Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semester Thesis

Improving Keyword Search in ConfSearch

Andrius Paukste

Supervisors:

Kuhn Michael

Prof. Roger Wattenhofer

Zurich

2009

Abstract

Confsearch (www.confsearch.org) is a search engine for computer science conferences. It offers several search types, one of which is keyword search. The keyword search mechanism is based on an analysis of publication titles. The goal of this semester work is to improve the keyword search engine. The two main contributions of this work towards that goal are integrating a stemming algorithm and finding the best phrase search solution. Finally, we evaluate the results and compare it to the existing keyword search engine implementation.

Table of Contents

Abstract **2**
1 Introduction **4**
1.1 Problem Description 4
1.2 Motivating examples 4
2 Background **9**
2.1 Stemming 9
2.2 Inverted index 9
2.3 Phrase search 10
2.3.1 Stop words in phrase search 10
2.3.2 Biwords 10
2.3.3 Positional index 11
2.4 Tf-idf 9
2.5 Confsearch 12
3 Related work **13**
3.1 Stemming 13
3.2 Phrase search 13
3.3 Confsearch 13
4 Phrase search **14**
4.1 Stemming 14
4.2 Stop words 14
4.3 Tf-idf extension for phrase search 14
4.4 Phrase search algorithm 15
5 Evaluation **18**
Conclusion **20**
References **21**
Appendix A. Implementation description **22**

1 Introduction

1.1 Problem Description

A scientific conference is a (typically annual) event, where the academic community specialized in a certain field gathers to present and discuss its most recent research results. Such conferences play an important role in science as well as in the life of every researcher. Experience shows that choosing the right conference to publish a new result is not always a simple task. Many factors, such as thematic scope, deadline for submission, quality, or location have to be considered. Traditional search engines do a rather bad job in finding appropriate conferences. *Confsearch* (www.confsearch.org) is a search engine for computer science conferences. It is based on data of *DBLP* (http://dblp.uni-trier.de), a digital library project containing the publications of a large number of computer science conferences. *Confsearch* currently offers different search modes, such as searching for conferences by keyword, by authors, or by other (similar) conferences.

The keyword search mechanism is based on an analysis of publication titles. The current implementation has two major shortcomings. First, it treats each keyword individually, without accounting for word combinations or *phrases* (e.g. “social network”, “computer science”, etc.). Second, the current algorithm is not able to identify semantically identical words, only differing by their grammatical form (e.g. “network” vs. “networks” vs. “networking”).

The following examples will illustrate this problem.

1.2 Motivating examples

Example 1. Most of widely knowing web search engines apply semantic word analysis in one or another way to improve search quality. In the following example we use the phrases “*service oriented architectures*” and “*services oriented architecture*”. Figures 1.1 and 1.2 show that in some cases the existing *confsearch* implementation produces completely different results for these two phrases. The same queries in one of the popular web search engines www.google.com, on the other hand, return almost the same results as we can see in Figures 1.3 and 1.4. Even more, Google finds pages where terms are not exactly equal to the query terms.

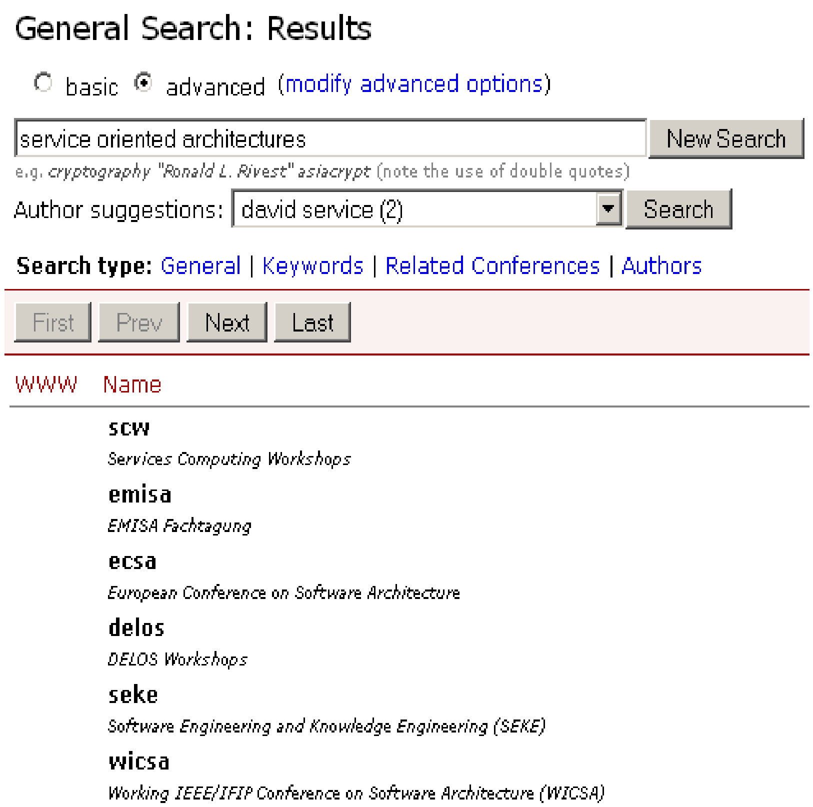

Figure 1.1. Confsearch search results for the query "*service oriented architectures*"

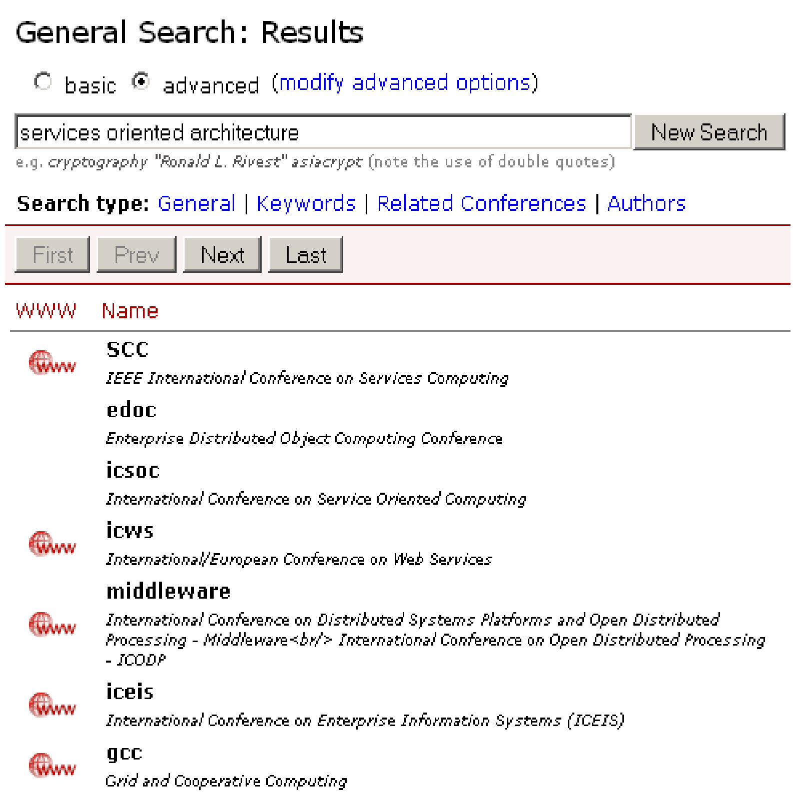

Figure 1.2. Confsearch search results for the query "*services oriented architecture*"

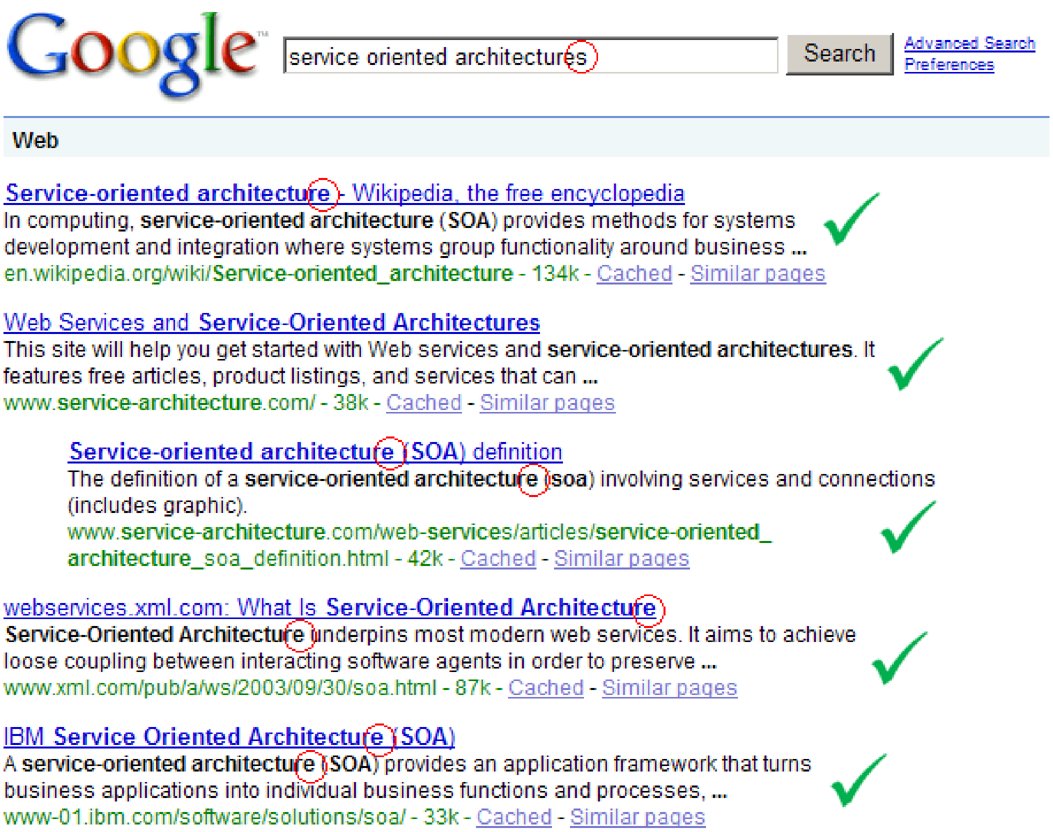

Figure 1.3. Google web search results for the query "*service oriented architectures*"

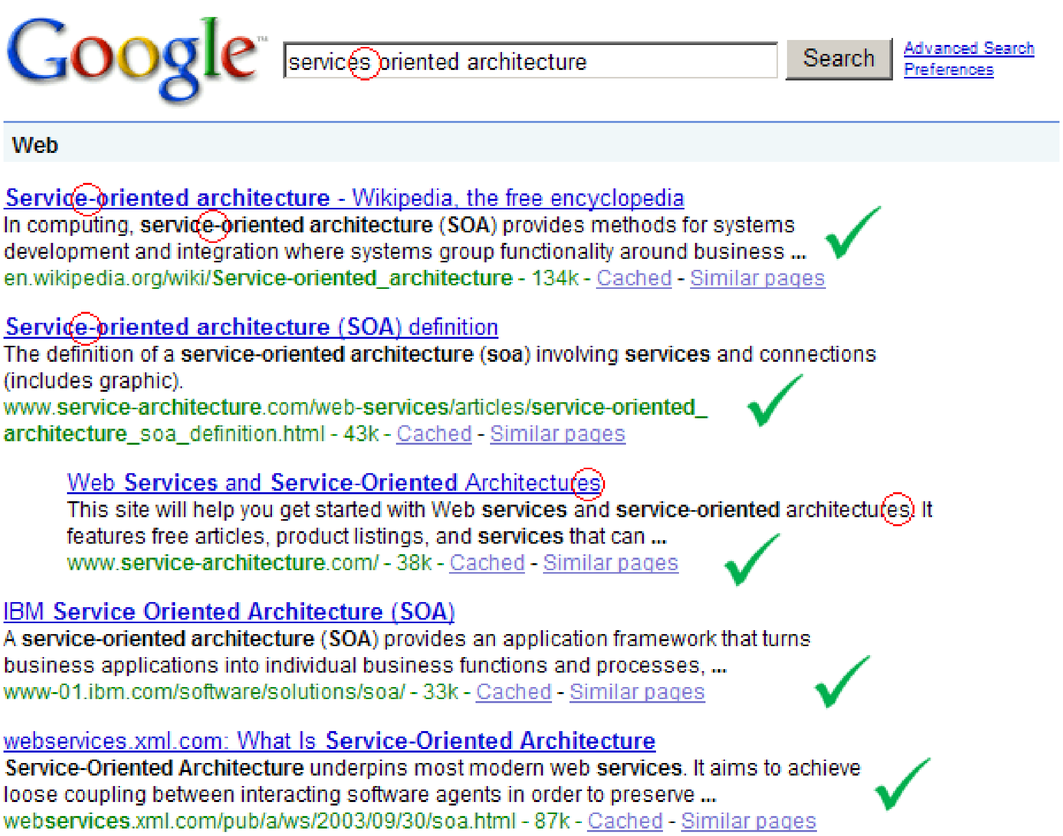

Figure 1.4. Google web search results for the query "*services oriented architecture*"

Example 2. The following example shows that without a phrase search algorithm we can get results from completely other computer science topic than what was meant by the query. If we look at the publications topics in first three conferences we will find a lot of words "*real*", "*time*" and "*systems*", but only one conference has publication with whole phrase "*real-time systems*" (See Figure 1.5)

Figure 1.5. Confsearch search results for the query "*real-time systems*"

$_{kj}$

2 Background

2.1 *Stemming*

Stemming is the process of semantic words analysis with the goal to get a word's base form (stem). Stemming can be very useful to search in text documents. The main benefit of using stemming is that we can find documents which contain different forms of a word than the search query. For example, if the query is "*social network*" we become able to find documents that contain the phrase "*social networks*".

2.2 *Inverted index*

Inverted index store term or term id and set of document id's where term appears. This type of index often used to search for the distinct term in the document.

Inverted index creation steps:

1. Split the document into terms with possible term preprocessing (for example stemming).
2. Store document id for every unique term.
3. Do steps 1 and 2 for every document.

Term	Document id's
social	1, 5, 9, 15, …
web	4, 5, 11, …
…	…

Example 2.1. Inverted index example

2.2.1 Tf-idf

Tf-idf (term frequency and inverse document frequency) is used to calculate term weight in a document and document collection, also to decide which term is more important. The importance is directly proportional to the number of term's occurrence in the document and indirectly proportional to the number of document in the collection where the term appears.

Tf increases tf-idf values for documents where term appears often and idf decreases tf-idf values for collections where term appears in a high amount of documents.

$$tfidf \quad = tf \quad * idf \quad ,$$

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}$$

Where $n_{i,j}$ is number of occurrence term *i* in document *j* and $\sum_k n_{kj}$ is sum of all terms in a document *j*

$$idf_{term} = \log(\frac{\mathrm{T}}{\mathrm{N}_{\mathrm{term}}})$$

Where T is total number of documents and $\mathrm{N}_{\mathrm{term}}$ is number of documents where term appears.

2.3 Phrase search

Phrase search is a type of text search that searching for a specified phrase. Search engines usually recognize phrase queries from a double or single quotes : "*information retrieval*".

2.3.1 Stop words

Stop words are used to eliminate unimportant words in document. To decide which word is unimportant we can use *tf-idf* values. For example the sentence "*'Barriers and solutions to the development of online advertising in China*" have five stop words: *"and" "to" "the" "of"* and *"in"* which are very common to text documents and has low *idf* value. However, stop words are not always beneficial for phrase search. The phrase "*usability evaluation*" has a different meaning than "*usability and evaluation*". We let this problem for future research and integrate stop words into the phrase search engine, because of important benefits to search results in many cases.

2.3.2 Biwords

A first approach to phrase search is to mark every pair of consecutive terms in a document as a phrase and index them.

Biword	Document id's
social web	1, 5, 9, 15, …
index all	4, 5, 11, …
…	…

Example 2.2. Biwords index example

To use biwords in phrase search, we first create an index of all biwords in all documents. Then we produce biwords from query and search them in the index for all produced biwords. If we have more than two terms in a query we have then to post-process results in order to find documents with the whole phrase.

Example 2.3: The sentence "*index all consecutive terms*:" produces the following biwords:

index all

all consecutive

consecutive terms

2.3.3 Positional index

The use of a positional index is second approach to phrase search. Positional index extends inverted index concept with positional information. In a positional index for every unique term and document we store position list of the term.

Term		**Document id's and positions**				
social		1	5	9	15	…
	positions	*2* *7* *11* …	*1* *3* *6* …	*4* *7* *15* …	*6* *7* *11* …	⋮
web		4	5	11	15	…
…	*positions*	*22* *25* *44* …	*2* *7* *19* …	*1* *5* *9* …	*5* *9* *15* …	⋮
…						

Figure 2.2. Positional index example.

We calculate a positional index for every term in the document as showed in the following example:

Example 2.2: Positional index with stop words elimination:

"Retrieval$_1$ evaluation$_2$ with incomplete$_3$ information$_4$"

Positional indexes are more efficient solution than biword indexes. With the help of a positional index we are able to find not only adjacent terms, but also phrases with additional terms inside. For example, we are able to find the phrase "*information storage and retrieval*" when we are looking for "*information retrieval*".

2.4 Confsearch

Confsearch – conference search engine was developed by members of the Distributed Computing Group at the Swiss Federal Institute of Technology (ETH). It provides various search types and possibility to add a new conference.

3 Related work

3.1 *Stemming*

[MRS08] propose that the most common and efficient algorithm for stemming English language words is Porter's algorithm The algorithm analyzes the word and returns a string that is common to (almost) all the words derived from the same stem. Porter's algorithm was created by Martin Porter who received the Tony Kent Strix for his work.

Example 3. Porter's algorithm examples:

studies -> studi

studying -> studi

study -> studi

3.2 *Phrase search*

[MRS08] report that as many as 10% of all web queries are phrase queries, and many more are implicit phrase queries. Most commonly used approach for phrase search is positional index.

3.3 *Confsearch*

In [KW] Michael Kuhn and Roger Wattenhofer analyzed scientific conference graph and showed that this graph consist at least two layers: thematic and quality. [KW] propose that a single author tends to publish in venues of similar quality, therefore we are able to separate conferences with similar quality. In confsearch quality layer is used to sort query results.

4 Phrase search

4.1 Stemming

Martin Porter released an official and free to use implementation (http://tartarus.org/~martin/PorterStemmer) of the algorithm. We decided to integrate this implementation into the confsearch engine.

4.2 Stop words

In confsearch, we eliminate words which consist of only one symbol or are in a stop word list.

4.3 Tf-idf extension for phrase search

Above definition of *tf-idf* is not directly applicable for phrase search. The main idea is that we calculate not the number of phrase occurrence in the document, but how exactly phrase match query. This is reflected by the weight *w* of a phrase.

We propose following *tf-idf* extension:

$$tf = \log(w_{phrase} + 1) * \frac{w_{phrase}}{\sum_k n_k}$$

where $w_{phrase} = \sum_{1..i} \frac{1}{s_{phrase_i}}$,

$s_{phrase} = \sum_{2..k} s_k$

and s_k is a distance between two terms in a phrase. If distance is less than zero (it means we found reverse phrase, for example "*retrieval information*") we use $-s_k$ and add a special constant RWC(reverse weight constant):

$$s_k = \begin{cases} Index_k - Index_{k-1} & \text{when } Index_k > Index_{k-1} \\ -(Index_k - Index_{k-1}) + RWC & \text{when } Index_k < Index_{k-1} \end{cases}$$

Example 4.2. S_{phrase} calculation with stop words elimination

Publication title	s_{phrase}
"Towards$_1$ the use of Prosodic$_2$ Information$_3$ for Spoken$_4$ Document$_5$ Retrieval$_6$"	6-3=**3**
"Retrieval$_1$ evaluation$_2$ with incomplete$_3$ information$_4$"	-(1-4)+1=**4**

4.4 Phrase search algorithm

Base algorithm idea is to find all publications which consist of all terms from query and then calculate distance between required terms in order to find better matching publications. Our phrase search algorithm is able to find titles where terms are not adjacent, such as "*information storage and retrieval*" when we are looking for "*information retrieval*", as well as reversed phrases such as "*retrieval of incomplete information*". Therefore for each conference we calculate minimal distance between phrase terms in publications and then we use this value to sort conferences. Higher rank conferences has less distance value. If the conferences has the same value we sort them by introduced *tf-idf.*

Algorithm steps:

1. Get indexes for all terms in phrase.
2. Find publication titles which contains all terms together.
3. Calculate distance between terms (*s)* in publications.
4. Calculate minimal distance (m_s) and weight (*w)* for each conference

$$w = \sum_{1..i} \frac{1}{s_i}, \text{ and } mS = \min(s_{1..i}) \quad \text{where } s_i \text{ is } s_{phrase} \text{ in } title_i$$

5. Calculate the modified tf-idf values for each conference:

   $tf - idf = tf * idf$,

   $$tf = \log(w+1) * \frac{w}{\sum_k n_k},$$

   where $\sum_k n_k$ is sum of number of occurrences of all terms in conference titles.

   $$idf = \log(\frac{\mathrm{T}}{\mathrm{N}})$$

   where T is total number of conferences and N is number of conferences where the phrase appears

6. Sort results by m_s and then by *tf-idf values*

Example 4.3 "*information retrieval*" phrase search

1. Separate the phrase into a list of *k* terms: *"information retrieval" ->"information" + "retrieval"*

2. For each term, get indexes and publication ID's ordered by publication ID. Start from first index:

"information"		*"retrieval"*	
Pub ID_1	$Index_1$	Pub ID_2	$Index_2$
→ 5	1	→ 6	7
6	2	9	2
6	10	9	8
7	4	10	1
…		…	

3. Now we have to find identical publications. If Publication ID of $word_{k-1\ 1}$ is greater than Publication ID of $word_k$ or Publication ID of $word_{k-1\ 1}$ is greater than Publication ID of $word_k$ then read new row of $word_k$:

"information"		*"retrieval"*	
Pub ID1	Index1	Pub ID2	Index2
5	1	→ 6	7
→ 6	2	9	2
6	10	9	8
7	4	10	1
…		…	

4. When publication ID's of $words_{1..k}$ is equal calculate *s*:

→ 7 *"information"*	4	→ 10 *"retrieval"*	1
Pub ID1 …	Index1	Pub ID2 …	Index2
5	1	6	3
6	2	7	6
6	11	9	8

S_1 =3-2=1(perfect mach!)

S_2 =7-11=-(-4)+1=5

S_3 =6-4=2

We found 3 phrases "*information retrieval*" with distance between terms 1, 2 and 5 in one conference)

5. Calculate the weight and minimal step:

   $weight = 1 + \frac{1}{2} + \frac{1}{5} = 1.7$, $mS = 1$

6. Calculate the modified *tf idf* values:

   (We assume that the sum of number of occurrences of all terms in conference titles is 80, total number of conferences is 10 and number of conferences where the phrase appears is 1)

   $tf = \log(1.7 + 1) * \frac{1.7}{80}$,

   $idf = \log(\frac{10}{1})$

7. Do steps 5 and 6 for all conferences and then sort results by m_s and *tf-idf*

5 Evaluation

To evaluate the quality of the proposed phrase search algorithm we compare the original *confsearch* engine keyword query results to the results of the new phrase search engine. To answer which results are better we need some references data that identifies conferences which are truly relevant for a given query. Not easy to find such reference. One of the sources we can use is Libra Academic Search(url: http://libra.msra.cn/). There we can find top conferences in a specific computer science field. Then we use the field name as a phrase in *confsearch* query. For evaluation, we count how many of these conferences a given search method returns on the first page (top 20 results), if the name of the scientific field is used as a phrase query.

Phrase	Quantity in original confsearch engine first 20 results	Quantity in new phrase search engine first 20 results (without stemming)	Quantity in new phrase search engine first 20 results
World Wide Web	2	2	3
Artificial Intelligence	0	1	1
Graphics	3	2	3
Real-Time	0	6	9
Machine Learning	1	1	1
Programming Languages	5	4	5
Computer Vision	0	6	5
Distributed Computing	1	3	1
Operating Systems	6	6	6
Overall	**18**	**31**	**34**

Table 1. The results of comparison with Libra Academic Search Top conference list.

As we see in Table 1 the new phrase search improves the query results. Stemming improves results not so dramatically and in two cases we even got better results without stemming. Both occasions includes stemmed word base '*comput*':

computer -> *compute*

computing -> *compute*

To avoid such cases in future work we can try improve the stemming algorithm or create not stemmed words list.

Conclusion

We integrated stemming into confsearch keyword search engine. We then compared different phrase search approaches and proposed an efficient algorithm for phrase search in confsearch. The algorithm is based on a modified *tf-idf* to meet our specific phrase search requirements. The new keyword search engine has been evaluated and results indicate that stemming and phrase search improves confsearch keyword search quality.

References

[MRS08] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. *An Introduction to Information Retrieval* Cambridge UP, 2008

[TH08] Thomas Hoffman. *Information Retrieval* slides, 2008 < http://www.systems.ethz.ch/education/courses/hs08/information-retrieval/course-materials/index >

[KW] Michael Kuhn, Roger Wattenhofer. *The Layered World of Scientific Conferences*

[KW1] Michael Kuhn, Roger Wattenhofer. *Semester Thesis: "Improving Keyword Search in ConfSearch"*

[W1] Wikipedia, the free encyclopedia. *Stemming* [viewed: 2008-10-15]. <http://en.wikipedia.org/wiki/Stemming>

[W2] Wikipedia, the free encyclopedia. Full text search [viewed: 2008-10-15]. < http://en.wikipedia.org/wiki/Full_text_search>

[H08] Ruud Hein. *How Search Really Works: "The" Index (2)* [viewed: 2008-11-25]. <http://www.searchenginepeople.com/blog/how-search-really-works-the-index-2.html>

Appendix A. Implementation description

1. Clases:

Class name:	*SearchResult*
Description:	Simple class with four attributes, used to hold search results (output for *ProceedSearch*) and to communicate between methods Example: `ArrayList<SearchResult> rez;`

Attributes:	`public int PlaceID    //Conference ID`
	`public double Weight  //Conference weight`
	`public double TfIdf   //Calculated Tf-Idf`
	`public int Step       //Minimal phrase step`

Methods:	none

2. Methods of class *PositionalIndexCreateEngine:*

Method name:	*IndexKeywords*
Description:	Main method for positional index creation. Creates positional index from publications table. Select publications where yea r=> `yearStart` and year <= `yearend` Index starts from 1 for every publication

Input parameters:	`Connection con,       // Valid connection`
	`Connection con1,      // Another valid connection required for synchronized read/write operations`
	`int yearStart,        // Minimal year value of title`
	`int yearEnd,          // Maximum year value of title`
	`boolean useStemming   // If True we use stemming`

Output parameters:	`void`

3. Methods of class *PositionalIndexSearchEngine:*

Method name:	*ProceedSearch*
Description:	Main method for phrase search. Manages all others methods: *getPhrases, pIndexSearch, mergeResults,and calculateTfIdfb* *Example:* ProceedSearch("*social network*", con, true), con)

Input parameters:	`String query, // Phrase(with '' or "") or keyword query`
	`Connection con,       // Valid connection`
	`boolean useStemming // If True we use stemming`

Output parameters:	`ArrayList<SearchResult>`

Method name:	*getPhrases*
Description:	Returns phrases (`ArrayList<String>`) separated by " or ' from string

Input parameters:	String query, // Phrase(with '' or "") or keyword query boolean useStemming // If True we use stemming
Output parameters:	ArrayList<String>

Method name:	*pIndexSearch*
Description:	Returns search results in stack for the phrase (few terms) or single term
Input parameters:	String[] terms, Connection conn
Output parameters:	Stack<SearchResult>

Method name:	*mergeResults*
Description:	Merges several search results, when we are searching for several terms or phrases
Input parameters:	Stack<SearchResult>[] rez // Output from pIndexSearch()
Output parameters:	ArrayList<SearchResult>

Method name:	*calculateTfIdf*
Description:	Calculates tf-idf for search results
Input parameters:	ArrayList<SearchResult> places, Connection con
Output parameters:	ArrayList<SearchResult>

Example 1. Positional index creation

```
PositionalIndexCreateEngine pc = new PositionalIndexCreateEngine();
pc.IndexKeywords(con, con1,2000,2009, useStemming);
```

Example 1. Positional index searching

```
PositionalIndexSearchEngine ps = new PositionalIndexSearchEngine();
ps.ShowResults(ps.ProceedSearch("social network", con, true), con);
```