The MIA search engine web interface.

Rasmus Ellebæk Kjær Michal Nowacki

January 16, 2003

Contents

Chapter 1

Abstract

This report is to document the authors' participation in the Fall 2002 course 'Algorithms for Web Indexing and Searching' and completing implemantation of a part of a web search engine. The group was responsible for making the web interface to the MIA search engine. Further in this report you can find the details of solutions chosen[1], what problems were encountered[2] and how the cooperation with other groups worked[3]. The demo version of MIA search engine is avaliable at `http://www.daimi.au.dk/~misiek/MIA-final/`.

[1] Chapter Strategy
[2] Chapter What problems did we have
[3] Chapter Cooperation

Chapter 2

Strategy

Now we'll try to present how the process of creation the webinterface was going on. First a list of what decisions we[1] made:

- we decided to write the interface in `PHP`. It seemed a reasonable choice because it is a language/tool that is strictly connected to the http server. Our reasoning was that it should be quite fast (and for a search engine it is important) and besides `PHP` also has a nice library of functions. Other choices considered were: `C` and `Python`.
- we decided to support queries simmilar to Google, ie. boolean and exact phrases. At this stage we decided to skip queries for similar pages[2].
- we agreed on boolean operators to be:
  - `+` for `and`
  - `-` for `not`
  - `' '`[3] for `or`

Having these decisions made we started implementing the interface. But soon it turned out our further work depended very much on the Indexing group[4]. After meeting with the Indexing Group we started to implement our side of the agreed interface.
Below are the details of what the web interface to the MIA search engine is doing.

[1] The Query Group
[2] And as it turned out we skipped it for good
[3] It is a whitespace
[4] See chapter Cooperation

2.1 Boolean queries

Since the Indexing Group is doing the boolean queries as they find document IDs the only thing we do here is parse the query and transform it to the form requested by the designed protocol.

2.2 Querying for exact phrases

Modifiers (+/-) in front of phrases are not supported, though a + is implicitly assumed (modifiers present in the query are ignored). Phrases are handled by requering the words of the phrase to be in the results (like a + in front of every word in the phrase). Then the results are checked for the exact phrase using the position server, and bad results are weeded out. Possibly, more results are requested and weeded, until the required number of results are found.

Initially, searching for phrases was extremely slow (at least 15 seconds for a simple query). The implementor of the server examined the protocol and found the main problem to be the new connection required for each word and page. After changing the protocol to allow for several requests on one connection, this problem was basically solved.

2.3 Presenting results

We present 20 documents[5] per page and allow user to go to the next and previous page with results[6]. Each result is presented in a following manner. We send a request to Page Server for first4 Kb of the document corresponding to the document ID and from that we extract the URL for this page (it is prepended by the page-server, in a <base> tag), pages title[7], page description[8]. And write such html code:

```
<a href=documentURL>Pages title</a><br>
```

[5]If there are that many

[6]If there are such pages. Since the conection with the index server cannot stay alive while changing the pages, every time the user changes the page the whole engine is started again with the same query with the different number of documents to request. At first glance it seem not a very efficient way but the second option we considered was preparing a temporary webpages (files) holding the results. Our motivation for choosing the first solution was that it would take more time to prepare temporary webpages than running the query again

[7]from html header

[8]from html meta tag

```
description<br>
documentURL<a href=page_server>Local copy</a>
```

In case the page has no title or description, we display a message about that instead of these fields. We considered displaying a part of the document with the queried words but the reason we do not do this is time. Consider an example: a query returns 20 pages each 40 Kb and the queried word (for simplicity one) is at the end of each document. First we would have to transfer these 20*40 Kb through TCP network and then process them to find the word. If the words are at the begining the difference between considered solutions shouldn't be that big because we could transfer documents in blocks.

Results are ranked based solely on pagerank. We considered using a more advanced ranking, considering things like the positions of words on pages, the font-size of the words, casing,font-type and so on (there are many possibilities). The basic problem with doing something like this is all we have on individual words are positions, so to get the information we had to load the page and find it ourselves, and this was considered too slow.

Chapter 3

Don't try this at home - What problems we had

The main problems we had was the way `PHP` handles some of its features. Below is a list of some features we didn't manage to get to work properly.

- we wanted to make object oriented code. After a couple of frustrating days we gave up, so we had to stick to not very nice style of global variables.
- keeping TCP connections alive when for a longer time than displaying the page. More detailed: the connections are kept alive but we couldn't use it any more, all file handles are lost.
- the problem which caused much trouble and confusion was reading and writing integers to streams, ie. the endians of integers sent to the `Index Server` and `Position Server`. In the end of the project we actually found some functions in php to help us, but there are no functions to write an integer directly to a socket (nor a byte, although reading a byte (char) is possible).
- The use of multi-dimensional arrays in php gave problems every time we tried it. Basically, the semantics are poorly documented and (from our point of view) not logical.

To sum up the problems: None of us had any knowledge of PHP before the start of the project. In the end, it turned out to be a poor choice. A lot of frustrating hours were used chasing errors, that would have been discovered immediatly if the language was strongly typed. A mispelled variable name (possible in a completely different file) is a hard error to find. A lot more could be said, like problems with library functions or different 'language features'.

Another problem is the output of the compiler in the case of errors (not very informative), and sometimes the compiler seemed to run indefinately, without output, mostly (it seems) because of syntax-errors in the code (like a missing ;). A hard error to find, if you can't remember what lines you changed. Basically, the heading of this chapter says it all. Do not do a project like this in PHP. If we should start all over, we would write a server in another language to which we could send the query and then read the results, having only presentation in PHP.

Chapter 4

Cooperation

At the begining it is neccessary to point out that the cooperation with other groups[1] was crucial since of our part of the project was dependent on their structures.

4.1 Indexing group

Since the indexing group has decided to recieve not only single words and return document IDs for documents containing that word but the whole variaty of boolean queries[2] and return document IDs fulfilling the query almost all of the task of implementing the engine was of our back. Now we were left to design some interface to access the index using the TCP-based protocol for accesing the index designed by the indexing group.
At this point one might think it is kind of one-side cooperation but it's not. The indexing group agreed with our request of remembering the word positions for each document. This database was to make querying for exact phrases a little faster than just 'grep'ing through' documenta for the phrase[3]. Again access to the `position database` is guaranteed by a simple TCP protocol specially designed for this purpose.

4.2 Crawling group

This group on the other hand crucial is because of another reason: they store almost everything, ie. the documents they have fetched during crawling. They were also responsible for translation document ID into real document.

[1]That is: Indexing and Crawling groups
[2]with the restriction that the query cannot contain more than 32 words
[3]See chapter Strategy for details

We had to know how to get such information and again another TCP protocol was designed. But in this case you are right if you say `one-way cooperation`. But the truth is we had nothing to offer to the Crawling group.

Chapter 5

Summary

To sum up the whole work over our part of project we want to point out that:

1. what we wanted was an interface resambling Google(tm)
2. however we do not support searching for `simmilar pages`
3. we do not support delivering a summary for the document fulfilling the query
4. Ranking is done solely on page-rank, and not any other kind of information.
5. the above feature is not supported in order to make the engine give results in google-like times
6. Searching for phrases is too slow.

Working on the project and participating in the course was encouraging and interesting experience, to see the theory in practice.