E-Shop
A Vertical Search Engine for Domain of Online Shopping

Vigan Abdurrahmani, Lule Ahmedi and Korab Rrmoku
Faculty of Electrical and Computer Engineering, University of Prishtina, Kodra e Diellit, Pristina, Kosovo

Keywords: Search Engine, Web Crawler, Indexing, Search Ranking.

Abstract: Along with general search engines which search the entire web, vertical search engines that are rather focused on specific domains of search have also gained in popularity in web search. In our case we are focused on domain of online shopping. We treat main processes of search engines which include crawling, indexing and query processing. Each process is developed based on general search engines and adapted to tackle the problems of the specified domain. Specifically, a search result ranking algorithm called Warehouse, which may be adapted to rank results of any specific domain, is introduced.

1 INTRODUCTION

Search engines have appliance in many different fields including searching medical literature, enterprise servers, and usually general search in web (Croft et al., 2010). Depending on their search domain these engines must address different issues, for example desktop search engines must index new documents, emails, web pages. On the other side web search engines like Google must crawl large scale of information and must have very short query response time. Issues that must be addressed by our system does not differ too much from general web search engines, because basically it's a web search engine. The only difference is that we focus on field of online shopping and not on the entire web. Main issues that must be treated in our system include efficient search, response time, freshness and extensibility.

In our paper we discuss three main components of search engines: Input agents, database engine and query servers (Connolly and Hoar, 2014). Input agents handle process of information collection, parsing and text processing of data before indexing process. On our system we focus on finding and extracting product articles on commercial websites. We have developed a scrapper for specific websites which retrieves minimal information required to identify a product, not the entire document/page (more on Section 4).

The database engine's main purpose is to store the information in a manner that can it be found easily during query processing, usually through inverted indexes. Data and index structure strongly depends on the ranking algorithm. In our case we have chosen a relational database system as data storage which offers more flexibility to convert user input into structured queries

The most important component of the system is query processing which determines the quality of the search. It consists of the ranking algorithm which determines the relevant information for user's request. In E-Shop system we have developed an algorithm called warehouse who treats articles as nodes with properties and applies some filters (feature functions) to determine the document score.

2 RELATED WORK

Web Search engines have existed since 90's. Early systems only searched for string matches inside large archives of documents (called web directories), but their efficiency was low. However during the time there have been developed many algorithms for intelligent and efficient ranking during the search (example Google's PageRank algorithm (Brin and Page, 1998)).

Today's search engines aren't focused only on entire web, but also on many specific domains (called as vertical search engines). Most known existing search engines for our domain include

Abdurrahmani, V., Ahmedi, L. and Rrmoku, K.
E-Shop - A Vertical Search Engine for Domain of Online Shopping.
DOI: 10.5220/0006366303760381
In *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*, pages 376-381
ISBN: 978-989-758-246-2

Google Shopping (Vise, et al., 2006), Yahoo Shopping[1] and local engines implemented in e-commerce sites (Zhou et al., 2013). However our approach is not based on any existing system for domain of appliance. Instead we have combined well known principles of general search engines and basic principles of Google Shopping.

3 SYSTEM ARCHITECTURE

E-Shop's architecture consists of three internal components and four external systems which are used as source for information retrieval. These external systems are web systems for online shopping and are labeled as web stores (Amazon, Ebay, Walmart and Ali Express).

The internal components consist of:

- Web Server,
- Database Server, and
- Windows Service.

Database component (Article Storage Server) serves as storage for retrieved information, and shortens the time needed for query processing because there is no need to re-retrieve and re-structure the information. The Web Server is the central component of the system which is involved nearly at all processes, and serves as a communication bridge between other components. The main purpose of Windows Service (Crawling Monitor Service) (Microsoft, n.d.) is to monitor the crawling and the indexing processes, where in this case the service is used as master who invokes tasks to other components. The reason why we have used a windows service is because web server is a passive entity and cannot invoke commands to himself.

Below is listed the physical model of the system and the interactions between its components (Figure 1).

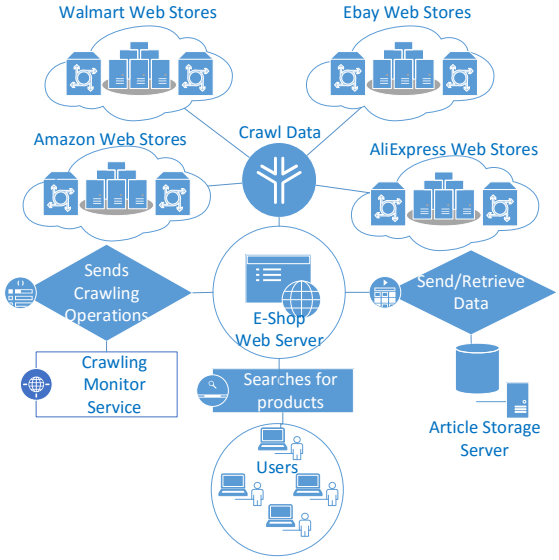

Figure 1: System architecture.

4 INPUT AGENTS

The indexing process is initialized by the windows service (master) based on a schedule when we suppose that the traffic at the system is lowest. First, the master requests the list of indexes and phrases to be crawled from the REST API of web server (slave). These indexes/phrases consist of two lists where the first one contains the phrases that doesn't relate to articles on the system's storage, and the second one includes the indexes (keywords) that are outdated. For the first list, for each phrase we send crawling request to the slave and also break the phrase into indexes which will be appended to the second list (outdated indexes). We need to mention that each term is considered outdated 1 month after last crawling to keep retrieving fresh information.

The crawling process starts when the master sends an index or phrase crawling command to the slave. Upon receiving the request, the slave starts 4 child threads (Figure 2) where each worker handles information collection for specific web store.

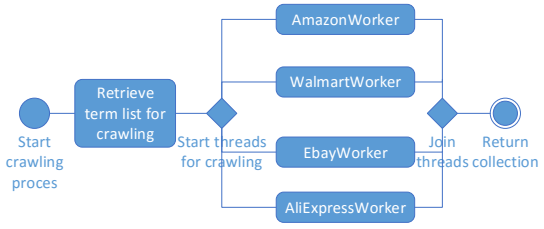

Figure 2: Crawling process activity diagram.

[1] https://shopping.yahoo.com

The worker (thread) uses a crawling component (called as Web Spider) to collect relevant data. Each web store has specific web spider which adapts to the structure of the web page. The spider component only crawls the first page and then uses a scrapper component (DOMNode) to extract the products. The scrapper treats the pages as XML documents and performs XPath queries and regular expressions to extract the information. In the end the scrapper performs a final check to discard the products which contains incomplete information.

After collecting and transforming the information each worker's collection is merged into single collection and prepared for indexing process. Below in Figure 3 we have listed a diagram showing the interaction between the crawling components.

Figure 3: Interaction between components of crawling process.

The weakness of the current implementation of crawling process is that if we want to extend the crawling to other web stores, we need to develop a new scrapper/spider component that adapts the structure of the web store. Also if any of existing web stores change their structure no information will be collected and therefore we need to rewrite the scrapper/spider component of specified web store.

The priority of current design is that it uses a single thread for each store and single request for each store. The only overhead is the time required to initialize the threads and merging the results which is still more efficient than single thread operation. Also we need to mention that we crawl information from other information retrieval systems, so without any knowledge how their algorithms works we benefit from them. There are also some side effects from the external systems because they have their policies how to rank the search results, and sometimes the higher ranked products are not much relevant. This is especially expressed in case of Ali Express store. To tackle this problem we have added an attribute which gives a relevance coefficient to each web store.

5 INDEXING PROCESS

Upon receiving the data collection from crawling process the slave first stores the articles to storage. Then the slave maps the relation between articles and keywords. This way we create the inverted index where the keyword points to articles.

The model of database storage that we have chosen is relational database model (Figure 4). The reason that we have chosen this model is that because it creates indexes for fast access by using key constraints. We have decided to implement numeric identifiers for both articles and index terms (keywords) because the search for numeric values is way faster than text search.

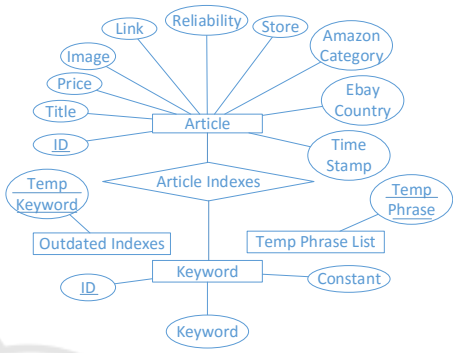

Figure 4: Entity Relationship model.

Each index also has a specific constant which is stored along the term on the database. This constant is calculated by the formula:

$$KC = 1 - \frac{F}{C} \quad (1)$$

In the above formula F represents the number of articles who contain the keyword (index), and C represents the total number of articles.

This constant (KC) defines the inverted term frequency for the specified term (Croft et al., 2010). These constants are updated every day during the low traffic time in system by the event schedulers provided by database. The constants of inverted indexes have big importance on Warehouse Algorithm which will be discussed on Section 7.

In our indexing process we haven't included stemming and stopping processes (Croft et al., 2010) which will reduce the efficiency of query process. The stopping process should not be included because we deal with many product brands and if we stop terms like "and" we may reduce relevance of retrieved information (e.g. "AND 1" brand). However, the inverted term frequency helps by reducing the weight of stop words. The stemming process will be treated on future works.

6 QUERY PROCESSING

Query processing is realized through simple steps by retrieving the products that match the user keywords and then by processing the Warehouse algorithm to rank the results. In our system we do not provide query expansion features or user tracking. We only transform the user query to SQL query to limit the relevant article list.

The first step of query processing involves transforming query input into indexes (keywords). After transformation we check into storage which indexes exist and add the indexes that doesn't exist to the list of outdated indexes for later crawling. Also if at least one keyword (index) doesn't exist we add the phrase to the list for later crawling. After this step we check the percentage of indexes that exist in storage. If percentage of found indexes is greater than 60% we consider that we have relevant data for user's query and perform the warehouse algorithm. If the percentage is lower than 60% we consider that our system doesn't have relevant data for user's query and we rely on external systems (web store) by performing direct crawling. In this case we do not store any data to the storage, we just show the retrieved ranked results from web stores (Figure 5). The only ranking that we perform to these results include relevance coefficient of web stores.

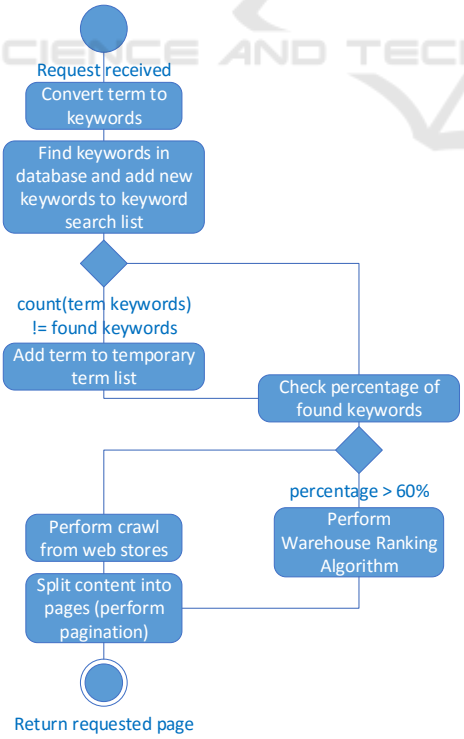

Figure 5: Workflow of query processing.

7 WAREHOUSE ALGORITHM

Warehouse ranking algorithm is a document at time scoring algorithm. The idea of this algorithm is to treat related data structures as node with properties, and then by applying featured functions (in our case called as filters) to the properties we compute the final document score where the score is ≤ 1. In warehouse algorithm we can extend featured functions to adapt the needs of domain of appliance. The only required featured function is "At least one rule".

7.1 Featured Functions

In Warehouse we have to kinds of filters that we apply to the nodes:

- Global filters,
- Local filters

The local filters are featured functions that apply to particular property of the node, example: word matching to the title of a book. The global filters are featured functions that apply to whole or a set of properties of the node, example: priority of sections of a web page. After performing all featured functions we calculate the final result as algebraic sum of product between local filters and global filters:

$$WR(n) = G_1 \times F_1 + G_2 \times F_2 + \cdots + G_n \times F_n = \sum_{i=1}^{n} G_i \times F_i \quad (2)$$

In the above formula G_i represents global filters, F_i represents local filters and $WR(n)$ the final document score for document n.

In next sections we will treat the featured functions applied at E-Shop system.

7.1.1 At Least One Rule

This rule belongs to local featured functions. Its purpose is to eliminate every node that don't have at least one of the search terms, this way it shortens the number of nodes to be ranked. In our case it applies to products where we eliminate the products that don't match any of the terms in user's query. This filter must be included at every application of warehouse algorithm no matter of the domain of appliance.

7.1.2 Word Closeness Rule

This rule also belongs to local featured functions, and its purpose is to search for word matches in a given text. The first step of this rule is to check if entire phrase is found in node's property. If there's no match the phrase is split into smaller phrases where the new phrases have length of $n - 1$ (n – total number of words in phrase). The splitting is done in way that we start at the beginning of the phrase and we continue to remove the first word and add new word at the end. If no match is found again we continue the splitting into $n - 2$, so this process is continued until the phrase length is one word. If the phrase length is one word the returned result will be 0. In case we find a match we apply the following formula to compute the result of the rule:

$$WCR = F \times \frac{(N - i)}{N \times C} \quad (3)$$

In the formula above F is the number of matches found in search in i depth, N is number of words in search term, C is number of words in property/text, i is number of phrase subgroups of search term.

```
ALGORITHM WCR(text, term)
C ← WordCount(text);
N ← WordCount(term);
F ← 0; i ← 0; k ← N; WCR ← 0;
wordGroups ← term;
while k > 1
    F ← CheckForMatches(property,
wordGroups);
    if F > 0
        WCR ← F x (N - i) / (N x C);
        break
    else
        k--; i++;
        wordGroups ← SubgroupTerm
(term, k)
```

To clarify more the rule below we have depicted a diagram in Figure 6 describing how the rule splits the groups until a match is found.

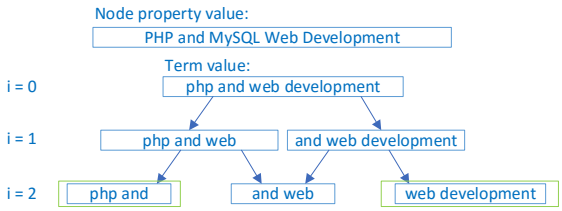

Figure 6: Appliance of Word Closeness Rule.

In the example above the result of WCR rule will be as follow:

$$WCR = F \times \frac{N - i}{N \times C} = 2 \times \frac{4 - 2}{4 \times 5} = 0.20 \quad (4)$$

In our system this rule is applied on title of the product. Disadvantage of this rule is that it only should apply at short texts. In large texts the search time will be very long.

7.1.3 Property Priority Rule

The purpose of this featured function is to define the weight of particular properties in manner that the sum of all weights must be 1, and belongs to global filters. In our case we divide the weight between only three properties: title (0.3), keywords (0.5) and reliability (0.2). The distribution of weights is not based on any analysis.

The final score of article with Warehouse ranking algorithm in our case will apply the following formula:

$$WR(n) = WCR(Title) \times 0.30 + KC(Keywords) \times 0.50 + Reliability \times 0.20 \quad (5)$$

To clarify how the warehouse algorithm is implemented in our system below is depicted a diagram that describes its appliance on a simple node (article in our case). In Figure 7 we show the applied local feature functions (filters on top of properties) and the applied global feature functions (filter on the left side). Also there are some extended featured functions that can be used using advanced search based on filters chosen by user interface.

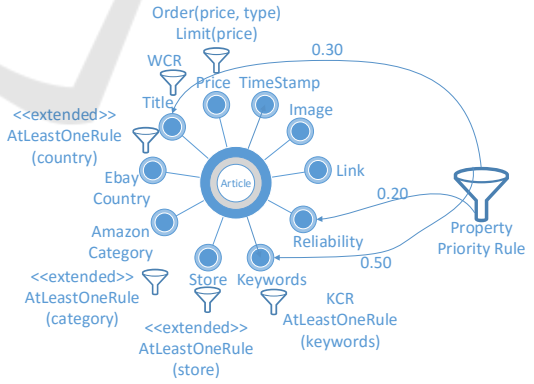

Figure 7: Appliance of Warehouse algorithm on a single node.

8 USE CASES

In this section we will provide two use cases. The first one demonstrates the situation where we

consider that we have relevant information on our system, and the second one where we perform direct crawling from web stores. The dataset where the query is performed consist of 72 index terms (mostly technology brands) and 7756 articles.

8.1 Use Case – Relevant Information Exists

In this use case we perform a query with keywords (indexes) that already exist on our system, so that we evaluate the efficiency of our ranking algorithm. The query that we request is “acer laptop chromebook” (Figure 9) where all the indexes already exist on storage and have relations to existing articles.

In this case the total time spend from request to response visualization is about 146.96ms and the top three results are shown on the Figure 8Figure where we see that the results are accurate.

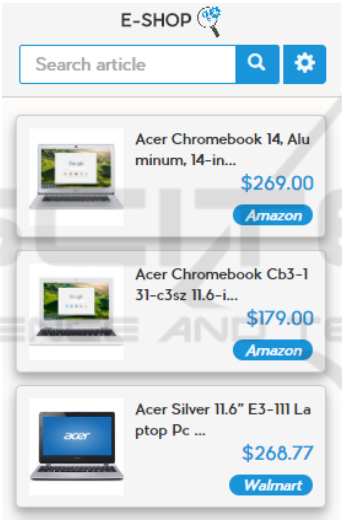

Figure 8: Search result page for query “acer laptop chromebook”.

8.2 Use Case – No Relevant Data Exists

In the second use case we perform the query "hugo" and the result contains mostly articles about books, movies and perfumes. The total time of response is about 2.41s and based on the search results we have seen that this approach is not providing highly relevant data (also the query which consists only from one word).

9 CONCLUSIONS

In our system we have developed an environment that adapts general search engines to the needs of specific domain. We have treated main functions of search engines (crawling, indexing, query processing), and tried to develop original methods for each of the processes. We also have developed a ranking algorithm that can be adapted to any dataset with simple modifications.

However we haven’t discussed how the system could be distributed on multiple machines, which may be treated on future works. Other problems that could be treated include developing stemming techniques for our system, making recommendations based on geo locations, and improving featured functions of warehouse algorithm. Additionally, on future works we may evaluate the system with larger datasets that may be acquired by gathering data from different queries entered by users.

REFERENCES

Anon., 2016. *Moz. Basics of Search Engine friendly design and development.*. [Online] Available at: https://moz.com/ beginners- guide-to-seo/ basics-of-search- engine- friendly- design- and- development [Accessed July 2016].

Brin, S. & Page, L., 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine.. *Computer networks and ISDN systems*, 30(1), pp. 107-117.

Clark, J., 1999. *XSL Transformations (XSLT).* [Online] Available at: https://www.w3.org/TR/xslt/

Clark, J. & DeRose, S., 1999. *XML Path Language (XPath).* [Online] Available at: https://www.w3.org/TR/xpath/

Connolly, R. & Hoar, R., 2014. *Fundamentals of Web Development.* 1st ed.:Pearson Education.

Croft, B. W., Metzler, D. & Strohman, T., 2010. *Search Engines, Information retrieval in practice.* 1st ed.:Pearson.

Microsoft, n.d. *Introduction to Windows Service Applications.* [Online] Available at: https://msdn.microsoft.com/en-us/library/d56de412(v=vs.110).aspx [Accessed 2016].

Vise, D. A., Malseed & Mark, 2006. *The Google Story.* reprint ed.:Delta Trade Paperbacks.

Zhou, K., Cummins, R., Lalmas, M. & Jose, J. M., 2013. *Which Vertical Search Engines are Relevant?*. Rio de Janeiro, Brazil, ACM, pp. 1557-1568.