cooperative games as category and 7 with International relations theory, which is a low number. Whitelist *Top200Edu* and *MathTechWiki* were used for the filtering, section 4.1.4 explains whitelist's implementation and their function in our project. *Top200Edu* includes only the top 200 relevant articles and *MathTechWiki* use only the top level categories in the Wikipedia's category hierarchy. In conclusion, the article's categories are not included in any of the whitelists, and therefore the example is not indexed.

If the article's list of categories had been richer, the example would have been included more likely. Although richer category lists for Wikipedia's articles would have solved the problem, it is not something our project can affect. Instead a whitelist that accommodates the less popular categories is a better solution. The problem is that all the categories have to be manually added to whitelists. The Whitelist *Edu* deals with the problem by including all categories, but also examining the most popular and removing the irrelevant categories from the whitelist. The disadvantage obtained by using *Edu* is that irrelevant categories will be included, but on the other hand, they are not connected too many articles.

## 5.5.2 Experiment II: Whitelist

The result of the error analysis in section 5.5.1 points out the significance of the whitelists used for the end result. Based on that, Experiment II will be conducted to explore to what degree the whitelists affect the results returned, and which of them are best to use. The tables used to display the result, will be identical to table 5.4 used in section 5.5.1 with one exception. The tables in this section also include the percentage of hits compared to the number of examples in the index, when each specific whitelist has been

---

applied.

The first step in the experiment is to check what the system returns when there is no whitelists used. When no whitelists are used, there exist 28 110 examples in the index. Table 5.5 shows the results.

Keyword	Total hits	% of hits	Top 5	Top 10
Logic	898	3.19	1	1
Programming	2135	7.60	0.8	0.9
Heuristic	81	0.29	1	0.8
Algebra	1629	5.80	1	0.9
Game Theory	6301	22.42	0.8	0.9
Fuzzy Logic	913	3.25	1	0.7
Java	523	1.86	0.8	0.9
Bayes Network	1370	4.87	0.8	0.9
Derivation	178	0.63	0.6	0.5
Chain Rule	1760	6.26	0.8	0.6
Prisoner's Dilemma	141	0.50	0.8	0.5
Nash Equilibrium	350	1.25	1	1
Cartesian Product	2119	7.54	0.6	0.7
Parrondo's Paradox	159	0.57	0.2	0.2
Zero Sum	2330	8.29	0	0.1

**Table 5.5:** The precision of the queries when all examples are included.

---

In the next step whitelist *Edu* will be evaluated. *Edu* contains most of the categories from the original category list, with the exception of categories that are popular, but not educational, being removed. A category's popularity is measured in how many examples links to the category. When the examples are filtered through *Edu*, 5 133 examples are discarded from the original 28 110. The results of the evaluation can be found in table 5.6.

Keyword	Total hits	% of hits	Top 5	Top 10
Logic	737	3.21	1	1
Programming	1783	7.76	0.8	0.9
Heuristic	72	0.31	1	0.8
Algebra	1337	5.82	1	0.8
Game Theory	5230	22.76	0.8	0.9
Fuzzy Logic	750	3.26	1	0.8
Java	442	1.92	0.8	0.9
Bayes Network	1171	5.10	0.8	0.9
Derivation	145	0.63	0.4	0.5
Chain Rule	1374	5.98	0.8	0.6
Prisoner's Dilemma	115	0.50	0.8	0.5
Nash Equilibrium	60	0.26	1	1
Cartesian Product	1739	7.57	0.6	0.6
Parrondo's Paradox	128	0.56	0.2	0.2
Zero Sum	1842	8.02	0	0.1

**Table 5.6:** The precision of the queries when whitelist *Edu* is used to filter the collection beforehand

---

The third evaluation is conducted on whitelist *Top200Edu*. *Top200Edu* is similar to *Edu* in terms of what type of categories are removed from the original list, with one big difference. *Top200Edu* only keeps the 200 most popular categories, which means the educational categories which are not popular enough, are not included. Using *Top200Edu* results in 21 746 examples being excluded from the index. Table 5.7 displays the results.

Keyword	Total hits	% of hits	Top 5	Top 10
Logic	338	5.31	1	0.9
Programming	864	13.58	0.8	0.9
Heuristic	31	0.49	1	0.5
Algebra	1034	16.25	1	1
Game Theory	2701	42.44	0.8	0.8
Fuzzy Logic	341	5.36	0.8	0.5
Java	267	4.20	1	1
Bayes Network	315	4.96	1	0.9
Derivation	62	0.97	1	0.9
Chain Rule	515	8.09	1	0.6
Prisoner's Dilemma	38	0.60	0.8	0.5
Nash Equilibrium	100	1.57	1	1
Cartesian Product	748	11.75	0.6	0.6
Parrondo's Paradox	46	0.72	0.4	0.4
Zero Sum	1842	28.94	0	0

**Table 5.7:** The precision of the queries when whitelist *Top200Edu* is used to filter the collection beforehand

---

In the fourth step, we evaluate whitelist *MathTech*. *MathTech* is based on *Top200Edu*, but it narrows the domains of the categories deemed relevant to only concern Technology and Mathematics. *MathTech* is a bit smaller than *Top200Edu*, with 157 different categories. 23 073 examples are filtered out, when *MathTech* is used. Table 5.8 reflects the results of the evaluation.

Keyword	Total hits	% of hits	Top 5	Top 10
Logic	326	6.47	1	0.9
Programming	852	16.91	0.8	0.9
Heuristic	19	0.38	0.8	0.6
Algebra	991	19.67	1	1
Game Theory	2386	47.37	0.8	0.9
Fuzzy Logic	329	6.53	0.8	0.5
Java	265	5.26	1	1
Bayes Network	262	5.20	1	0.9
Derivation	56	1.11	1	1
Chain Rule	402	7.98	1	0.6
Prisoner's Dilemma	33	0.66	0.8	0.5
Nash Equilibrium	87	1.73	1	1
Cartesian Product	606	12.03	0.6	0.6
Parrondo's Paradox	37	0.73	0.4	0.4
Zero Sum	917	18.21	0	0

**Table 5.8:** The precision of the queries when whitelist *MathTech* is used to filter the collection beforehand

---

In the final step we evaluate the most conservative whitelist, *MathTechWiki*. *MathTechWiki* is based on Wikipedia’s category hierarchy, where Mathematics, Logic, Mathematical Sciences, Computing and their sub categories have been included. Applying *MathTechWiki* leads to the removal of 27 107 examples. Table 5.9 contains the results of the final evaluation for Experiment II.

Keyword	Total hits	% of hits	Top 5	Top 10
Logic	100	9.97	1	1
Programming	107	10.67	0.2	0.3
Heuristic	20	1.99	0.6	0.4
Algebra	165	16.45	0.8	0.6
Game Theory	476	47.46	0.8	0.8
Fuzzy Logic	101	10.07	1	0.6
Java	9	0.90	0.4	3/9
Bayes Network	102	10.17	0.8	0.6
Derivation	22	2.19	0.6	0.4
Chain Rule	120	11.96	0.2	0.1
Prisoner’s Dilemma	8	0.80	0.4	2/8
Nash Equilibrium	17	1.69	1	0.9
Cartesian Product	126	12.56	0	0.1
Parrondo’s Paradox	18	1.79	0	0
Zero Sum	169	16.85	0.2	0.1

**Table 5.9:** The precision of the queries when whitelist *MathTechWiki* is used to filter the collection beforehand



---

### 5.5.3 Experiment III: Evaluation by F-score

To further explore the systems ability of retrieving relevant examples, we will use F-score to measure the system's performance. F-score is a weighted average between precision and recall, therefore both has to be measured to calculate the F-score.

While precision measures the quality of the retrieved examples, recall measures the quantity of relevant examples returned. Recall is calculated as follows:

$$\frac{|R \cap E|}{|R|}$$

In the equation above  $R$  is the set of all relevant examples and  $E$  is the set of all the retrieved examples. If no relevant examples are returned the recall is 0, and if all the relevant examples are retrieved the recall is 1. To properly measure recall all the examples have been manually inspected beforehand, to decide which examples are relevant or not relevant for a search with a specific keyword. When a measure for both precision and recall has been acquired, F-score can be calculated. We will calculate the F-score in the traditional way, which is also known as balanced F-score or  $F_1$  score. For F-score the best value is 1 and the worst is 0, it is calculated in the following way:

$$F_1 = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

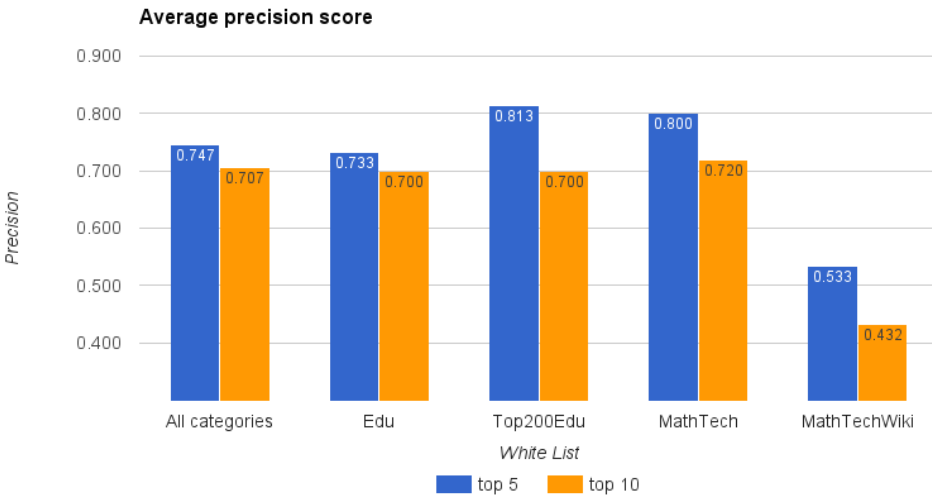
Before performing the experiment, a selection of 66 examples was chosen from the database based on three chosen keywords. Then, each example was evaluated regarding its relevance to the keywords. The keywords chosen were *java*, *algebra* and *nash equilibrium*. Finally, the keywords were used for three individual searches and the results were recorded. Table 5.10 shows the result of the experiment. The column *Total hits* contains the number of all examples retrieved by that search. *Relevant hits* contains how many relevant examples the search returned, while *Total relevant* has the number of how many relevant examples there exist in the entire example set for that specific keyword. Based on the numbers in the first three columns the F-score is calculated and displayed in the last column.

Keyword	Total hits	Relevant hits	Total relevant	F-score
Algebra	26	20	20	0.870
Java	24	18	18	0.857
Nash Equilibrium	16	15	15	0.968

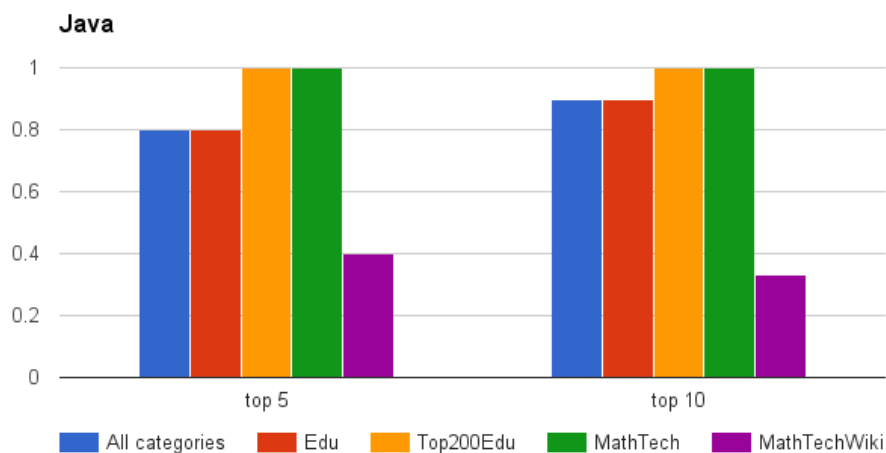
**Table 5.10:** Results of how well the system retrieves example from a manually inspected set.

### 5.5.4 Discussion of results

Figure 5.5 shows the average precision for each whitelist. It gives an overview of how the different whitelists performed during Experiment II, making a comparison between them easy. While the score for top 10 has a very low deviation, *Top200Edu* and *MathTech* separates themselves from the rest for the top 5 measure. This indicates that keeping the most popular 200 categories can work well as a general strategy. The rest of this section will discuss different observations made during and after the experiment. Charts showing the results of different keywords have been added and is referenced to in the discussion, see appendix D for charts over all keywords in Experiment II.



**Figure 5.5:** Average precision for all whitelists



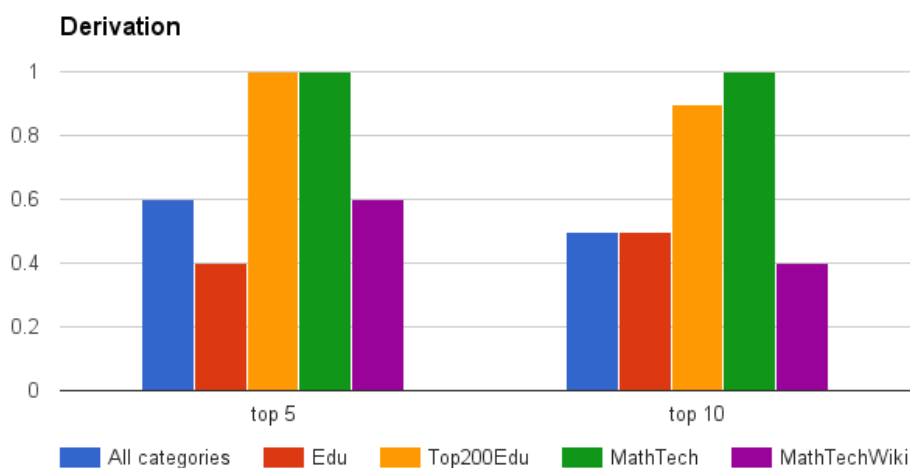
**Figure 5.6:** Precision score when the keyword *Java* is used as a search phrase, with the different whitelists applied.

### Few returned documents

In table 5.9 we note that many of the keywords have very few total hits. In particular, the keywords *java* and *prisoner's dilemma* returns less than 10 results, making the top 10 test inaccurate. Having so few total hits is reflected in the precision also dropping quite low. In figure 5.5 we can see that the whitelist *MathTechWiki* separates itself from the rest of the lists with almost 50 percent worse precision. This big loss of precision indicates that the whitelist excludes too many relevant examples from the index's corpus. Figure 5.6 demonstrates how the keyword *java* overall has extremely good results, but when *MathTechWiki* is applied, the score drops significantly lower.

### Excluding irrelevant examples

Whitelists were used to exclude irrelevant examples by focusing on a particular domain. There is though a golden middle ground of how inclusive the whitelist should be. The whitelists *Top200Edu* and *MathTech* seems to have more or less the right amount of inclusiveness. In Figure 5.5, they



**Figure 5.7:** Precision score when the keyword *Derivation* is used as a search phrase, with the different whitelists applied.

have the best average score for the top 5 results, and they are equal with *Edu* regarding top 10. When the keyword *derivation* was evaluated, *Top200Edu* and *MathTech* accomplished a perfect score, except for one irrelevant result among top 10 for *Top200Edu*. Figure 5.7 shows how the score is significantly better than when the other lists were evaluated. The results returned explains the big difference. A lot of results related to spoken languages and the word *derive* lead too many irrelevant examples. Since *Top200Edu* and *MathTech* are much stricter whitelists, those irrelevant examples was not a part of the index's corpus.

### Multiple words in search phrase

While discussing the results from keyword *zero sum*, poor handling of the keyword, was a part of the conclusion. The poor handling is experienced by highly unrelated examples being returned after a search. *Zero sum* demonstrates the most extreme side of the issue, but similar behaviour exists for other keywords also. The problem occurs when the keyword uses two words because Elasticsearch splits the keyword into two search terms. Elasticsearch will not prioritize or reward examples which contains those

---

two words directly next to each other. The rarity of the words also has a big part to play. For instance *nash equilibrium* also contains two words, but both are pretty rare. "Nash" is never used without being accompanied by "equilibrium". Although "equilibrium" exist in some examples which are not related to Nash Equilibrium, they are very few. Some keywords have one rare word and one common, for instance *prisoner's dilemma* and *cartesian product*. These keywords achieve a mediocre result, with an average precision around 0.6. When considering the effects rarity of the word causes for keywords containing two words, it makes sense that *zero sum*, which has two common words, have such a low precision.

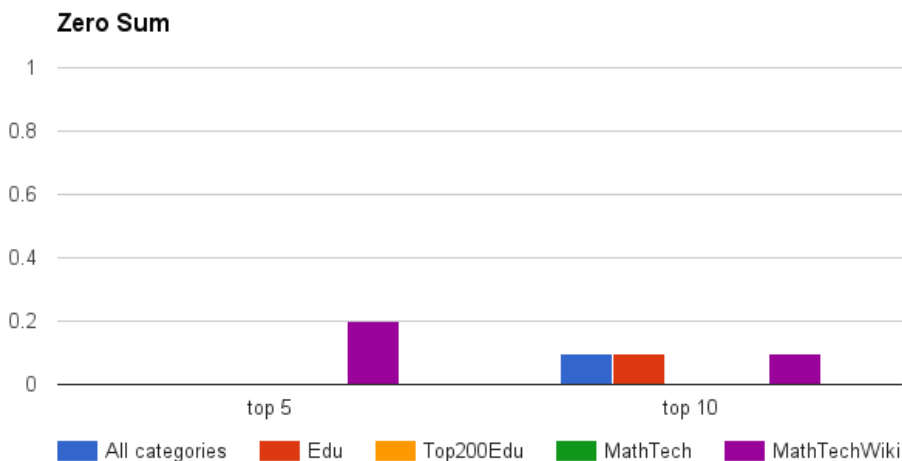
Order and proximity of the words are not a new problem in the field of information retrieval. Boosting the score of results based on how close the words appear, and if in right order, is a common technique. It could be done for this system, but it might end up being a to complicated solution, for an pretty straightforward problem. A simpler solution is to surround the words with quotation marks. The quotation marks will tell the index to handle all words between the quotation marks as a single search term. Since users often will search after name of examples or subjects, it will be enough to only accept examples where the terms are directly next to each other when the quotation mark surrounds the keyword.

### **Lack of relevant results**

Figure 5.8 displays the results when all the different whitelists have been applied when the keyword *zero sum* has been used as search phrase. All the whitelists have very low precision, either zero results or only one among the top ten. Although the results are inadequate, they do contradict the claim in section 5.5.1's error analysis, which suspected that no Zero Sum examples existed in the collection. Section 5.5.1 drew that conclusion because the main article about Zero sum was not included in the collection. When only *MathTechWiki* were evaluated, one relevant example was returned. This leads us to believe the abnormality was caused by using a bad combination of whitelists, making the other example not appearing among the top ten. The more thorough evaluations in experiment II, also points to the query not handling the keyword as a search phrase very well, because of the two common words used to form the search phrase. In conclusion,

---

the combination of faulty whitelist union and poor handling of the search phrase, are the reasons for the *Zero sum*'s complete lack of relevant results in Experiment I.



**Figure 5.8:** Precision score when the keyword *Zero Sum* is used as a search phrase, with the different whitelists applied.

### Union of whitelists

In Experiment II, the union of two whitelists were used, *Top200Edu* and *MathTechWiki*. This union gave the average score of 0.8, none of the whitelists in Experiment II matched this score, *MathTech* was closest with 0.76. Although this whitelist union gave better results than any independent whitelist, it showed weaknesses for other cases, in particular the case of no results for *zero sum*, and *algebra* achieving a perfect precision for *MathTech*. With that in mind, a new union of whitelists should be formed, that also could handle edge cases, either by finding a new combination or creating new whitelists.

---

## Recall of 1

In Experiment III, all three keywords achieved a recall of 1, the highest possible score for recall, which means that all relevant examples for each specific keyword was returned. Based purely on these results, the system seems to handle recall well. Although the measure of recall achieves a score of 1, it does not mean it is handled optimally. In any collection of documents, it is possible to achieve a recall of 1 by simply returning all documents in the collection, but as a consequence the precision will be significantly lower. The precision will be lower because of the trade-off relationship between precision and recall, increasing one will often lead to decreasing the other. The column in table 5.10 *Total hits* shows that returning all documents in the database is not the case for our system. However, it is still possible that the recall of 1 in our system lowers the precision.

A lower precision than necessary can be a problem for our system since it acts similar to a web search. In a web search users search by keywords and rarely look at results beyond the first page, hence precision becomes very important. The earlier experiments revealed that our system's precision is satisfactory, but by making the search even stricter, the precision might increase even more with an acceptable loss of recall.

## F-score

Although the recall achieved a perfect score in Experiment III, the F-score did not. As already discussed, this is because good recall often sacrifices good precision and vice versa. F-score represents the harmonic mean between these two, as both equally influences the F-score. Consequently, F-score can tell us something about the efficiency of our system. With all three scores above 0.85, the system performs well in terms of F-score.

Since the examples included in the subset for Experiment III, was partially chosen based on the three keywords, and not totally random, some bias may exist in the set. If the experiment had been scaled from the 66 examples to all 28 110, the score would most likely decrease. Using all examples would be more accurate, but classifying all examples were not possible during this project. Another way of altering the F-score is to add a

---

weight that is multiplied with the precision. The value of this weight would decide how much precision influences the final score. If we for instance assumed that our system should prioritize precision above recall, this version of F-score would be beneficial. Since we have no clear notion of what is most important, the experiment found the equally weighted version best.

## 5.6 Querying by examples

### 5.6.1 Experiment IV: Finding related examples

When querying by examples, relevant examples that assist in learning of the subject, are expected to be returned. To evaluate how the system accomplishes this, we will make use of observations from section 5.5. The observations will be used to create an optimal environment, by choosing the best performing whitelist and a keyword with perfect precision. The whitelist *MathTech* performed best for the top ten results with an average precision of 0.72, and will therefore be the whitelist chosen. As keyword, *Java* will be used for mainly three reasons. First, it is one of the keywords with a perfect score when *MathTech* was applied as whitelist. Second, *java* is a search phrase, but at the same time not the exact name of a category. Finally, it is very easy to consistently judge whether an example is about java or not, since it often contains java code. For an example to be deemed relevant, it will be satisfactory for the example to be an example within the Java domain.

There are three numbers that will be considered for the assessment of the system's performance in this experiment. First, how many examples are shown together with the main example. Second, how many of the examples are relevant. In addition, the number of the main example's categories will be included, to explore a possible correlation. The first ten examples will be used in the experiment. The results will be divided between the right and the left list.



---

Example	Total matches		Partial matches		Categories
	Total	Relevant	Total	Relevant	
1	2	2	10	10	3
2	2	2	10	10	3
3	2	2	10	10	6
4	2	2	10	10	3
5	4	4	10	10	5
6	4	4	10	10	5
7	0	0	10	10	3
8	27	27	10	10	6
9	2	2	10	10	6
10	3	3	10	10	2

**Table 5.11:** Statistics when querying by examples with *java* as keyword

The first column of table 5.11 represents an example, with the number being its order among the returned examples after the search. The next column shows total number of examples and how many of them that are relevant, both for the lists at the left and the right side. The last column shows how many categories the example have.

A quick look at the results presented in table 5.11, reveals consistently good results. To verify whether the performance of the system is as good as the results tell or if Java is a special case, a second evaluation will be performed. The keyword *nash equilibrium* is chosen, based on the same reason as why *java* was chosen. There is one important difference that might make an impact. Java is more general and therefore has 265 total hits, meanwhile the more specific *nash equilibrium* has 87 total hits. In the second evaluation, all examples discussing or referencing Nash Equilibrium will be deemed relevant.

---

Example	Total matches		Partial matches		Categories
	Total	Relevant	Total	Relevant	
1	6	6	9	3	4
2	6	6	9	3	4
3	6	6	9	3	4
4	6	6	9	3	4
5	72	29	0	0	1
6	72	29	0	0	1
7	72	29	0	0	1
8	6	6	9	3	4
9	0	0	10	5	3
10	6	6	9	3	4

**Table 5.12:** Statistics when querying by examples with *nash equilibrium* as keyword

## 5.6.2 Discussion of results

After conducting the second evaluation, a big variation in the quality of results occurred compared to the first evaluation. Despite the numbers differing, one pattern emerged in both evaluations. In table 5.11 example 1, 2 and 4 have the pattern, and 1, 2, 3, 4, 8 and 10 in 5.12. For these examples, the numbers are completely identical. They are identical because they come from the same article. Many Wikipedia articles have several example sections, which are used to explain the subject. Several examples originating from the same article not only results in them all achieving very similar score after a keyword search, but they will also have exactly the same categories.

A similar pattern emerges between example 5, 6 and 7 in table 5.12. The difference in this case is that the examples all originate from different articles. The reason they still achieve the same results is because they all have the same set of categories in common, since the algorithm used to find and measure related examples solely uses categories. *Game theory* is the only category in this case. Since its only **one** category, it is also causing the big difference between the left and the right list. All the examples achieving a perfect match with the selected example, only need to have Game

---

Theory as a category. This leads to related examples either having a total match or none match at all, with few of the examples having a total match being related. A solution could either be connecting more categories to the examples or create a better version of the matching algorithm.

There is some positive patterns that can be noticed to. For instance, all the related examples returned for the keyword *java* is also relevant. One of the things that causes this is already explained, being many examples originating from the same article. Another factor causing the high degree of relevance is the observation that many of the examples is included in a considerable amount of the lists. A big portion of the examples were also a part of the initial search results. These remarks point towards the system showing a good tendency regarding finding related examples.

---

# Conclusion and Future Work

## 6.1 Conclusion

The aim of this project has been to augment the use of examples for learning, by making use of educational technology. A system parsing Wikipedia articles for the extraction of sections containing examples has been created. Four research goals were established in chapter 1 in order to manage the projects workflow into desired results. In chapter 2, other peoples work regarding text data mining, semi-structured text and Wikimedia was examined, to help discover the usefulness and possibilities of this project. The concept of a pipeline turning raw source data from Wikipedia into an index containing examples, were explained in chapter 3. In addition, how to search the index was also expressed. To optimize how the system handles the collecting and serving of examples, an analysis of example's structure and content were performed as well. Chapter 4 explained in detail how the defined concept were implemented into a working system. Finally chapter 5 examined the accomplishment of the research goals. Research goal 1, 2 and 3 were summed up and concluded, while four experiments were conducted to evaluate the fourth research goal. The first experiment evaluated the precision for several keywords when used as a search phrase. The second experiment evaluated the precision of the results when the four

---

different whitelists were applied one at a time. Experiment III evaluated the system's F-score on with three keywords used on a small subset of the whole collection. Finally, the fourth experiment evaluated how well the system finds related examples when a specific example is selected.

Through Experiment I we showed that the system are able to find relevant examples with the original implementation of applying the whitelists *Top200Edu* and *MathTechWiki* to filter examples before being added to the index. A set of keywords with different degree of generality and from different domains gave an average precision of 0.8. This is a satisfying number for the system's first implementation. The experiment did reveal that the whitelists applied influences the system to a significant degree. Consequently, Experiment II were conducted to discover how the whitelists influenced the system, and to find the best whitelist. The results of Experiment II taught us that the system is affected by many different aspects which determines its performance. Format of the search phrase and amount of relevant examples in the whole collection revealed patters that had a negative impact on the precision. All things considered, the two whitelists *Top200Edu* and *MathTech*, gave better results than the others. They had both very similar score, which reflects that the two lists also are very similar, with *MathTech* having excluded 43 categories compared to *Top200Edu*. Although these whitelists performed best, the first implemented union of *Top200Edu* and *MathTechWiki* scored better. The experiments revealed some weaknesses regarding the union of these whitelists though. For instance lacking relevant results for some keywords, where the independent whitelists managed better. Therefore different unions of whitelists should be combined and evaluated in attempt of finding one that patches the weaknesses found in Experiment I and II.

In Experiment III, the system's F-score was measured with a small subset of manually inspected examples from the database. All possible relevant documents in the subset were returned for each keyword used, which entails a recall of 1. With also a fairly high precision, the system performed well when F-score was measured for all three keywords. Although a recall of 1 seems good, it most likely causes a lower precision for the system. Sacrificing some recall for better precision could give the system a more optimal performance. If precision also is assumed to be more important than recall, weights could be added to the F-score equation to enhance precisions

---

importance. It is important to note that the subset most likely reflects the entire set with a degree of bias, therefore an experiment which covers more of the complete collection of examples would be more accurate.

Experiment IV evaluated how well the system found and rated related examples based on an example selected by a user. The query combines information from the initial search and the categories of the example, which the user selected. The experiment revealed that the set of categories for the selected example greatly influenced the results. Since many of the examples had not a lacking category list, the results ended up being unstable. The algorithm's rating of categories worked well when the example had many categories, but that is not always the case.

## 6.2 Future Work

There are several aspects of this project which would benefit from an extended amount of work or research

Firstly the system itself, can be greatly improved by being able to deal with examples from different sources. The foundation for doing so already exists in the system, since the idea of extracting examples from different sources have existed since the beginning. For simplicity and saving time, this project has focused on only using Wikipedia as source, and therefore is excessively tailored for Wikipedia articles. But since the different processes in the pipeline is extremely independent, replacing them to accommodate other sources should be a trivial task. Accommodating more sources would result in a richer database of examples, which in turn would help the end user.

Another aspect that could benefit the system is a deeper knowledge of examples, and how they relate to each other. A better understanding could improve both the rating of their relevance score when searched for and displaying related examples. Improvement on finding related examples, can give a natural learning progress when browsing from one example and to its related ones.

---

The search itself could also be more optimized. Since Elasticsearch was chosen to manage the database of examples, the search API served by the Elasticsearch process could be explored further. Elasticsearch offers a great amount of different customizations that can be applied to the queries used for the search, which would make the search more complex, but also could improve the search results. Experiment IV indicated that the system has a very high recall. Making use of methods available in Elasticsearch's search API, a more optimal trade-off between precision and recall could be achieved.

Improvements can also be made regarding querying for examples based on a user selected example. The algorithm rating the results rely too much on categories, when the current collection of examples does not facilitate use of categories enough. One approach is to alter the algorithm, for instance making use of references in the articles has been explored during the project, although it was not successful enough to make it in the final implementation. Another approach could be to change our view of categories. It could be more beneficial to look at them as tags instead. The current categories could be converted to tags, and a system for manually tagging examples could be implemented. This way, tags would work similar as they do for a YouTube video, helping both the search and creating a set of related videos. A combination of the two mentioned approaches, should improve the retrieval of related examples to a selected example.

Finally usability testing should be conducted. The system is created to help the user, therefore it is very important to make sure the user interaction is optimal. Both the design of the user interface and the user interaction should be tested, and the feedback should be used to learn more about the system's purpose and further improve it. By analyzing user's interaction with the system, we can also get feedback that will help optimize the search. Especially in regards with the level of recall and precision. For instance if users rarely explores the lower ranked results, increasing precision is a good idea. On the other hand, if the users explores many of the returned results, good recall can be more beneficial.



# Bibliography

- [1] J. P. McCarthy and L. Anderson, “Active learning techniques versus traditional teaching styles: Two experiments from history and political science,” *Innovative Higher Education*, vol. 24, no. 4, pp. 279–294, 2000.
- [2] J. P. Lalley and R. H. Miller, “The learning pyramid: Does it point teachers in the right direction?” *Education*, vol. 128, no. 1, p. 64, 2007.
- [3] R. Robinson, M. Molenda, and L. Rezabek, “Facilitating learning,” *Educational technology: A definition with commentary*, pp. 15–48, 2008.
- [4] S. Soderland, “Learning information extraction rules for semi-structured and free text,” *Machine learning*, vol. 34, no. 1-3, pp. 233–272, 1999.
- [5] W. Foundation, *Wikimedia foundation*, <https://wikimediafoundation.org/wiki/Home>, [Online; accessed 17-March-2016], 2016.
- [6] J. Wales and L. Sanger, *Wikipedia, the free encyclopedia*, <https://www.wikipedia.org>, [Online; accessed 17-March-2016], 2016.
- [7] D. J. Barrett, *MediaWiki*. O’Reilly, 2008.
- [8] A. Montero-Asenjo and C. A. Iglesias, *Turning wikipedia into a resource for language research*, Grupo de Sistemas Inteligentes. Universidad Politécnica de Madrid, 2008.

- 
- [9] E. Gabrilovich and S. Markovitch, "Overcoming the brittleness bottleneck using wikipedia: Enhancing text categorization with encyclopedic knowledge," in *AAAI*, vol. 6, 2006, pp. 1301–1306.
- [10] I. Witten and D. Milne, "An effective, low-cost measure of semantic relatedness obtained from wikipedia links," in *Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy*, AAAI Press, Chicago, USA, 2008, pp. 25–30.
- [11] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley Longman Publishing Company, 1999.
- [12] M. A. Hearst, "Untangling text data mining," in *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, Association for Computational Linguistics, 1999, pp. 3–10.
- [13] R. Schenkel, F. M. Suchanek, and G. Kasneci, "Yawn: A semantically annotated wikipedia xml corpus.," in *BTW*, Citeseer, vol. 103, 2007, pp. 277–291.
- [14] R. Cilibrasi and P. Vitanyi, "The Google Similarity Distance," *IEEE Transactions on Knowledge and Data Engineering* 19, no. 3, pp. 370–383, 2007.
- [15] E. Foundation, *SMILA - Unified Information Access Architecture*, <http://www.eclipse.org/smila>, [Online; accessed 22-March-2016], 2015.
- [16] P. Avgeriou and U. Zdun, "Architectural patterns revisited—a pattern," *Proc. 10th European Conf. Pattern Languages of Programs (EuroPLOP)*, UVK Konstanz, 2005.
- [17] N. S. A. Hennig E. Marongiu and R. J. Whitrow, "DART—a software architecture for the creation of a distributed asynchronous recognition toolbox," in *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, IEEE, vol. 1, 1997, pp. 439–443.
- [18] S. Tilkov and S. Vinoski, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, p. 80, 2010.

# Appendices



Appendix **A**

## Database statistics

---

Category	Count
Articles created via the Article Wizard	86
Articles containing video clips	85
Articles containing proofs	76
Ring theory	74
Protein domains	73
Linear algebra	65
Formal languages	57
Algebraic geometry	53
Living people	51
Naval guns of the United Kingdom	51
Commutative algebra	50
Game theory	47
Algebraic structures	47
Topology	45
Stochastic processes	44
Marketing	44
Functional analysis	42
Group theory	42
Coastal artillery	42
Combinatorics	41
Abstract algebra	38
Functional languages	38
Weather warnings and advisories	36
General topology	36
Management	36
Articles with example Java code	36
Victorian-era weapons of the United Kingdom	36
Signal processing	35
Category theory	35
Number theory	35
Social psychology	34
Architectural styles	34
Concepts in physics	34
1067 mm gauge locomotives of Japan	34
Algebra	33

**Table A.1:** Statistics over most popular categories

Appendix **B**

Example samples

---

## Examples and exercises on Pareto efficiency

### Example

Consider an economy that contains only one good, which everyone likes. Then **every** allocation is Pareto efficient: the only way to make someone better off is to give them more of the good, in which case someone else will have less of the good, and hence be worse off.

### Example

An economy contains two people and two goods, apples and bananas. Person 1 likes apples and dislikes bananas (the more bananas she has, the worse off she is), and person 2 likes bananas and dislikes apples. There are 100 apples and 100 bananas available.

The only allocation that is Pareto efficient is that in which person 1 has all the apples and person 2 has all the bananas. For any other allocation, one of the persons has some units of the good she does not like, and would be better off if the other person had those units.

### Example

An economy contains two people and two goods, apples and bananas. Person 1 likes apples and doesn't care one way or the other about bananas (she is indifferent between any bundles  $(a,b)$  and  $(a,b')$ , where  $a$  is some number of apples and  $b$  and  $b'$  are numbers of bananas). Person 2 likes bananas and doesn't care one way or the other about apples. There are 100 apples and 100 bananas available.

The only allocation that is Pareto efficient is that in which person 1 has all the apples and person 2 has all the bananas. For any other allocation, one of the persons has some units of the good about which she doesn't care; transferring those units to the other person would have no effect on her and would make the other person better off.

### Example

An economy contains two people and two goods, apples and bananas. Both people like both goods, but value them differently. For person 1, 1 apple is exactly equivalent to 2 bananas: she is indifferent between any bundles  $(a, b)$  and  $(a - n, b + 2n)$ , where  $a$  is some number of apples,  $b$  is some number of bananas, and  $n$  is some number. For person 2, 2 apples are exactly equivalent to 1 banana.

An allocation is Pareto efficient if and only if

- either person 1 has no bananas
- or person 2 has no apples.

Why? Suppose person 1 has some bananas and person 2 has some apples. Then by transferring one banana from person 1 to person 2 and one apple from person 2 to person 1 we make both of them better off. On the other hand, if person 1 has no bananas then any trade that makes her better off must involve her getting at least twice as many bananas as she gives up in apples, which results in person 2 being worse off. Similarly, if person 2 has no apples then any trade that makes her better off must involve her getting at least twice as many apples as she gives up in bananas, which results in person 1 being worse off.



Three of the allocations that are Pareto efficient are those in which

- person 1 has all the apples and person 2 has all the bananas
- person 1 has all the apples and all the bananas
- person 2 has all the apples and all the bananas.

---

[Copyright](#) © 1997 by [Martin J. Osborne](#)

---

## Robinson Crusoe example

Yossi Spiegel

Consider an island economy with one agent (Robinson Crusoe) who has an endowment of  $\bar{L}$  units of an input that can be used to produce 2 final goods,  $x$  and  $y$ . Although this example is probably not very interesting (presumably Robinson Crusoe can take care of himself and does not need to rely on a market mechanism, or any mechanism for that matter, to determine how much  $x$  and  $y$  to consume), it is nonetheless the simplest example for a production economy we can imagine that is still non-trivial. For instance, if  $\bar{L}$  could have been used to produce only one final good, say  $x$ , instead of both  $x$  and  $y$ , then Robinson would have simply converted all of  $\bar{L}$  into  $x$  and there would be no problem to analyze. Yet, with two final goods that can be produced, we can ask how many units of  $x$  and how many units of  $y$  will be produced and consumed by Robinson.

To make the problem more concrete, suppose that the production functions for  $x$  and  $y$  are given by

$$x = \sqrt{L_x}, \quad y = \frac{\sqrt{L_y}}{2}, \quad (1)$$

where  $L_x$  and  $L_y$  are the quantities of  $L$  used in the production of  $x$  and  $y$ , respectively. In addition suppose that Robinson's utility function is given by

$$U(x,y) = \sqrt{x} \sqrt{y}. \quad (2)$$

Given the description of the economy we will now characterize the set of Pareto efficient allocations and compute the Walrasian equilibrium and show that the two coincide.

### Pareto efficiency

First we need to determine the Production Possibilities Frontier (PPF). In other words, find all combinations of  $x$  and  $y$  that can be produced efficiently. That is, all combinations of  $x$  and  $y$

such that there is no way to get more of both  $x$  and  $y$  by reallocating  $\bar{L}$ . To derive the PPF, note that since Robinson does not want to consume  $L$ , he will use all of  $\bar{L}$  in the production of either  $x$  or  $y$ . Hence, it must be the case that

$$L_x + L_y = \bar{L}. \quad (3)$$

Using this expression, we can express the production functions for  $x$  and  $y$  as follows:

$$x = \sqrt{L_x}, \quad y = \frac{\sqrt{\bar{L} - L_x}}{2}, \quad (4)$$

or equivalently,

$$L_x = x^2, \quad L_x = \bar{L} - 4y^2. \quad (5)$$

Using the two expressions in equation (5) we get:

$$x^2 = \bar{L} - 4y^2, \Leftrightarrow T(x,y) \equiv x^2 + 4y^2 - \bar{L} = 0. \quad (6)$$

The equality  $T(x,y) = 0$  characterizes then the efficient combinations of  $x$  and  $y$  and therefore defines the PPF.

Having derived the PPF we are now ready to solve for the Pareto efficient allocations. Noting that the economy here contains only one individual, the set of Pareto efficient allocations is determined by the following maximization problem:

$$\begin{aligned} \underset{x,y}{\text{Max}} \quad & \sqrt{x}\sqrt{y} \\ \text{s.t.} \quad & T(x,y) = 0. \end{aligned} \quad (7)$$

The Lagrangian that corresponds to this problem is given by

$$\underset{x,y,\lambda}{\text{Max}} \quad \mathfrak{L}(x,y,\lambda) \equiv \sqrt{x}\sqrt{y} + \lambda (T(x,y) - 0). \quad (8)$$

The first order conditions for the problem are:

$$\frac{\partial \mathcal{L}(x, y, \lambda)}{\partial x} = \frac{\sqrt{y}}{2\sqrt{x}} + 2\lambda x = 0, \quad (9)$$

$$\frac{\partial \mathcal{L}(x, y, \lambda)}{\partial y} = \frac{\sqrt{x}}{2\sqrt{y}} + 8\lambda y = 0, \quad (10)$$

and

$$\frac{\partial \mathcal{L}(x, y, \lambda)}{\partial \lambda} = x^2 + 4y^2 - \bar{L} = 0. \quad (11)$$

Solving equations (9)-(11), the Pareto efficient allocation is given by:

$$x^* = \frac{\sqrt{\bar{L}}}{\sqrt{2}}, \quad y^* = \frac{\sqrt{\bar{L}}}{\sqrt{8}}. \quad (12)$$

### Walrasian equilibrium

To characterize the Walrasian equilibrium, suppose that Robinson establishes a firm, buys the input  $\bar{L}$  from himself for a price of  $w$  per unit, then as the owner of the firm, decides how many units of  $x$  and how many units of  $y$  to produce to maximize profits given the prices of the two goods,  $p_x$  and  $p_y$ , and then sells the firm's output to himself. Moreover, the prices  $w$ ,  $p_x$ ,  $p_y$ , are called by an auctioneer (question: if Robinson lives on the island all by himself, who is the auctioneer?) with Robinson responding by submitting his demands and supplies until all three markets, the market for  $x$ , the market for  $y$ , and the market for  $L$ , are cleared. This obviously sounds not only schizophrenic (Robinson deals with himself twice at arm-length: as a provider of the inputs and as a consumer of the final goods), but rather silly: why should Robinson rely

on an auctioneer to determine the market clearing prices? Yet again, this is an example of how the market mechanism works and how it leads to Pareto efficient allocations.

To characterize the Walrasian equilibrium, note that what we are looking for is a list,  $(p_x^*, p_y^*, w)$  such that the following conditions are met:

$$x^D(p_x^*, p_y^*, w^*) = x^S(p_x^*, p_y^*, w^*), \quad (13)$$

$$y^D(p_x^*, p_y^*, w^*) = y^S(p_x^*, p_y^*, w^*), \quad (14)$$

and

$$L_x(p_x^*, p_y^*, w^*) + L_y(p_x^*, p_y^*, w^*) = \bar{L}, \quad (15)$$

where  $x^D$  and  $y^D$  are Robinson's demands for goods  $x$  and  $y$  given the prices,  $p_x^*$ ,  $p_y^*$ , and  $w^*$ ;  $x^S$  and  $y^S$  are the firm's supplies of goods  $x$  and  $y$  given the prices,  $p_x^*$ ,  $p_y^*$ , and  $w^*$ ; and  $L_x$  and  $L_y$  are the quantities of input used in the production of  $x$  and  $y$ . Therefore, equation (13) is the market clearing condition for good  $x$ , equation (14) is the market clearing condition for good  $y$ , and equation (15) is the market clearing condition for the input.

To derive the  $L_x$  and  $L_y$ , note that the firm's profit, given the production functions of  $x$  and  $y$  is given by

$$\pi = p_x \sqrt{L_x} + p_y \frac{\sqrt{L_y}}{2} - w L_x - w L_y. \quad (16)$$

Hence, the firm's demands for input are defined by the following first order conditions:

$$\frac{\partial \pi}{\partial L_x} = \frac{p_x}{2\sqrt{L_x}} - w = 0. \quad (17)$$

and

$$\frac{\partial \pi}{\partial L_y} = \frac{p_y}{4\sqrt{L_y}} - w = 0. \quad (18)$$

Solving these two conditions reveals that

$$L_x(p_x, p_y, w) = \frac{p_x^2}{4w^2}, \quad L_y(p_x, p_y, w) = \frac{p_y^2}{16w^2}. \quad (19)$$

Given  $L_x$  and  $L_y$ , the supplies of goods x and y, respectively are:

$$x^S(p_x, p_y, w) = \sqrt{L_x(p_x, p_y, w)} = \frac{p_x}{2w}, \quad y^S(p_x, p_y, w) = \frac{\sqrt{L_y(p_x, p_y, w)}}{2} = \frac{p_y}{8w}. \quad (20)$$

Having solved for the firm's demands for inputs and supplies of final products, the firm's profits given the prices  $p_x$ ,  $p_y$ , and  $w$  are:

$$\pi = p_x \sqrt{L_x} + p_y \frac{\sqrt{L_y}}{2} - w L_x - w L_y = \frac{4p_x^2 + p_y^2}{16w}. \quad (21)$$

Since Robinson owns the firm, his income is equal to the firm's profits plus his income from selling  $\bar{L}$  units of input to the firm at a price of  $w$  per unit. Hence, Robinson's income is:

$$M = \pi + w\bar{L} = \frac{4p_x^2 + p_y^2}{16w} + w\bar{L}. \quad (22)$$

Now we are ready to characterize Robinson's demand for goods x and y. The demands for x and y are determined by the solution to the following problem:

$$\begin{aligned} \underset{x,y}{\text{Max}} \quad & \sqrt{x} \sqrt{y} \\ \text{s.t.} \quad & p_x x + p_y y = M, \end{aligned} \quad (23)$$

where  $M$  is defined in equation (22). The Lagrangian associated with this problem is

$$\text{Max}_{x,y,\lambda} \mathcal{L}(x,y,\lambda) \equiv \sqrt{x}\sqrt{y} + \lambda (M - p_x x - p_y y). \quad (24)$$

The first order conditions for the problem are:

$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial x} = \frac{\sqrt{y}}{2\sqrt{x}} - \lambda p_x = 0, \quad (25)$$

$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial y} = \frac{\sqrt{x}}{2\sqrt{y}} - \lambda p_y = 0, \quad (26)$$

and

$$\frac{\partial \mathcal{L}(x,y,\lambda)}{\partial \lambda} = M - p_x x - p_y y = 0. \quad (27)$$

Solving equations (25)-(27), the demands of Robinson are given by:

$$x^D(p_x, p_y, w) = \frac{M}{2p_x}, \quad y^D(p_x, p_y, w) = \frac{M}{2p_y}. \quad (28)$$

Since we already solved for the firm's demands for inputs in equation (19), the firm's supply of  $x$  and  $y$  in equation (20) and Robinson's demand for goods  $x$  and  $y$  in equation (28), we can determine the Walrasian equilibrium by substituting from equations (19), (20), and (28),

and recalling that Robinson's income is given by equation (22) into the equilibrium conditions in equations (13)-(15). This leads to the following 3 equations that must hold in equilibrium:

$$\frac{\frac{4p_x^{*2} + p_y^{*2}}{16w^*} + w^* \bar{L}}{2p_x^*} = \frac{p_x^*}{2w^*}, \quad (29)$$

$$\frac{\frac{4p_x^{*2} + p_y^{*2}}{16w^*} + w^* \bar{L}}{2p_y^*} = \frac{p_y^*}{8w^*}, \quad (30)$$

and

$$\frac{p_x^{*2}}{4w^{*2}} + \frac{p_y^{*2}}{16w^{*2}} = \bar{L}. \quad (31)$$

The Walrasian equilibrium,  $(p_x^*, p_y^*, w^*)$  is the solution to equations (30)-(32). However, instead of trying to solve the system of 3 equations directly, we can note that by Walras' law we can normalize one of the prices to 1. Since it does not matter which price we normalize to 1, let's pick  $w^* = 1$ . Now, we need to solve the system with  $w^* = 1$  and find  $p_x^*$  and  $p_y^*$ .

To this end, note that equations (29) and (30) can be rewritten as follows:

$$4p_x^{*2} + p_y^{*2} + 16\bar{L} = 16p_x^{*2}, \quad (32)$$

and

$$4p_x^{*2} + p_y^{*2} + 4\bar{L} = 4p_y^{*2}. \quad (33)$$



But since the left side of the two equations is the same, it follows that

$$16p_x^{*2} = 4p_y^{*2}, \Leftrightarrow 2p_x^* = p_y^*. \quad (34)$$

Substituting for  $2p_x^* = p_y^*$  and  $w^* = 1$  in equation (31), the equation becomes:

$$\frac{p_x^{*2}}{4} + \frac{4p_x^{*2}}{16} = \bar{L}, \Leftrightarrow p_x^* = \sqrt{2\bar{L}}. \quad (35)$$

Hence,

$$p_y^* = 2p_x^* = \sqrt{8\bar{L}}. \quad (36)$$

That is, the walrasian equilibrium is given by:

$$p_x^* = \sqrt{2\bar{L}}, \quad p_y^* = \sqrt{8\bar{L}}, \quad w^* = 1. \quad (37)$$

To verify that the resulting allocation is Pareto efficient, note that given the equilibrium prices, the quantities of  $x$  and  $y$  that Robinson will consume are

$$x^* = \frac{p_x}{2w^*} = \frac{\sqrt{\bar{L}}}{\sqrt{2}}, \quad y^* = \frac{p_y}{8w^*} = \frac{\sqrt{\bar{L}}}{\sqrt{8}}, \quad (38)$$

which is exactly the Pareto efficient allocation we found earlier.

The conclusion is that the Walrasian mechanism is a way to implement Pareto efficient allocations in a decentralized manner: Robinson does not need to see the "big" picture. When he acts as a manager of a firm he responds to the price of the input and the prices of the two goods and decides how much to produce. As a buyer he simply decides how much to buy given the prices and his income. Yet, despite the fact that he acts in two different roles, the final outcome is Pareto efficient exactly as if he were to think about the whole problem of finding the most beneficial allocation from his point of view.

---

## Example white lists

Category	Count
Articles containing proofs	76
Ring theory	74
Protein domains	73
Linear algebra	65
Formal languages	57
Algebraic geometry	53
Commutative algebra	50
Algebraic structures	47
Game theory	47
Topology	45
Marketing	44
Stochastic processes	44
Group theory	42
Functional analysis	42
Combinatorics	41
Abstract algebra	38
Functional languages	38
General topology	36
Articles with example Java code	36
Signal processing	35
Category theory	35

---

Number theory	35
Concepts in physics	34
Architectural styles	34
Social psychology	34
Algebra	33
Polynomials	33
Mathematical relations	33
XML-based standards	33
Object-oriented programming languages	33
Dynamical systems	32
Quantum mechanics	32
Functions and mappings	32
Statistical terminology	31
Decision theory	31
Genetics	30
Model theory	30
Ethology	30
Articles with example pseudocode	30
Procedural programming languages	30
Mathematical optimization	30
Probability theory	29
Matrices	29
Algebraic topology	29
Properties of topological spaces	29
Order theory	29
Cross-platform software	28
Estimation theory	27
Integer sequences	27
Protein families	27
Field theory	27
Semigroup theory	27
Financial risk	26
Fluid dynamics	26
Molecular biology	26
Measure theory	26
Evolutionary biology	26
Cryptography	26
Properties of groups	25

---

---

Combinatorics on words	25
Semantics	25
Statistical theory	25
Metric geometry	25
Cognitive biases	25
Java platform	25
Software design patterns	24
Lie algebras	24
Scripting languages	24
Quantum field theory	24
Differential geometry	24
Representation theory	24
Ecology	24
Economics terminology	24
Mathematical finance	23
Control theory	23
Mathematical logic	23
Statistical models	23
Theory of probability distributions	23
Data management	23
Binary operations	23
Computer file formats	23
Syntax	23
Technical communication	23
Software using the MIT license	22
Object-oriented programming	22
Numerical analysis	22
Measures (measure theory)	22
Software testing	22
Mathematical analysis	22
Diagrams	22
Finance	22
Graph families	22
Classical mechanics	22
Articles with example C++ code	22
World Wide Web Consortium standards	22
Architectural elements	22
Data analysis	21

---

---

Systems theory	21
Bioinformatics	21
Sociological terminology	21
Computational complexity theory	21
Homotopy theory	20
Logic	20
Module theory	20
Markup languages	20
Theory of computation	20
Political terminology	20
Partial differential equations	20
Hydrology	20
Matrix theory	20
Linguistics	20
Chess terminology	19
E-commerce	19
Parallel computing	19
Physical quantities	19
Educational psychology	19
Legal terms	19
Medical terminology	19
Geometric group theory	19
Artificial intelligence	19
Epidemiology	19
Algebraic number theory	19
Numerical linear algebra	19
Articles with example code	19
Rhetoric	19
Unix SUS2008 utilities	19
Factorial and binomial topics	19
Types of functions	19
Knowledge representation	19
Sociolinguistics	18
Articles with example C code	18
Digital signal processing	18
Application programming interfaces	18
Modular arithmetic	18
Source code	18

---

---

Graph algorithms	18
Route diagram templates	18
Mathematical physics	18
Graph theory	18
Equations	18
Deception	18
Geometry	18
Lossless compression algorithms	18
Numerical differential equations	17
Operator theory	17
Probability theorems	17
Mechanics	17
Fourier analysis	17
Design of experiments	17
Urban studies and planning	17
Algorithms	17
Mathematical terminology	17
Regression analysis	17
Encodings	17
Image processing	17
Symmetry	17
Homological algebra	17
Emerging technologies	17
Sustainability	17
Web application frameworks	17
Network protocols	17
Free compilers and interpreters	16
Theoretical physics	16
Free software	16
Financial terminology	16
Scientific modeling	16
Investment	16
Rhetorical techniques	16
Java (programming language) libraries	16
Permutations	16
Regular graphs	16
Windows administration	16
Quadratic forms	16

---

---

Information theory	16
Corporate finance	16
Dynamically typed programming languages	16
Free software programmed in Java (programming language)	16
Operations research	16
Economic problems	16
Figures of speech	16
Algebras	16
Computability theory	16
Internet protocols	16
Project management	16
Tensors	16
Neuroscience	16
Geomorphology	16
Probability distributions	16
Lie groups	15
Risk	15
Metadata	15
Free software programmed in C	15
Cross-platform free software	15
Programming constructs	15
Fractals	15
Coding theory	15
Statistical mechanics	15
Physical chemistry	15
Statistical ratios	15
Polyhedra	15

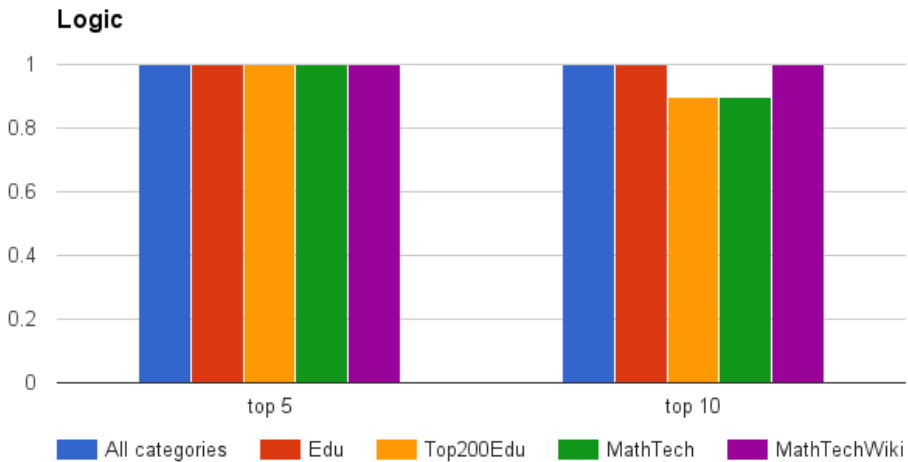
**Table C.1:** Removes categories not related to educational topics within science. For instance topics regarding history is removed. Includes only the top 200 most popular categories in the list.



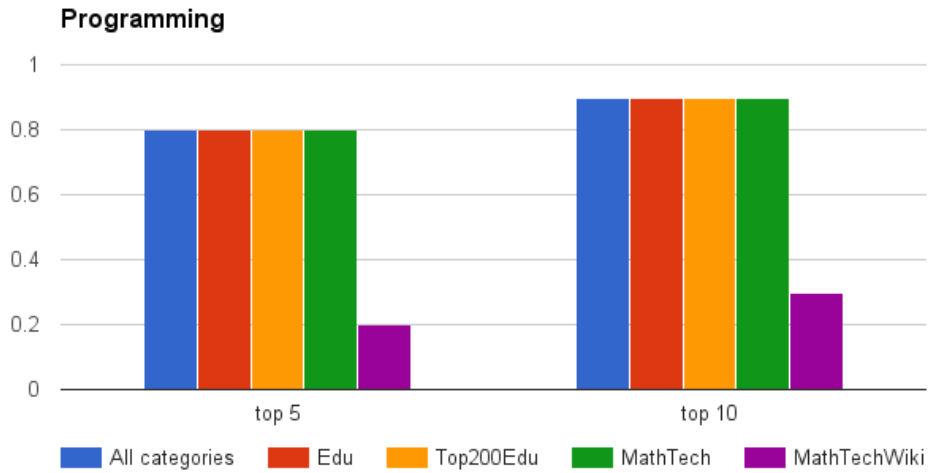
# Appendix D

## Charts from Experiment II

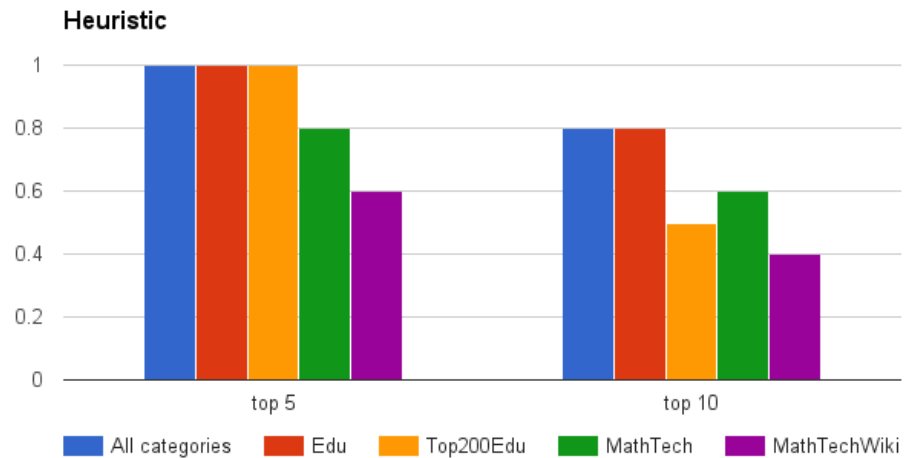
Below the charts for experiment II that were not discussed, is included.



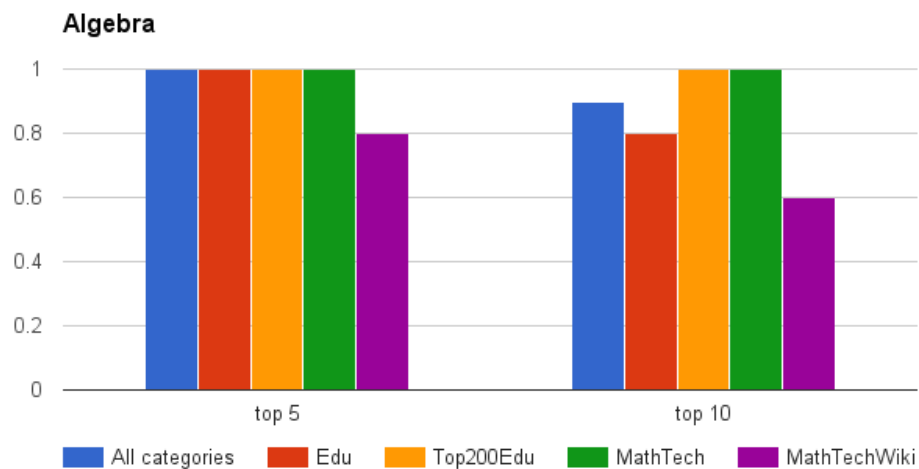
**Figure D.1:** Precision score when the keyword *Logic* is used as a search phrase, with the different white lists applied.



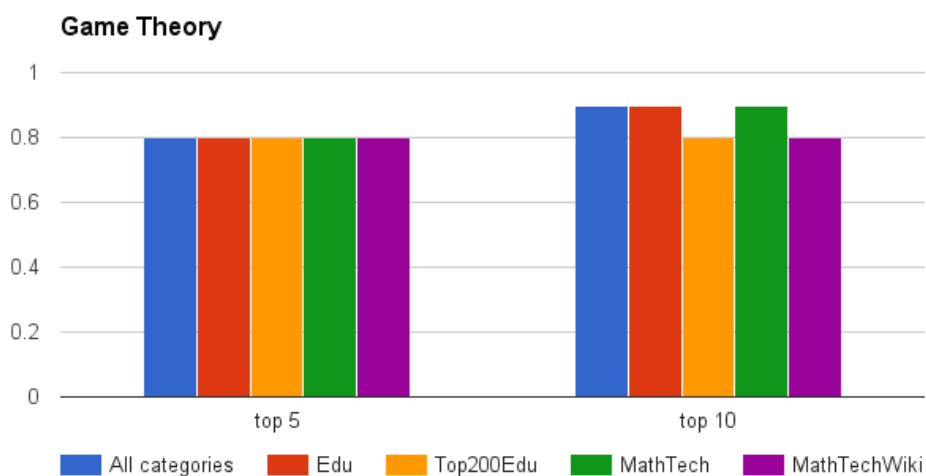
**Figure D.2:** Precision score when the keyword *Programming* is used as a search phrase, with the different white lists applied.



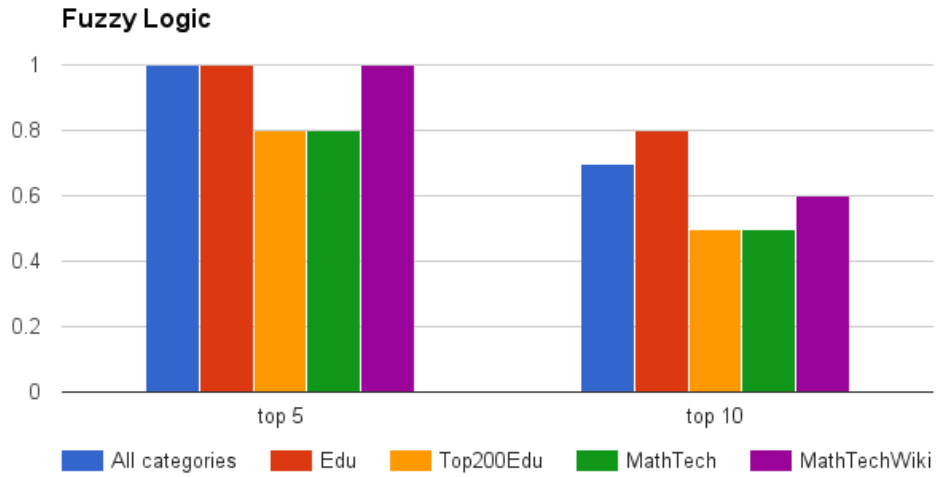
**Figure D.3:** Precision score when the keyword *Heuristic* is used as a search phrase, with the different white lists applied.



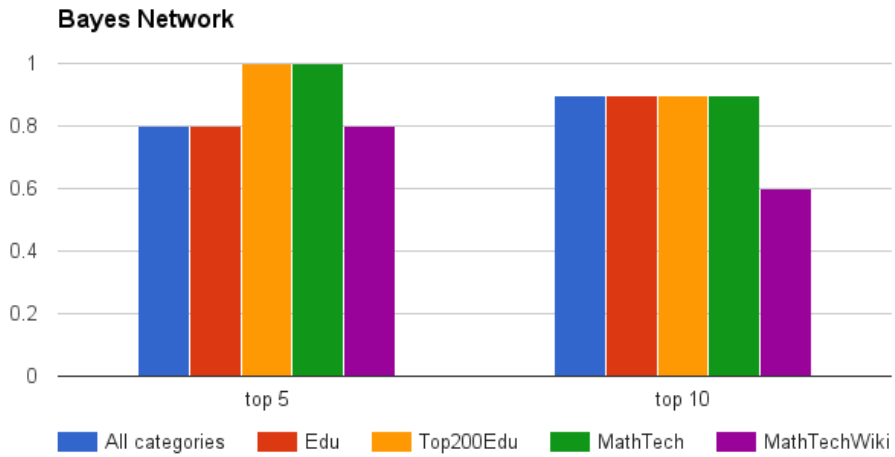
**Figure D.4:** Precision score when the keyword *Algebra* is used as a search phrase, with the different white lists applied.



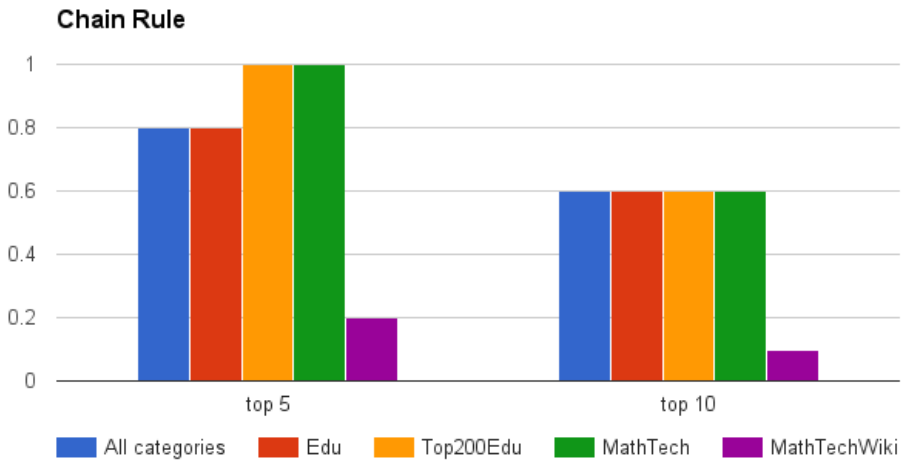
**Figure D.5:** Precision score when the keyword *Game Theory* is used as a search phrase, with the different white lists applied.



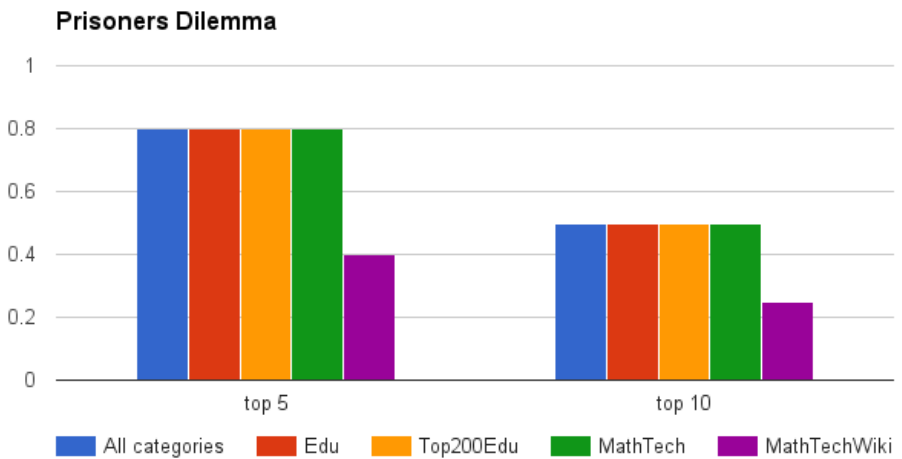
**Figure D.6:** Precision score when the keyword *Fuzzy Logic* is used as a search phrase, with the different white lists applied.



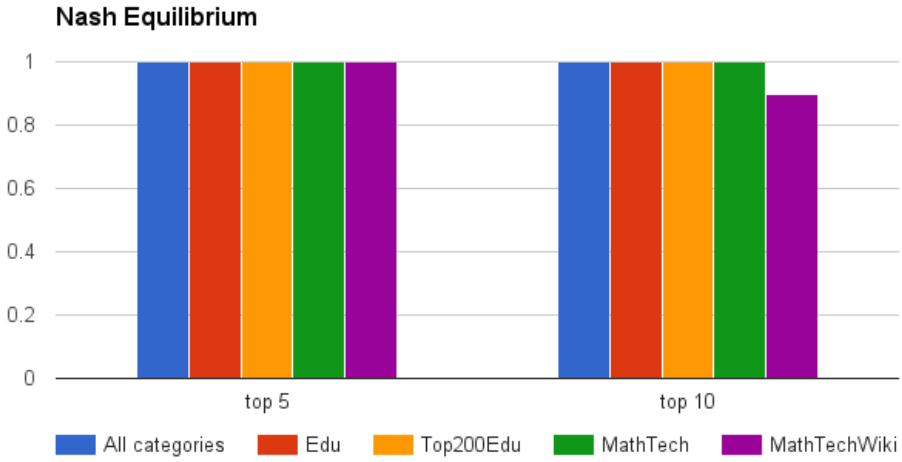
**Figure D.7:** Precision score when the keyword *Bayes Network* is used as a search phrase, with the different white lists applied.



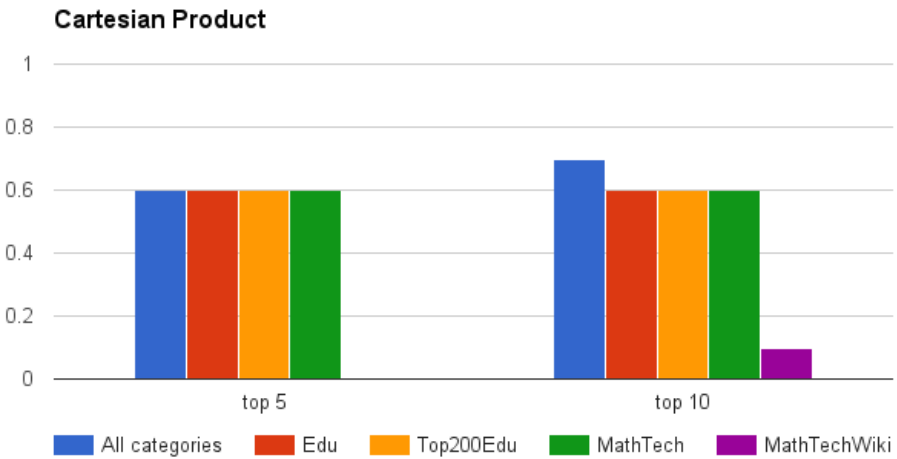
**Figure D.8:** Precision score when the keyword *Chain Rule* is used as a search phrase, with the different white lists applied.



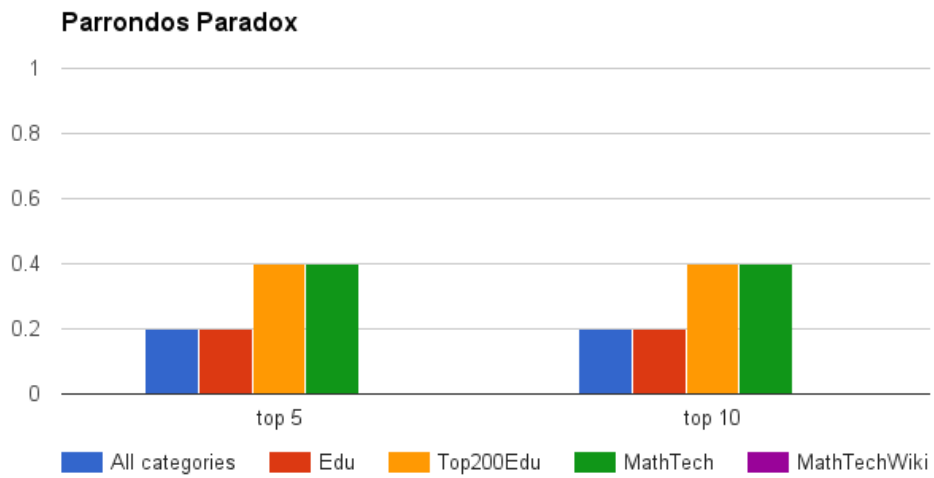
**Figure D.9:** Precision score when the keyword *Prisoners Dilemma* is used as a search phrase, with the different white lists applied.



**Figure D.10:** Precision score when the keyword *Nash Equilibrium* is used as a search phrase, with the different white lists applied.



**Figure D.11:** Precision score when the keyword *Cartesian Product* is used as a search phrase, with the different white lists applied.



**Figure D.12:** Precision score when the keyword *Parrondos Paradox* is used as a search phrase, with the different white lists applied.