Pony: An Information Retrieval System

Indexing, Vector Space Searching, OKAPI BM25 Searching, Web Crawler

Giannis Apostolidis
japostol@csd.uoc.gr

Michalis Athanasakis
michath@csd.uoc.gr

1 Abstract

Information retrieval systems are continuously gain attention through the last fifteen years after the explosion of the Internet. A good performing search engine is always useful and can introduce new understanding of a dataset. We introduce *Pony*, a modern offline indexing and search engine for documents.

2 Introduction

Our implementation is separated under three basic components. *Indexing*, *Searching*, *Graphics User Interface*(GUI). Each component can provide functionality separated from the rest, except GUI. In the phase of indexing the system is guided to a directory of documents for indexing and performs an indexing algorithm described below to finally store a custom file database to the hard drive. In the phase of searching the program retrieves the database stored from the indexing phase and performs searches using two different algorithms. The program can be executed from a terminal but also provides a GUI functionality to improve usability. Under the GUI mode the user can perform both indexing and searching with just importing the directory of documents to be searched. All parts of the program are implemented with `Java`.

3 Internals

For the need of this project we implemented a custom database to store all the necessary information without spending unnecessary size on the disk. The database is separated in three big parts that can be accessed separately from the disk. This enables us to fetch things from the disk when is only necessary during the searching phase. The only part that is on memory while the phase of search is the database of the words(terms) that our system has indexed. The below figures provides an abstract idea on how our system is implemented and how a term is indexed and searched respectively.

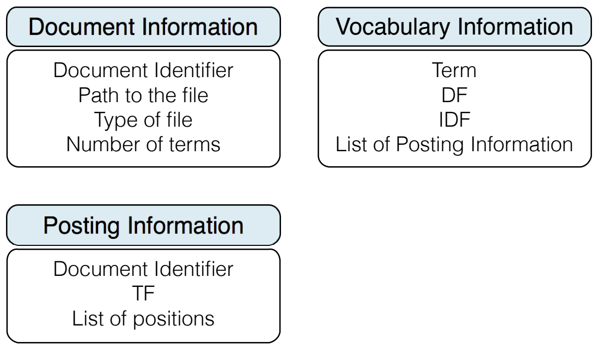

Figure 1: Structure of information entries

The backbone of the database is the Vocabulary database where all unique terms are indexed and later searched. The Vocabulary database is populated during the indexing phase with with all unique terms located inside the document collection. Inside each Vocabulary Entry(Vocabulary Information in the figure) the system store some information and some pointers to other parts of the database for fast access. For each term there is a list of Posting information which it provides more specific information. Posting information holds the document identifier, calculations and the locations of each sighting of the term inside the specific document. Creating a Posting information list for each term essentially provides the system with information about all the sightings of each term. With the usage of the document identifier we can later retrieve the document information, retrieve the file and complete the search. The final output provides plenty of information about the location and the similarity of the returned document to the specific search we request.

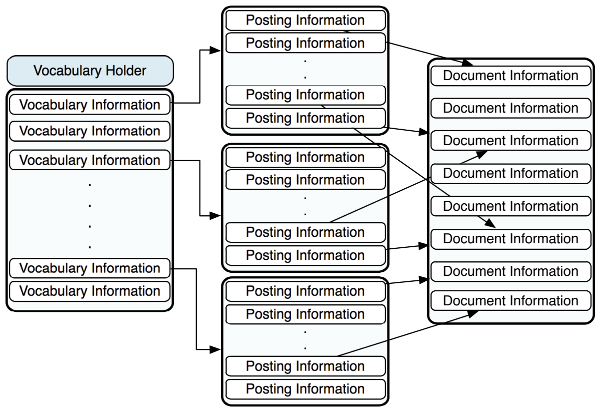

Figure 2: Database structure

When we use the term *list* we imply a data structure responsible to store the information. The actual implementation for each specific data structure is chosen after considering each special characteristic of that part of the database and other factor, performance on writing and on reading.

4 Indexing

The phase of *Indexing* is the phase where the database is initialized using a specific document collection provided by the user. This phase can be done both by terminal or the GUI our system offer. The system has the ability to index documents type of *.txt* with UTF-8 encoding. This provides the ability to index documents in languages different of English. Pony also offer the functionality of stop words, words that when located in the documents are ignored. Along with the stop words specific types of data are ignored. The user has the ability to select a directory with the stop words files of his choice to feed the system to alter the indexing terms. After the valid words have been processed another part of the algorithm is to provide some *stemming* functionality to improve the indexing. The list below provides the types and the behavior the system has on them.

Numbers All digits are ignored from the search. In the case a digit is located inside a word during indexing then the word is separated into two words, the word before the digit and the word after. After that, each of the newly created word is again processed to finally separate all the words from the numbers. All the terms are then processed into indexing.

Symbols A large variety of symbols are ignored during the search. In the case a symbol is located inside a word during indexing then the word is separated into two words, the word before the symbol and the word after. After that, each of the newly created word is again processed to finally separate all the words from the symbols. All the terms are then processed into indexing.

List of all the symbols can be located inside the directory:

`/src/PonyIndexer/stopWords.java`

5 Searching

Searching is the second phase of this system. After all the documents are indexed the user can search in them. The search returns a bundle of information to help the user understand how similar is the document that was returned regarding his search terms. Along with that snippet of text of the documents are returned along with the specific offset from the start of the file where the term is located. The search engine provides the same functionality with the usage of stop words and by ignoring the same numbers and symbols. Also, the search engine ignores

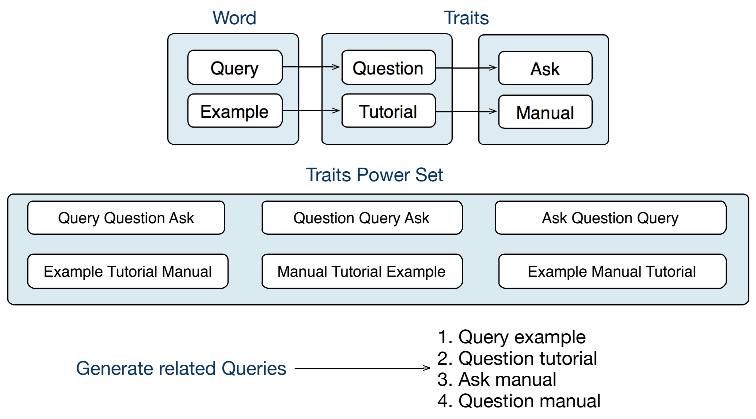

Figure 3: Generation of related queries

the capital and lower letters improving the returning results. In some languages like Greek also punctuation marks are also ignored to offer another improvement. Finally, the search terms are also being stemmed to offer compatibility between indexing and searching.

The actual searching algorithm has been implemented using two separated models. The one model is the *Vector Space* Model and the other is based on the *OKAPI BM25* standard.

6 Searching Optimizations (*Phase B*)

Searching Optimizations are separated in two different aproaches. Both schemes utilize the *Universal Word net(UWN)* to fetch related words to each term of the query. The first scheme tries to improve accuracy by adding specific terms to the inputed query from the user. For each query term the UWN returns a list of words, their lexicograph family and how related are with the term. Using these data the most related synonyms, hyperonysms, hypnonysms, holonyms and meronyms are selected and inserted into the actual query. This scheme improves the accuracy and the raw findings of the search engine by returning a subset of documents that may not contain the actual terms but their closest related words. The other optimization scheme is the generation of Related Queries to suggest to the user. It is possible a user might not know how to express a question he/she has and the suggestions of related queries can help improve the search by expressing the same or similar questions. The way this scheme works is through a related query component. This component is trying to generate related queries using a list of words and their traits as input. The typical procedure is seperated into two basic steps. First, it calculates a power set using each word's traits. This procedures produces as a result a list of words that has a list of power set of traits. In the second step it is trying to extract the newly generated related queries using the power set as an input. We can assume that a related query is one column from the traits of the powerset list.

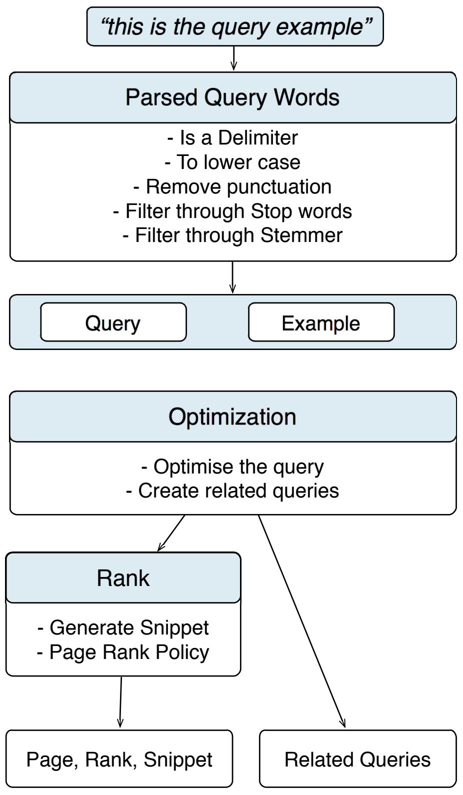

Figure 4: Query structure

7 GUI

The system provides command line functionality but the user is advised to prefer the GUI interface for better optical and structured results. The GUI provides functionality to both index and search documents. If a document indexing database does not exist the GUI being provided with the documents collection directory and a stop words directory can index them and ”export” a database. If the database already exists in the specified directory the search engine loads it and is ready to provide search functionality. After the search the results are being displayed inside the same GUI along with a variety of information.

8 Web Crawler

Web crawler is a mining tool over the web. The purpose of this tool is to download as many web pages as possible to a target storing layer. This tool can be used as the first step of a search engine. The main functionality is the following:

1. Download a web page given as argument.
2. Parse the page using a Xml parser.
3. Find all the `a` tags, and extract the href attribute.
4. Save only the text content of the page.
5. Repeat the same procedure for every extracted href, as many times as the depth argument.

9 Evaluation

After the design and the implementation of our system we provide a evaluation under 3 scopes. The first two scopes are combined under section "Time & Space Evaluation". The other section describes the accuracy and what are the expected results from the system, both in Phase A and Phase B where our optimization are used.

9.1 Time & Space Optimizations

In this section there are 8 charts and graphs to provide information on how the system is being evaluated under many different aspects. What we notice are the following.

- The size of the indexing files are almost as much size of the document collection we examine. Still, we believe that when the document collection will get even bigger the size of indexing database will not change that drastically.
- Due to changes in the implementation of Phase B the searching mechanism is even faster and returns better and more ordered documents.
- Even when we run the evaluations multiple times, one hundred times for each number of query terms times four times the minimum and maximum times are pretty random and are related to what the rest of the system does that 3 ms. The average values shows how stable the system is.

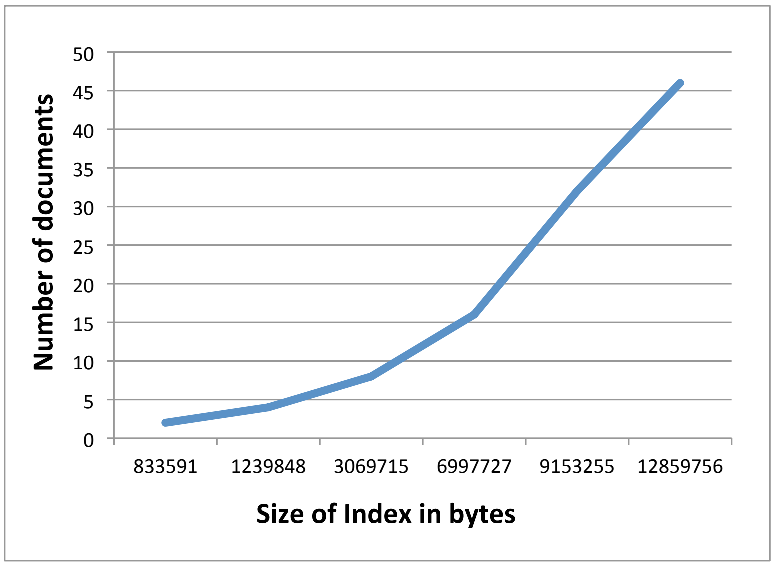

Figure 5: Number of document collection and their respective size of the index database.

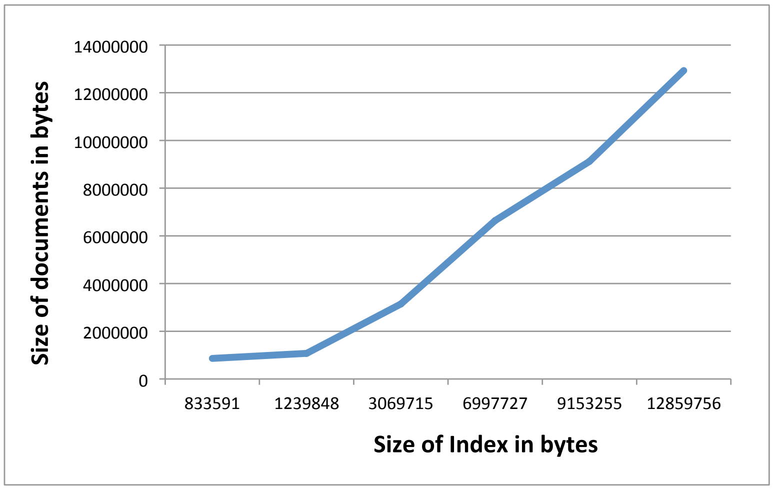

Figure 6: Size of document collection and their respective size of the index database.

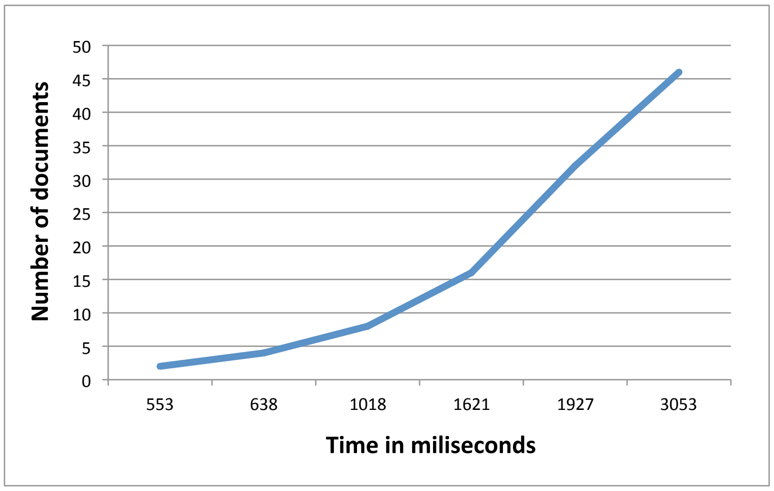

Figure 7: Size of document collection and the time it is required for indexing.

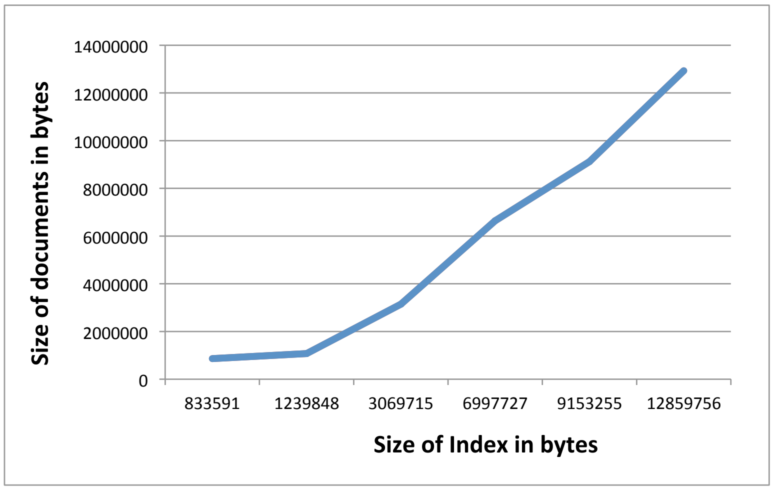

Figure 8: Size of document collection and the time it is required for indexing.

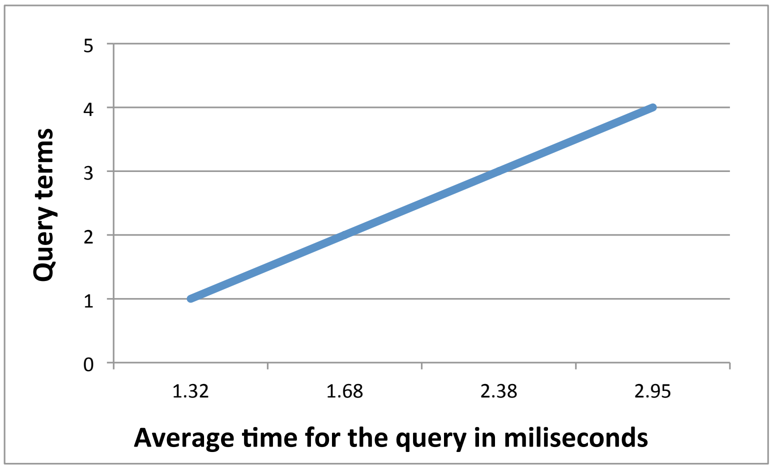

Figure 9: Length of the query and average time to search for system A.

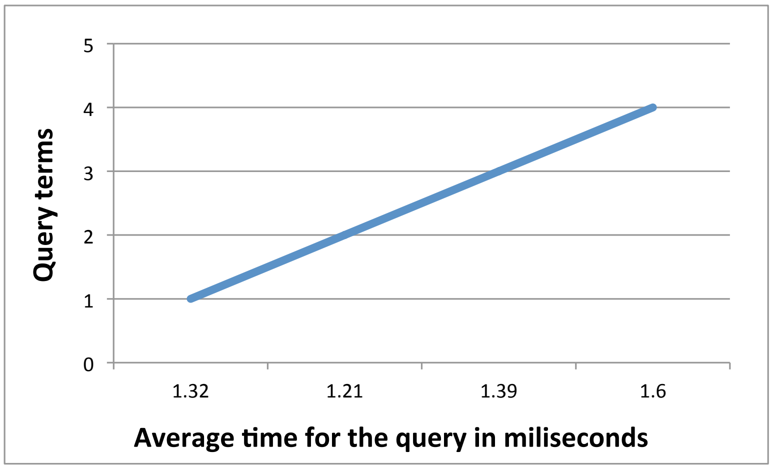

Figure 10: Length of the query and average time to search for system B.

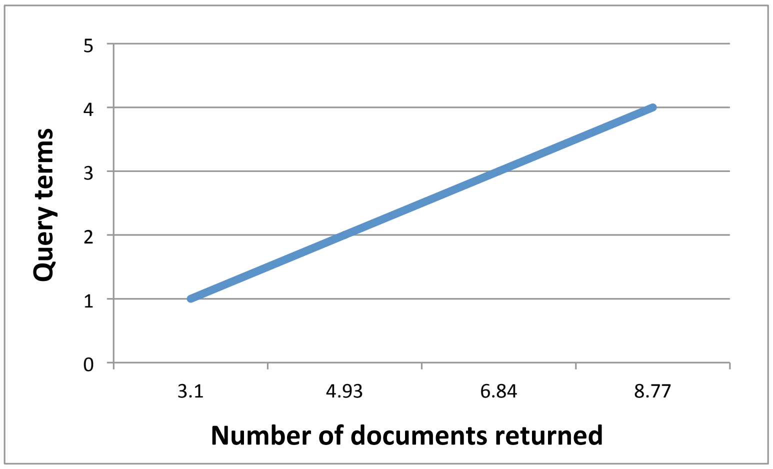

Figure 11: Length of the query and average returned documents for system A.

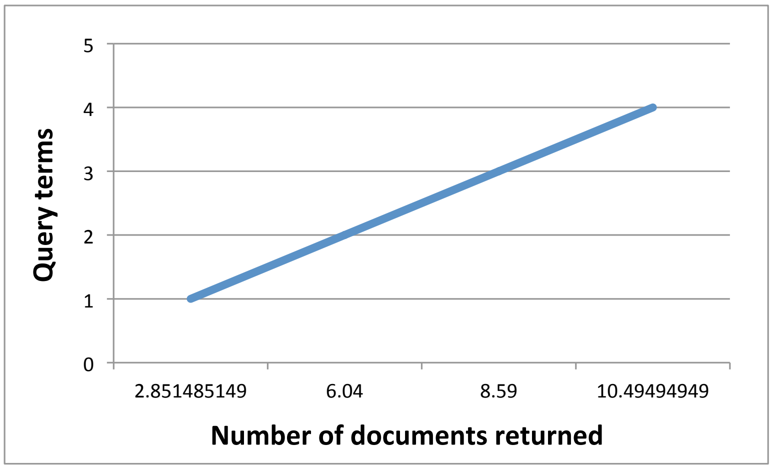

Figure 12: Length of the query and average returned documents for system B.

Table 1: **System A: Times in ms for each number of terms in the query**

# Query terms	Minimum	Average	Maximum
1	0	1.32	11
2	0	1.68	10
3	0	2.38	17
4	0	2.95	17

Table 2: **System A: Number of returned documents each number of terms in the query**

# Query terms	Minimum	Average	Maximum
1	1	3.01	20
2	1	4.93	21
3	1	6.84	30
4	2	8.77	30

Table 3: **System B: Times in ms for each number of terms in the query**

# Query terms	Minimum	Average	Maximum
1	0	1.32	22
2	0	1.21	14
3	0	1.39	6
4	0	1.60	11

9.2 Accuracy

We do our accuracy tests by implementing the following scheme.

1. We choose randomly 20 words from the vocabulary database.
2. For each of these words we run the UWN database and we take, randomly, four related words and we store it to a temporary structure.
3. Using all of the above words we start creating documents, we create ten documents with 20 words inside each document.
4. The words to be put at each document are chosen in this way. In the first step, we choose 7 to 10 words from the vocabulary database. After this, for each words from the chosen above we add 0 to 3 synonyms etc inside the documents.
5. We read the documents manually, we the humans, and then we created four queries with some relation to the words we read.
6. We input the queries to the system and we observe and evaluate the results.

Our findings are the following. The system in Phase A is worse in every aspect such as Precision, Recall and F-measure and R-Precision. Still, it shows that the fallout is better from the system at Phase B. This is due to the fact that we use extensively the UWN database to enrich the user's seach query. The weight we assign to the results of the UWN DB are static numbers and we reach to the conclusion that should be adjusted, even with feedback from the user, to provide more accurate results. Another point to bring up is that in order to evaluate the system we need a more extended description on how to evaluate a system and a proper common document collection to perform the evaluation with predefined correct answers and their defined progression as returning results.

Table 4: **System B: Number of returned documents each number of terms in the query**

# Query terms	Minimum	Average	Maximum
1	1	2.85	23
2	1	6.04	24
3	2	8.59	36
4	2	10.49	36

10 Build instructions

Our system was deployed with: `Netbeans 8.0` and `Java 1.7.0_45`.Building instructions should be similar to other versions of Netbeans.

The following steps show to import the source and dependencies into a Netbeans project and build it from there.

1. We create a new Netbeans Java project by
   `File > New Project...> Java Project with Existing Sources`
2. We name the project *Common* and the store location and press `Next`
3. We add the source folder located at the *Common* folder using `Add folder...`
4. After the project is created, we import the *Stemmer* library located at `Common`
   `dependencies`.
5. We follow the steps 1 to 3 to create another project under the name "PonyIndexer" and we import the existing source files from the directory `PonyIndexer`
   `src`.
6. We again import the Stemmer library and this time we also import our created project "Common" by right-click on `Libraries` of the "PonyIndexer" project and by following the steps of choice `Add Project...`
7. Final step is to build the project. Almost every package has it's own main and we can build the GUI separately.

11 Run instructions

- If you use Netbeans just Build & Run.
- For a command line environment we simply change directory to `dist` and we run the command `java -jar NameOfJar.jar`. Please be make sure to copy all the necessary libraries.
- For the GUI, simply double-click the jar file.