PyThinSearch: A Simple Web Search Engine

Andri Mirzal
Graduate School of Information Science and Technology,
Hokkaido University, Kita 14 Nishi 9, Kita-Ku, Sapporo 060-0814, Japan
andri@complex.eng.hokudai.ac.jp

Abstract

We describe a simple, functioning web search engine for indexing and searching online documents using python programming language. Python was chosen because it is an elegant language with simple syntax, is easy to learn and debug, and supports main operating systems almost evenly. The remarkable characteristics of this program are an adjustable search function that allows users to rank documents with several combinations of score functions and the focus on anchor text analysis as we provide four additional schemes to calculate scores based on anchor text. We also provide an additional ranking algorithm based on link addition process in network motivated by PageRank and HITS as an experimental tool. This algorithm is the original contribution of this paper.

1. Introduction

Many papers written in the web information retrieval (IR) utilize their own web crawlers to crawl, index and analyze contents, hyperlink texts, and network structure of web pages, and sometimes also search functions to return the sorted relevant pages to users' queries [1], [2]. *Crawler* and *search function* are considered to be the fundamental components of a search engine [3], and each has its own research challenges and problems.

Web crawler, also known as spider or robot, is the component that responsible to fetch pages, parse hyperlinks, manage crawl queue, and index pages' contents. In a more sophisticated form, this component also implements politeness policies (for example obeying robot.txt instructions and not overloading servers with repetitive pages queries [3]—[5]), indexes anchor text [1], [6], decides in advanced which pages are spam [3], [7] or with different URLs but similar contents [7], [8] so that crawler can avoid downloading these pages, and network bandwidth and processing resources can be saved.

The real working crawlers like Mercator [4], [9], Polybot [5], UbiCrawler [10], Googlebot [11], iRobot [7], and IRLbot [3] clearly are much more sophisticated than can be described here. Even though some of them provide good documentations on their designs and implementations [7], [11], the descriptions are still too general to make any reproduction efforts possible.

This is also true for second component, the search function. Starts with basic search facility, content-based scores which only index and search for Boolean matching between content of pages and queries, evolves to link analysis methods, like PageRank [11], HITS [12], and Salsa [13], and then also extends to more advanced methods like anchor text analysis [1], [14]—[18] and user-click behavior [6], [19]. Even though all of these methods are simply about how to find and rank the relevant pages according to the queries, the actual implementations of the algorithms, database designs, optimization issues, and memory management have already become too complex to deal with. This situation becomes even more difficult by the fact that web search engines are big businesses with very competitive nature, so that the designs and implementations of real web search engines remain closed.

This is really a disappointing point in the open source era, where software should become tool to teach people and spread the knowledge without any restriction. Of course we can find chunk of codes that randomly shattered in web pages to construct our own system. And actually this effort can become easier because there are already good open source projects working on this field. Some notable projects are Lemur[1] and Lucene[2]. But due to their complex designs

[1] http://www.lemurproject.org/
[2] http://lucene.apache.org/

978-0-7695-3575-3/09 $25.00 © 2009 IEEE
DOI 10.1109/CISIS.2009.12

and implementations, these projects are still not a good starting point.

Here the importance of open source search engine project that the code and design are easy to understand is emphasized, because if learning is easy and fun the complex details can follow later. Also there are many occasions that we want to do something different than those already have done, like implementing new ranking algorithms, doing stemming on pages' contents and hyperlinks texts, and using new methods in anchor texts analysis. All these require us to alter the existing codes and add new functions or modules to the source code.

To provide software that can satisfy the above mentioned conditions, we propose using search engine program written by Toby Segaran in his book, *Programming Collective Intelligence* [6]. This program is written in python, a free and open source script language that has been very popular recently due to its simple syntax and semantic, comprehensive libraries, and strong support for main operating systems. The program itself comprises of three classes, `crawler`, `searcher` and `nn` (neural network) class[3]. We will only use, improve and add new functionalities to `crawler` and `searcher` class because we are only interested in building search engine to crawl web pages and provide search function to the users, not in training the network with feedback from users (which requires user log dataset). Users who interested in implementing neural network are recommended to consult directly with the book [6] and a paper by Baeza-Yates et. al. [19] in which they used search log information to classify users' queries.

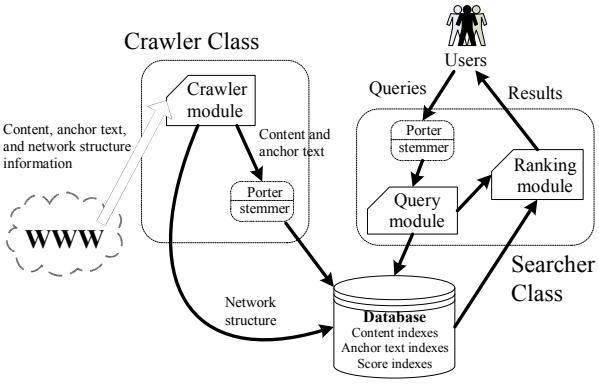

Figure. 1. The system design in top level view.

2. System overview

Figure 1 shows the overall design of the system which mostly inherits its structure from the original system, and figure 2 describes URLs queue management, where crawler downloads URLs from list in RAM, parses hyperlinks on pages, and sends it to disk. If the URLs are unique, they will be added to `urllist` table and then will be sent to the URLs queue in RAM. The following subsections describe the modifications to the original system.

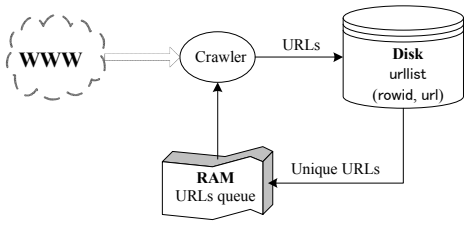

Figure. 2. URLs queue management.

2.1. Improve the reliability of the crawling

In the original code, the crawler will attempt to open a URL in the URLs list (python set) provided by user. If it fails, the crawler will start with the next URL. The problem is even though the URL can be opened, it doesn't mean that the content can be read. Further if the content can be read, sometimes the indexer functions in `crawler` class can fail to input the indexes into corresponding tables in database (see figure 3). In many cases, these errors can cause the program stops working, which is very inconvenient, because in web data mining the ability to continue crawling process by ignoring unreadable pages is an indispensable quality.

We overcome this shortcoming by simply introduce `try-except` statements for opening and reading URLs, and indexing processes which include content, anchor text, and network structure indexing (see figure 3). With this simply approach we were able to crawl web pages for three days without stopping and downloaded 74,243 pages (243 MB) from scholarpedia[4] in June 2008 using notebook with 1.86 GHz Intel processor, 2 GB RAM, and 100 Mbps (download speed: 53 Mbps and upload speed: 5.5 Mbps) Internet connection. Comparing to the original system, where the crawling will end immediately if there is an error in the process, this is a significant improvement.

We also add stop words according to English

[3] http://kiwitobes.com/PCI_Code.zip

[4] http://www.scholarpedia.org

standard stop words[5] to reduce the size of database. But 'computer' term is eliminated from the list because we believe this term is meaningful in search purpose.

2.2. Add optional stemming function

We also add stemming function using porter stemmer algorithm[6] [20] to both classes. Stemming is a process to reduce inflected or derived words into their stems. Stemming process frequently will improve the performance of an IR system, because words with the same stem usually have similar meaning. The stem itself need not to be the same with its morphological root, but a good stemming algorithm should return the same stem word for words that have the same morphological root. For example 'navigational', 'navigation', and 'navigate' have morphological root 'navigate'. If porter stemmer is used, 'navig' will be returned instead of 'navigate' for these words.

As shown in figure 1, the stemming function is located before contents and anchor texts enter the database in `crawler` class and before queries enter query module in `searcher` class. This procedure will ensure that all words in `wordlist` table (see figure 3 (a)) are saved in their stem forms, and all terms in queries will enter the query module also in their stem forms. One important thing to remember is, if crawling process is done with stemming function being activated, searching process must also be done with stemming function being activated.

2.3. Add new score functions

We incorporate some standard score functions that haven't available in the original program. The first function is `bm25score()` that uses Okapi bm25 algorithm [21], an algorithm that calculates the frequency of appearance of query terms in the pages with assumption that each term is independent.

The second function is `calculatelength()` that creates `pagelength` table with `urlid` as the primary key and `length` of pages as the return values. This function scores pages based on their lengths where longer pages have higher scores than the shorter ones. Function that accesses `pagelength` table and returns length scores is `lengthscore()`. Note that this function not only returns pages that contain terms in query, but also pages that being linked by anchor texts which contain terms in query. We use this approach because anchor text is proven to be a good indicator about pages' contents and behaves as a "consensus title" [18].

The third function is `calculatehits()` that calculates authority and hub scores of each page based on query-independent HITS [22 pp. 124—126], an algorithm that computes global authority and hub vector which slightly reduces the influence of link spamming. This function creates two tables, `auth_hits` and `hub_hits` for each `urlid`. Function `authorityscore()` accesses `auth_hits` table and return authority scores. Curious readers can also write a function similar to `authorityscore()` that returns hub scores by only change `auth_hits` with `hub_hits` statement (in real application hub scores are rarely being used to rank relevant pages).

The fourth function is `calculatemyhits()` that calculates authority and hub score of each page using proposed algorithm (see section 4). This function creates two tables, `auth_myhits` and `hub_myhits` for each `urlid`. Function `myauthorityscore()` accesses `auth_myhits` table and return authority scores.

And the last added score function is `anchorscore()`, a function with four optional schemes that returns relevant pages according to the frequency of appearance of query terms in the anchor texts linking to the pages. This is quite a new method and has been proved to be very effective in finding the relevant pages, especially if anchor model (scheme 4, the default scheme), proposed by Fujii, is used. The detail discussion and performance evaluation can be found in the author's paper [17].

2.4. Organize the score functions into query-dependent and query-independent

Some scores are query-dependent, and some scores are query-independent so that can be calculated in advanced and stored in database. Generally, the former is content-based or anchor text-based scores, and the latter is link structure-based scores. However, `lengthscore()` is unique because it is actually a content-based score but its value is query-independent.

The query-dependent functions can be changed into query-independent by passing string argument other than '`qd`' into `query()` function in `crawler` class. The `query()` itself receives two arguments; the first is a query (string) and the second is type of scores, whether it is query-dependent or query-independent. The default value is '`qd`' (query-dependent). See readme.txt

[5] http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words
[6] http://tartarus.org/~martin/PorterStemmer/python.txt

on the source code[7] for more information.

2.5. Add additional indexes to linkword table

Because four additional schemes to calculate scores based on anchor texts [17] are implemented, it is inevitable to create new indexes for linkid and wordid column in linkword table to boost the performance of score functions that call entries on linkid and wordid. By using these indexes, the execution time for linkword related score functions, linktextscore() and anchorscore(), becomes 3-4 times faster.

3. Database design

There are 12 tables in the database which divided into three categories, content, anchor text, and score indexes tables. The shaded tables are the added tables. The content indexes tables are mainly used in content-based score functions, anchor text indexes tables are mainly used in anchor text based score functions, and score indexes tables are the tables that store query-independent scores.

The urllist table stores the crawled URLs in consecutive numbers based on the time they crawled. These numbers, rowids, are the identity numbers for corresponding urls. All other tables that utilize urls refer to rowids instead of directly using urls. The arrows from urllist to entries in others tables indicate that those entries are pointers to urls by using rowids of urllist table.

The same situation also goes to wordlist table, where its rowids are the identity numbers for corresponding words. And wordlocation table utilizes both urllist and wordlist to make a list of location of words in every page. The location itself is simply the order of word's appearance on the page.

The link table stores network structure of crawled web pages. It is the table that link structure ranking algorithms use to calculate pages' scores. The linkword table stores wordids of anchor texts of corresponding links. This table is used in the anchor text analysis and scoring, including linktextscore() and anchorscore() function.

All tables in score indexes store corresponding scores of every page in the collection. When a user inputs query, search function will call appropriate score functions (based on predefined score functions combination) that utilize these tables. The third entries in auth_myhits, hub_myhits, and mypagerank, are constants that derived from our proposed algorithms.

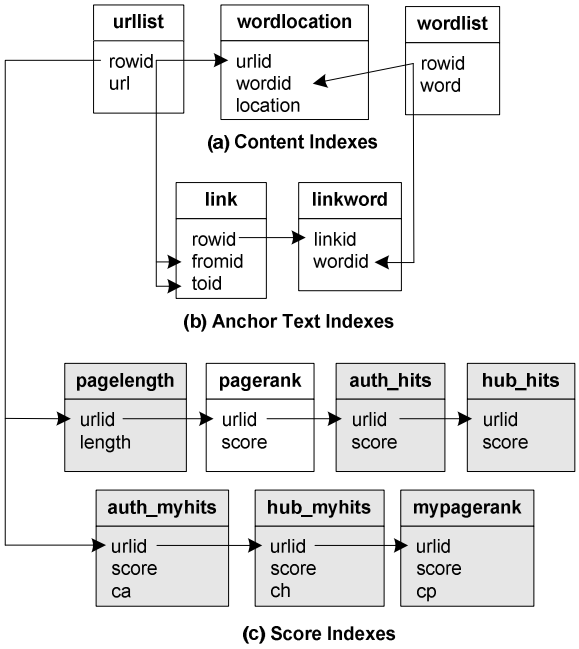

Figure. 3. Database structure.

4. Proposed algorithm

Proposed algorithm will be briefly discussed here. Further discussion can be found in [23], [24]. The development of the algorithm is motivated by the need to analyze trading network by using link structure ranking algorithm to cluster the most similar nodes. It is known that beside its function to determine degree of importance of the nodes, ranking algorithm can also be used to characterize the nodes and group them based on their similarity in characteristic vectors [25]. But before we point out the differences in the process of link addition between WWW network and trading networks, HITS, a ranking algorithm for WWW network, has been used to characterize users in e-commerce activities networks [25].

The process of link addition in network is the key to define ranking algorithms like PageRank [11], HITS [12], and SALSA [13]. By pointing out the differences between trading network and WWW network in this process, we are able to construct a new algorithm for trading network. The difference is: link addition in trading network is the exchange of two different resources between two nodes, on the other hand, in

[7] http://pythinsearch.googlepages.com/

WWW network is simply the creation of new hyperlinks. This process also implies an importance fact: link addition process in trading network is mutual, when node A creates a new link to node B, node B also creates a new link to node A. However, in WWW network the process is not mutual, so when page A creates a new hyperlink to page B, it doesn't necessary that page B also creates a new hyperlink to page A. Another difference, but not related to the construction of the algorithm, is links' weights in trading networks are usually the price/quantity of exchanged goods, whereas in WWW network, the weights are usually only binary, either 0 if there is no link and 1 if there is a link connected two pages. Further, the purpose of link addition in trading network is to maximize the benefit of the transactions. Thus, in the selling side, each node competes to get transactions from other nodes that are lack of the resource it offers, and in buying side, each node competes to get transactions from other nodes that have abundant resource it needs. In WWW network, the purpose of link addition is to get inlinks from popular pages.

In trading network, the resources are usually limited. Each node should be careful in creating and receiving new links. It is more preferable to create new outlinks to other nodes that are lack of resources and receive new inlinks from others that are plenty of resources, because creating outlinks means giving up resources and receiving inlinks means getting the resources.

Proposed algorithm for trading network can be stated with the following definition: *a node becomes more important if being pointed to by others that have many inlinks and points to others that have many outlinks*. By comparing this statement with HITS definition, *good authorities (nodes with many inlinks) are pointed to by good hubs (nodes with many outlinks) and good hubs point to good authorities*, and its mathematical model [22 pp.115], proposed algorithm for trading network can be written as:

$$r(n_i) = \beta \sum_{j \to i} \left(r(n_j) \times c_a(n_j) \right) + (1-\beta) \sum_{i \to j} \left(r(n_j) \times c_h(n_j) \right) \qquad (1)$$

where

$$c_a(n_j) = \frac{\sum n_j \, inlinks}{\sum n_j \, links} \times \left| \sum n_j \, inlinks - \sum n_j \, outlinks \right|^{p_j},$$

$$c_h(n_j) = \frac{\sum n_j \, outlinks}{\sum n_j \, links} \times \left| \sum n_j \, inlinks - \sum n_j \, outlinks \right|^{-p_j},$$

and

$$p_j = \begin{cases} 1 & \text{if } \sum n_j \, inlinks > \sum n_j \, outlinks \\ -1 & \text{if } \sum n_j \, inlinks < \sum n_j \, outlinks \\ 0 & \text{if } \sum n_j \, inlinks = \sum n_j \, outlinks \end{cases}$$

$r(n_i)$ denotes ranking score of node i, $|*|$ denotes absolute value of $*$, $i \to j$ denotes node i links to node j, and $\sum n_j$ inlinks / outlinks / links denotes number of inlinks / outlinks / links node j has. Parameter β ($0.0 \leq \beta \leq 1.0$) is used to determine which type of link is more important. If outllink and inlink are equal sets $\beta = 0.5$, if outlink is more important than inlink sets $\beta < 0.5$, and $\beta > 0.5$ otherwise.

As shown in eq. (1), $r(n_j)$ is weighted with constant $c_a(n_j)$ in the first term of right hand part, where the bigger the number of inlinks and the smaller the number of outlinks, the larger $c_a(n_j)$ becomes. And in the second term, $r(n_j)$ is weighted with constant $c_h(n_j)$, where the bigger the number of outlinks and the smaller the number of inlinks, the larger $c_h(n_j)$ becomes. Thus, the above equation agrees with proposed algorithm definition.

After ranking algorithm for trading network is defined, it can be translated to WWW network. Here we are only interested in creating algorithm based on the most famous one, HITS (actually algorithm based on PageRank is also constructed, but there is still convergence issue, so it hasn't included in this paper yet although the code for it has already been included) and we call it modified HITS (in the code, function that implements it is `calculatemyhits()`).

Modified HITS algorithm is defined by restated HITS definition (shown above) in the style of proposed algorithm definition for trading network, *a node becomes more important if being pointed to by others that have many outlinks (good hub) and points to others that have many inlinks (good authorities)*. This is our modified HITS definition for WWW network, and it can be shown that this statement is actually opposite to proposed algorithm definition for trading network, and can be written as:

$$r(n_i) = \alpha \sum_{j \to i} \left(r(n_j) \times c_h(n_j) \right) + (1-\alpha) \sum_{i \to j} \left(r(n_j) \times c_a(n_j) \right) \qquad (2)$$

Parameter α ($0.0 \leq \alpha \leq 1.0$) is used to determine which type of link is more important. If outllink and inlink are equal sets $\alpha = 0.5$, if outlink is more important than inlink sets $\alpha > 0.5$, and $\alpha < 0.5$ otherwise.

We will rewrite eq. (2) in matrix form for three reasons. First, it will allow us to see the network's property from linear algebra perspective. Second, in

the very large network, the only visible method to compute $r(n_i)$ is by using power method that applied to the adjacency matrix of the network [22 pp.40—41] (power method finds $r(n_i)$ by calculating dominant left-hand eigenvector of network's adjacency matrix). And third, in some programming languages that work on matrix like MATLAB, it is easier to write codes in matrix form.

In matrix form eq. (2) can be rewritten as:

$$\mathbf{r}^{(k+1)T} = \mathbf{r}^{(k)T}\left(\alpha \mathbf{C_h L} + (1-\alpha)\mathbf{C_a L}^T\right) \quad (3)$$

where $\mathbf{L}$ is adjacency matrix of the network, $\mathbf{C_h} = \mathbf{K}^{-1}\mathbf{D_o}^{-1}$, $\mathbf{C_a} = \mathbf{K}\mathbf{D}^{-1}\mathbf{D_i}$, and k = 0, 1, 2, ... is iteration number. Diagonal matrices $\mathbf{D_i}$, $\mathbf{D_o}$ and $\mathbf{D}$ are defined as:

$$\mathbf{D_i} = \mathrm{diag}(\mathbf{d_i}), \mathbf{D_o} = \mathrm{diag}(\mathbf{d_o}), \text{and } \mathbf{D} = \mathbf{D_i} + \mathbf{D_o} \quad (4)$$

and $\mathbf{K}$ is a diagonal matrix with diagonal entries defined as:

$$K_{ii} = \left|\left(\mathbf{D_i} - \mathbf{D_o}\right)_{ii}\right|^{p_i} \quad (5)$$

given $i_i = \sum_j L_{ji}$ is indegree of node i, $o_i = \sum_k L_{ik}$ is outdegree of node i, $\mathbf{d_i} = (i_1, i_2, \ldots, i_N)^T$ is inlink vector, and $\mathbf{d_o} = (o_1, o_2, \ldots, o_N)^T$ is outlink vector of the network.

Because in HITS authority and hub vector have different role (authority scores are usually used to describe the degree of importance of pages, and hub scores are usually used to find portal pages, pages that link to important pages), eq. (3) will be rewritten to separate these vectors. The modified HITS can be defined as:

$$\mathbf{a}^{(k+1)T} = \mathbf{h}^{(k)T}\mathbf{C_h L} \quad (6)$$

for authority vector, and

$$\mathbf{h}^{(k+1)T} = \mathbf{a}^{(k+1)T}\mathbf{C_a L}^T \quad (7)$$

for hub vector.

Table 1 summarizes the algorithms, table 2 and 3 give the approximation of cost and memory requirement respectively. Here $nnz(\mathbf{L})$ is the number of nonzeros in $\mathbf{L}$.

Table 1. Ranking algorithms summary.

Method	Authority	Hub
PageRank	$\mathbf{pr}^{(k+1)T} = \mathbf{pr}^{(k)T}\mathbf{D_o}^{-1}\mathbf{L}$	-
HITS	$\mathbf{a}^{(k+1)T} = \mathbf{h}^{(k)T}\mathbf{L}$	$\mathbf{h}^{(k+1)T} = \mathbf{a}^{(k+1)T}\mathbf{L}^T$
Proposed Algorithm	$\mathbf{a}^{(k+1)T} = \mathbf{h}^{(k)T}\mathbf{C_h L}$	$\mathbf{h}^{(k+1)T} = \mathbf{a}^{(k+1)T}\mathbf{C_a L}^T$

Table 2. Needed costs per iteration.

Method	Multiplication	Addition
PageRank	N	$nnz(\mathbf{L}) + N$
HITS	N	$2nnz(\mathbf{L}) + N$
Proposed Algorithm	$3N$	$2nnz(\mathbf{L}) + N$

Table 3. Memory requirement.

Method	Memory
PageRank	$nnz(\mathbf{L}) + N$ integers
HITS	$nnz(\mathbf{L})$ integers and N doubles
Proposed Algorithm	$nnz(\mathbf{L})$ integers and $3N$ doubles

5. Experiment results

Experiments are conducted to inspect the performance and characteristic of search function part, because this is where our main contributions to the original system lie. *First* convergence rates of PageRank, HITS (authority), and proposed algorithm (authority), are compared by using two datasets, wikipedia[3] (number of pages: 10,431 and links: 46,152) and scholarpedia (number of pages: 74,243 and links: 1,077,781).

As shown in figure 4, PageRank is the fastest to converge followed by proposed algorithm and then HITS. But in figure 5, PageRank is the slowest, followed by HITS and then proposed algorithm. These are actually twisted results because many HITS experiments have shown that HITS requires many fewer iterations than PageRank (less than quarter to reach the same error rate and about twice as long per iteration [22 pp. 126]). So only in the second dataset, our results agree with previous experiments.

The most interesting fact is, for both datasets, proposed algorithm takes fewer iterations than HITS (about half) and faster (about 70%) to reach the same error rate. Unfortunately, these results are not conclusive yet, because we only tried it in the relatively small datasets.

And *second*, by using the same datasets, similarities between ranking vectors are also calculated. Table 4, 5, and 6 show the results in cosine, Spearman correlation coefficient and Kendall's Tau criterion (P is PageRank, H is HITS, M is modified HITS (proposed algorithm), and I is indegree of a page). Indegree is added because

this parameter is the simplest indicator in measuring popularity of a page.

In wikipedia dataset, proposed algorithm is most similar to HITS followed by indegree and then PageRank. And for Spearman and Kendall's Tau criterion, proposed algorithm gives small negative correlations with PageRank. The situation is almost the same to the second dataset, scholarpedia, where proposed algorithm is least similar to PageRank and quite similar to HITS and indegree.

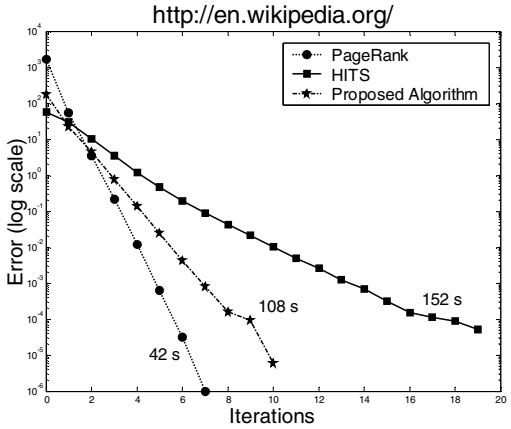

Figure. 4. Convergence rate and time to reach error rate 10^{-4} for wikipedia dataset.

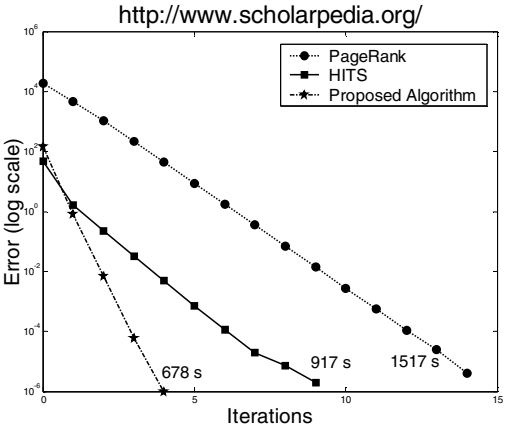

Figure. 5. Convergence rate and time to reach error rate 10^{-5} for scholarpedia dataset.

Spearman and Kendall's Tau criterion calculate similarity of two vectors based on ranking of vector's entries, whereas cosine criterion is based on ranking scores. Thus, in the case of web pages ranking where the scores from link structure ranking algorithms will be combined with other scores (content scores and anchor text scores) to form final scores, the better measurement is cosine criterion because it calculates the similarity of ranking scores, not the ranking itself.

Table 4. Cosine criterion.

Wikipedia dataset

	P	H	M	I
P	1	0.52	0.61	0.50
H	0.52	1	0.84	0.95
M	0.61	0.84	1	0.76
I	0.50	0.95	0.76	1

Scholarpedia dataset

P	1	0.98	0.86	0.98
H	0.98	1	0.89	≈1
M	0.86	0.89	1	0.88
I	0.98	≈1	0.88	1

Table 5. Spearman criterion.

Wikipedia dataset

	P	H	M	I
P	1	0.28	-0.31	0.68
H	0.28	1	0.56	0.73
M	-0.31	0.56	1	0.55
I	0.68	0.73	0.55	1

Scholarpedia dataset

P	1	0.61	-0.21	0.66
H	0.61	1	0.41	0.90
M	-0.21	0.41	1	0.57
I	0.66	0.90	0.57	1

Table 6. Kendall's Tau criterion.

Wikipedia dataset

	P	H	M	I
P	1	0.19	-0.32	0.52
H	0.19	1	0.42	0.58
M	-0.32	0.42	1	0.37
I	0.52	0.58	0.37	1

Scholarpedia dataset

P	1	0.46	-0.22	0.50
H	0.46	1	0.30	0.83
M	-0.22	0.30	1	0.41
I	0.50	0.83	0.41	1

6. Discussion and future researches

The main modifications we did in the original code are in the search function. The crawler part is almost unchanged (even though many additional functions are located in `crawler` class, they are unrelated to crawling process). As noted in the introduction, each of these has already become sophisticated research field which requires expertise in order to improve the system. Readers who have read the original documentation would notice that the crawling method used is breadth first search without politeness policies (for example, obeying robot.txt and controlling access to the servers), spam pages detection, priority URLs queue, and good memory management plan to divide disk and RAM

load for list of next URLs to be downloaded. Without good memory management, all of URLs seen tasks are done by looking urllist table in the disk which is very time consuming. We will address the crawler design problem in the future researches.

As shown in figure 4 and 5, proposed algorithm seems to have interesting properties so that it can beat query-independent HITS in both required iteration number and needed time to reach the same error rates. At this time the explanation we have is, the introduction of constant c_a and c_h makes an authoritative node becomes more authoritative and a "hubby" node becomes more "hubby". Consequently the iterations converge faster. Because these are interesting results, we will also address it in our future researches.

There is no single best way to choose the combination of score functions and their weights. Users are encouraged to experiment with their own databases. But perhaps some guidelines here can be considered. For example if database contains set of pages that linked by very descriptive anchor texts, like documents from online encyclopedia, linktextscore() and anchorscore() can be given relatively strong weights. And because authors can easily create pages with many occurrences of keywords, frequencyscore() and bm25score() will give misleading results in this situation, so their weights must be set smaller compare to more reliable content-based metric, locationscore().

7. References

[1] T.G. Kolda, B.W. Bader, and J.P. Kenny, "Higher-Order Web Link Analysis Using Multilinear Algebra," in *Proc. 5th IEEE International Conference on Data Mining*, 2005, pp. 242—249.

[2] C. Ding, X. He, H. Zha, and H. Simon, "PageRank, HITS, and A Unified Framework for Link Analysis," in *Proc. 25th ACM SIGIR Conference*, 2002, pp. 353—354.

[3] H.T. Lee, D. Leonard, X. Wang, and D. Logulnov, "IRLbot: Scaling to 6 Billion Pages and Beyond," in *Proc. 17th International WWW Conference*, Beijing, 2008, pp. 427—436.

[4] A. Heydon and M. Najork, "Mercator: A Scalable, Extensible Web Crawler," *World Wide Web*, vol. 2, no. 4, 1999, pp. 219—229.

[5] V. Shkapenyuk and T. Suel, "Design and Implementation of a High-Performance Distributed Web Crawler," in *Proc. IEEE ICDE*, 2002, pp. 357—368.

[6] T. Segaran, *Programming Collective Intelligence: Building Smart Web 2.0 Applications*, O'Reilly Media Inc., 2007, pp. 49—52.

[7] R. Cai, J.M. Yang, W. Lai, Y. Wang, and L. Zhang, "IRobot: An Intelligent Crawler for Web Forums," in *Proc. 17th International WWW Conference*, 2008, pp. 447—456.

[8] Z. Bar-Yossef, I. Keidar, and U. Schonfeld, "Do Not Crawl in the DUST: different URLs with similar text," in *Proc. 16th International WWW Conference*, 2007, pp. 111—120.

[9] M. Najork and A. Heydon, "High-Performance Web Crawling," Compaq Systems Research Center, Tech. Report 173, 2001.

[10] P. Boldi, M. Santini, and S. Vigna, "UbiCrawler: A Scalable Fully Distributed Web Crawler," *Software, Practices & Experience*, vol. 34, no. 8, 2004, pp. 711—726.

[11] S. Brin and L. Page, "The Anatomy of a Large-scale Hypertextual Web Search Engine," *Computer Networks and ISDN Systems* vol. 33, 1998, pp. 107—117.

[12] J. Kleinberg, "Authoritative Sources in A Hyperlink Environment," *Journal of the ACM* vol. 46, 1999, pp. 604—632.

[13] R. Lempel and S. Moran, "SALSA: The Stochastic Approach for link-structure analysis," *ACM Trans. Inf. Syst.*, vol. 19, 2001, pp. 131—160.

[14] N. Craswell, D. Hawking, and S. Robertson, "Effective Site Finding using Link Anchor Information," in *Proc. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001, pp. 250—257.

[15] T. Westerveld, W. Kraaij, and D. Hiemstra, "Retrieving Web pages using Content, Links, URLs and Anchors," in *Proc. of the 10th Text Retrieval Conference*, 2001, pp. 663—672.

[16] K. Yang, "Combining Text- and Link-based Retrieval Methods for Web IR," in *Proc. 10th Text Retrieval Conference*, 2001, pp. 609—618.

[17] A. Fujii, "Modeling Anchor Text and Classifying Queries to Enhance Web Document Retrieval," in *Proc. 17th International World Wide Web Conference*, 2008, pp. 337—346.

[18] N. Eiron and K.S. McKurley, "Analysis of Anchor Text for Web Search," in *Proc. ACM SIGIR*, 2003, pp. 459—460.

[19] R. Baeza-Yates, L. Calderon-Benavides, and C. Gonzales-Caro, "The Intention Behind Web Queries," in *Proc. 13th International Conference on String Processing and Information Retrieval*, 2006, pp. 98—109.

[20] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter, "New Models in Probabilistic Information Retrieval," *British Library Research and Development Report*, No. 5587, 1980.

[21] K.S. Jones, S. Walker, and S.E. Robertson, "A Probabilistic Model of Information Retrieval: Development and Comparative Experiments," *Information Processing and Management* vol. 36, 2000, pp. 779—840.

[22] A.N. Langville and C.D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, 2006.

[23] A. Mirzal, "Link Structure Ranking Algorithm for Trading Network," in Proc. *International Conference on Complex, Intelligent and Software Intensive Systems*, 2009.

[24] A. Mirzal, "The Development of Link Structure Ranking Algorithms and Clustering Methods for Networks," Master Thesis, Grad. School of Inf. Sci. and Tech., Hokkaido Univ., Japan, 2008, pp. 9—33.

[25] Y. Kawachi, S. Yoshii, and M. Furukawa, "Labeled Link Analysis for Extracting User Characteristics in E-commerce Activity Network," in *Proc. IEEE/WIC/ACM International Conference on Web Intelligence*, 2006, pp. 73—80.