

Parallel Online Ranking of Web Pages

Y. Ganji Saffar, K. Sheykh Esmaili, M. Ghodsi, H. Abolhassani

Computer Engineering Department

Sharif University of Technology, Tehran, Iran

{ganji,shesmail}@ce.sharif.edu , {ghodsi,abolhassani}@sharif.edu

Abstract—Modern search engines use link structure of the World Wide Web in order to gain better results for ranking the results of users' queries. One of the most popular ranking algorithms which is based on link analysis is HITS. It generates very accurate outputs but because of huge amount of online computations, this algorithm is relatively slow. In this paper we introduce PHITS, a parallelized version of the HITS algorithm that is suitable for working with huge web graphs in a reasonable time. For implementing this algorithm, we use WebGraph framework and we focus on parallelizing access to web graph as the main bottleneck in the HITS algorithm.

I. INTRODUCTION

Search technology is one of the most important reasons for success of the web. The huge amount of information available on the web, its high growth rate, and its unstructured nature, all increase the need for search engines with high performance and accurate results.

One of the major components of each search engine is its ranking algorithm. Traditional Information Retrieval (IR) systems usually use some models like VMS [4] and compute rank of results using content similarity measures between user's query and retrieved documents. But in the context of the web, there are some problems with these approaches. For example, spamming may lead to inefficient ranking. Some methods have been proposed to encounter these problems most of which uses some implicit information which is embedded in the web graph. These methods are known as Link-Analysis based algorithms.

PageRank [5] and HITS (Hyperlink Induced Topic Search) [1] are the most well known algorithms in this category. PageRank, which is used by Google for ranking its results, is an offline and query-independent ranking algorithm. This means that the ranking is independent of the specific queries of users and therefore can be done once and used for all of the upcoming queries. On the other hand, HITS is an online and query-dependent algorithm. Being query dependent makes HITS more precise but it has some disadvantages too. In fact, required online computations for this algorithm is too much and the response time of the search engine after submitting queries by users is not acceptable. To overcome this problem, in this paper we will exploit the parallel processing methods to improve the execution performance of the algorithm.

The rest of this paper is organized as follows. In section II, link-analysis based algorithms in general and HITS as a special case are discussed. At the end of this section, some of the variations and improvements for the HITS algorithm that are suggested in the literature are also described. Implementing the

HITS algorithm and its parallel version, PHITS, are discussed in sections III and IV respectively. Finally, last section of this paper contains conclusion and some ideas for future work in this topic.

II. RANKING ALGORITHMS BASED ON LINK STRUCTURE ANALYSIS

Current popular search engines apply many advanced techniques to improve the user satisfaction and the relevance of the returned results to the given query. One of the most important techniques is to exploit the connectivity information (i.e. link structure) hidden in the web documents to improve the search results.

Nowadays nearly all commercial web search engines use one or more algorithms for analyzing the link structure of the web. The main idea of these methods is using the implicit information entailed from the link structure of the web. For example, we can observe that often the pages that are linked directly to each other have similar contents. As mentioned in the previous section the most well known algorithms for analyzing the link structure of the web graph are the HITS and the PageRank algorithms. The main focus of this paper is on the HITS algorithm.

A. The HITS Algorithm

In this algorithm pages are divided into two categories: *hubs* and *authorities*. Hub pages are those that have good lists of links related to a subject. Authority pages are those that are listed on good hub pages. A good hub page for a subject links to many authoritative pages for that subject. Similarly, a good authority page for a subject is linked by many good hub pages for that subject. Goodness is measured by hub and authority scores. Due to the circular definition, these scores are calculated by iteration.

The steps of detecting hubs and authorities in this algorithm are as follows. First the user's query is sent to a keyword based search engine and some of the pages (e.g. 200 of them) that contain the keywords of the user's query are selected. These pages are called the *root set*. The second step of this algorithm is to expand the root set via adding pages that are directly linked to these pages by using forward or backward links. This augmented set is called the *base set*. Authority and hub values for base set nodes are maintained in two vectors, x and y . The elements of these vectors are initialized to 1 and for computing their final values the following iterative equations

are used:

$$x_i = \sum_{j:(j,i) \in E} y_j \quad (1)$$

$$y_i = \sum_{j:(i,j) \in E} x_j \quad (2)$$

In these equations, $(i, j) \in E$ means that a link from page i to page j is in the web graph.

In [1], it is proved that the sequences x_1, x_2, \dots and y_1, y_2, \dots converge. But the question is, how many iterations of the algorithm are required for these sequences to converge. For answering this question, we ran the algorithm with random data several times until the results became stable. The average result is shown in Fig. 1. As this figure shows, HITS algorithm converges very rapidly.

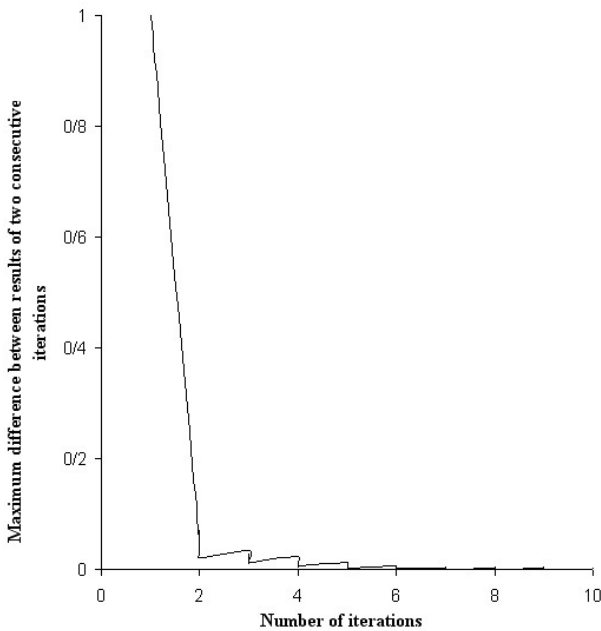


Fig. 1. Convergence of the HITS algorithm

Some search engines have used the HITS algorithm for implementing their ranking subsystems. The first of them is Clever [6], [7], a project sponsored by IBM, which uses a modified version of HITS. Another well known HITS-based search engine is DiscoWeb [8]. But the best and the newest one is TEOMA [9]. Response of this search engine for each query contains three parts: Relevance pages (authority pages), Refines (some suggestion to refine query) and Resources (hub pages). Interested readers can refer to [10] for a comparison between Google [11], Yahoo Search [12], and TEOMA search engines.

B. Variations of the HITS algorithm

After proposing HITS in 1998, a lot of variations and improvement on this algorithm have been suggested in the literature. Most of these researches focus on the use of web graph adjacency matrix.

In [1], it is proved that if A denote the web graph adjacency matrix then x and y vectors will converge to the principal eigenvectors of $A^T A$ and AA^T respectively. Most of the improvements suggested on the HITS algorithm in these researches use this fact as the base. In general these variations fall into two different categories:

- *Modification on the web graph:* main concern of these methods is to deal with cases that we have repeated eigenvector for base matrices. Some types of these algorithms aim at stabilizing HITS (preventing occurring big changes in eigenvector when we modify adjacency matrix a little). Solutions for these problems contain: using path instead of direct link between nodes [13], maintaining user log record for web pages [13], assigning negative weight to some edges in order to decrease their effect [14], creating pseudo hubs and virtual hubs [14], probability and random traveling between pages [15].
- *Modification on computing vectors:* these algorithms work on computing authority and hub vectors and they emphasize on managing eigenvectors for base matrices. The techniques that are usually used for this purpose are: choosing only a special subset of eigenvector space and work with them [15], using projection of eigenvectors on the root set [16], and filtering base root [17], [16].

In our implementation, instead of working with adjacency matrices, we use adjacency lists.

III. IMPLEMENTATION OF THE HITS ALGORITHM

For implementing the HITS algorithm, we used several tools and frameworks which are introduced in the following sections.

The WebGraph Framework

Studying web graphs is often difficult due to their large sizes. The WebGraph framework is a suite of codes, algorithms and tools that exploits the inner redundancies of the web to store a web graph in a limited space of memory. This framework has a simple API that makes it easy to manipulate large web graphs. It is an open source package and is publicly available in [18].

Datasets

In our study, we used the datasets that are publicly available at [19]. These datasets contain data that are gathered by a number of crawlers and are compressed and encoded by the WebGraph compression algorithm [2]. Each dataset is composed of several files. The most important files are:

- A file which contains the compressed graph, and the relative property file which contains metadata about the graph.
- A file which contains the transposed version of the graph, and the relative property file which contains metadata about the transposed graph.
- An offsets file, which is used for random access to nodes.
- A URLs file, in which the i -th line contains the URL of the i -th node.

The following table summarizes the properties of the datasets that we used during our study:

Crawled domain	Crawling year	Nodes	Links
.uk	2002	18520486	298113762
.eu.int	2005	862664	19235140
.it	2004	41291594	1150725436

The pseudo code of our implementation, which is adapted from [1], is shown in Fig. 2. As it is shown in this pseudo code, there are a number of input parameters that effect the results of the algorithm. These parameters are:

- t : The size of the root set.
- d : Maximum number of pages that have a link to one of the pages in the root set and can be added to the base set. There are many popular pages that have incoming links from many other pages on the web. This parameter will prevent the size of the base set from becoming rapidly large.
- k : Number of the iterations that must be done before sending the result to users. As shown in Fig. 1, typically 5-10 iterations are sufficient.

By tuning these parameters, we can manage the trade off between the accuracy of the algorithm and its performance. That means by choosing larger values for these parameters we can have more accurate results but the time needed for doing the computations will be larger.

IV. PARALLELIZING THE HITS ALGORITHM

In summary, the HITS algorithm is composed of these steps: 1) Creating the *root set*. 2) Creating the *base set* by expanding the root set. 3) Performing iterations. The most time consuming part of the algorithm is the third step, i.e. performing the iterations. It is because that during the execution of this part, the processor must have random access to huge compressed graphs which are located in a large portion of the memory, to find incoming and outgoing links of the base set pages. Having random access to large portions of the memory decreases the cache hit rate and therefore execution performance reduces significantly. For dealing with this problem, instead of accessing the huge web graph each time the list of successors of a node is needed, we find the successors and predecessors of each node once and save them in two adjacency lists: one for successors of nodes and the other for predecessors. Although having a little memory overhead, this technique increases the execution performance significantly. Using this technique when working with the crawl of *.uk* domain, which is a relatively large graph, the average response time decreased from 158.4 to 18.2 milliseconds, i.e. it became approximately 8.7 times faster.

So we modified the HITS algorithm as follows: 1) Create the *root set*. 2) Create the *base set* by expanding the root set. 3) Create the adjacency lists. 4) Perform iterations by only accessing the adjacency lists. According to our statistical analysis, after implementing this modification, only about 10 percent of the execution time of the algorithm is elapsed during

the fourth step. This means that the bottleneck of the algorithm is now changed from the last step to the first three steps. The reason is clear: during these steps we need to work with huge graphs that are spanned along a large portion of the memory. Parallelizing steps one and two does not bring any significant performance improvements, so we have focused on parallelizing the third step.

The pseudo code of our algorithm is shown in Fig. 3. In the next section the proposed architecture for implementing the algorithm is discussed.

A. Proposed Architecture

The proposed architecture is showed in Fig. 4. The main part of this architecture is the *server component*, which receives user's query and after calculating the result, sends back it to him. If only one processing node is available, the server component must do all the calculations individually. But if *cooperator* nodes are available, they can register themselves on the server and the server delegates some parts of the calculations to them.

When a cooperator registers itself on the server, a *delegate* thread on the server is being started for handling the communication with the cooperator. Each cooperator must have a copy of the dataset on its hard disk. After the registration of the cooperators on the server, the server component will order the cooperator to load the web graph or the transposed version of it on its memory. In our implementation, cooperators with even ranks are ordered to load the web graph and cooperators with odd ranks are ordered to load the transposed version of it. Because of the huge size of the two graphs, this strategy (loading one of the graphs instead of loading both of them on each node) lets us to use cooperators that are not very resource rich. Therefore even nodes construct AL and odd nodes construct AL_T . Size of these two lists is equal to the number of pages that are in the base set. In AL , each element of the list contains two data, the ID of one of the pages which are in the base set and a pointer to the head of a linked list in which the IDs of pages that this page has links to (outgoing links) are saved. On the other hand, in AL_T , each element contains the ID of one of the pages which are in the base set and a pointer to the head of a linked list in which the IDs of pages that have links to this page (incoming links) are saved.

On receiving queries from users, the server component creates the root and base sets individually. The reason for not doing these steps in parallel is that after implementing the algorithm we found that because of the communication overhead, parallelizing these steps not only doesn't increase the execution performance, but also decreases it. The server component then commands each cooperator which part of the adjacency lists is its responsibility to calculate. Then it does its share of the calculations and waits for all of the cooperators to finish their jobs. After finishing its job, each cooperator sends the result to its correspondent delegate thread, and the thread will put the result in the server memory.

When all of the cooperators finished their jobs, the server starts the iterations and after performing them and reaching

```

HITS( $q, e, t, d, k, c$ )
   $q$ : a query string
   $e$ : a text-based search engine
   $t, d$ : natural numbers
   $R$  = top  $t$  results of  $e$  on  $q$ .
   $S = R$ 
  For each page  $p \in R$ 
    Add to  $S$  all of the pages that  $p$  points to.
    Add to  $S$  at most  $d$  pages that point to  $p$ .
   $\forall i, x_i = 1$ 
   $\forall i, y_i = 1$ 
  For  $i = 1$  to  $k$ 
    Apply operation 1 to  $(x_{i-1}, y_{i-1})$ , obtaining new x-weights  $x'_i$ .
    Apply operation 2 to  $(x'_i, y_{i-1})$ , obtaining new y-weights  $y'_i$ .
    Normalize  $x'_i$ , obtaining  $x_i$ .
    Normalize  $y'_i$ , obtaining  $y_i$ .
  Report the pages with the  $c$  largest coordinates in  $x_k$  as authorities.
  Report the pages with the  $c$  largest coordinates in  $y_k$  as hubs.

```

Fig. 2. The HITS Algorithm (adapted from [1])

```

PHITS( $q, e, t, d, k, c$ )
   $q$ : a query string
   $e$ : a text-based search engine
   $t, d$ : natural numbers
   $R$  = top  $t$  results of  $e$  on  $q$ .
   $S = R$ 
  For each page  $p \in R$ 
    Add to  $S$  all of the pages that  $p$  points to.
    Add to  $S$  at most  $d$  pages that point to  $p$ .
  Create  $AL$ , adjacency list of the web graph, in parallel.
  // in this list, each page of the base set has a linked list of pages that this page has links to.
  Create  $AL_T$ , adjacency list of the pages which are in the base set, in parallel
  // in this list, each page of the base set has a linked list of pages that have links to this page.
   $\forall i, x_i = 1$ 
   $\forall i, y_i = 1$ 
  For  $i = 1$  to  $k$ 
    Apply operation 1 to  $(x_{i-1}, y_{i-1})$ , obtaining new x-weights  $x'_i$ .
    //  $AL_T$  is used for finding pages that have link to each page of the base set.
    Apply operation 2 to  $(x'_i, y_{i-1})$ , obtaining new y-weights  $y'_i$ .
    //  $AL$  is used for finding pages that each page in the base set has links to.
    Normalize  $x'_i$ , obtaining  $x_i$ .
    Normalize  $y'_i$ , obtaining  $y_i$ .
  Report the pages with the  $c$  largest coordinates in  $x_k$  as authorities.
  Report the pages with the  $c$  largest coordinates in  $y_k$  as hubs.

```

Fig. 3. The PHITS (Parallel HITS) Algorithm

to a convergence in the values, it sends the ranked hubs and authorities back to the user. Because of the separation between the calculation component of the server and the cooperators,

there must be a way for the server component to detect that all of the cooperators have finished their jobs. For doing this, when the calculation for responding to a new query is started,

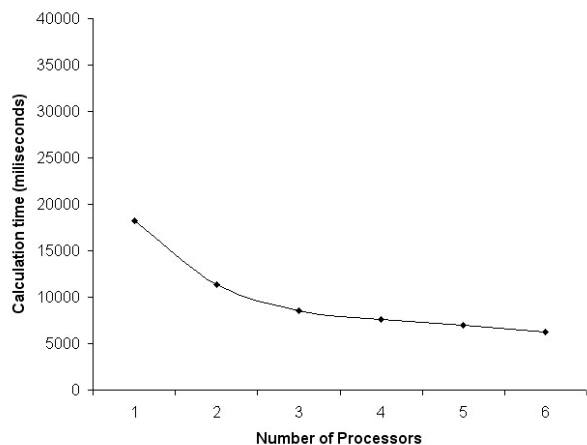


Fig. 5. Calculation time of the PHITS algorithm for $t=200$, $d=50$, $k=20$

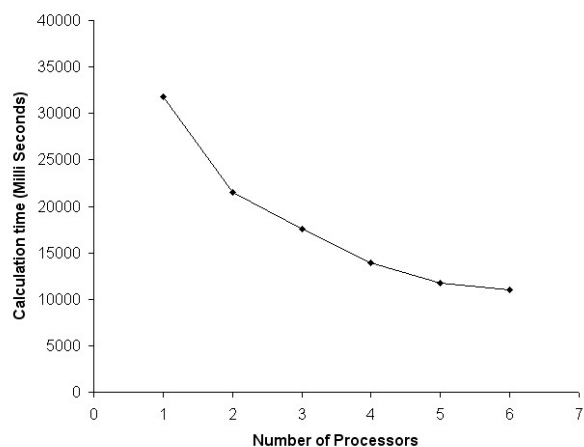


Fig. 7. Calculation time of the PHITS algorithm for $t=500$, $d=50$, $k=10$

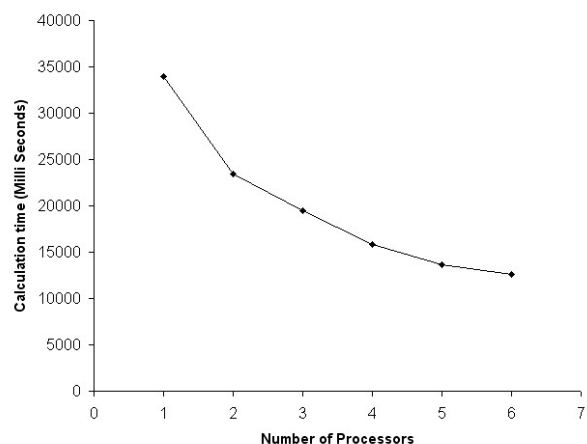


Fig. 6. Calculation time of the PHITS algorithm for $t=500$, $d=50$, $k=20$

the number of participating cooperators is saved somewhere on the server memory. Whenever the job of a cooperator finishes, its correspondent delegate thread decreases this number. It must also be mentioned that the access to this place of memory is done at a critical section, which means only one thread at a time can access this part of the memory. Now for detecting whether all of the cooperators have done their jobs, the server component can check in a loop whether the number is reached to zero or not.

In figures 5 to 7, the result of running our implementation on crawl of *.uk* domain, which contains 18520486 nodes, is shown. In this study we used a Pentium IV 2.4 GHz with 1GB of RAM as our main server and several Pentium IV 2.4 GHz with 512 MB of RAM as cooperators. The hub which was used for connecting these nodes was a 1Gbps hub. The followings can be deduced from these figures:

- 1) The calculation time decreases when new cooperators are used to do the calculations. But using more than four nodes for doing calculations on this dataset doesn't improve the calculation time significantly and is not cost

effective. Current Google index contains more than 8G pages, which is approximately 440 times bigger than *.uk* data set that we used. It is clear that when working with the complete web graph, which has more than 8G pages, using more nodes is cost effective and the speed up will be better.

- 2) As can be seen in figures 6 and 7, decreasing the number of iterations from 20 to 10, doesn't affect the calculation time significantly. This shows that the main bottleneck of the algorithm is in creating the adjacency lists and not in performing the iterations.

V. CONCLUSIONS AND FUTURE WORK

In this paper we used parallel processing methods for improving the efficiency of accessing compressed web graphs. After a brief introduction to the HITS algorithm, we used some techniques to improve its performance in the serial implementations. Then we proposed an architecture for parallelizing this algorithm. In the last section we discussed the results of the execution of our parallel algorithm on a sample dataset and showed the improvement gain when parallelizing the algorithm.

In our proposed architecture, if one of the cooperators fail or have a long latency in sending results to the server, the total calculation will be failed or slowed significantly. For dealing with this problem, in our future work, we intend to add replicas to the architecture so that when a cooperator fails or have a long response latency, one of its replicas does its job. In fact instead of calculating each part of the adjacency lists with one cooperator, several replicated cooperators can do the calculations in parallel and the one which finishes first, will send the result to the corresponding delegate thread in the server.

Although we used adjacency lists to compute x and y vectors, as discussed in the literature, these vectors can be calculated by finding the eigenvectors of the AA^T and $A^T A$ matrices. It is intended to implement this method too and compare the results to the method discussed in the paper.

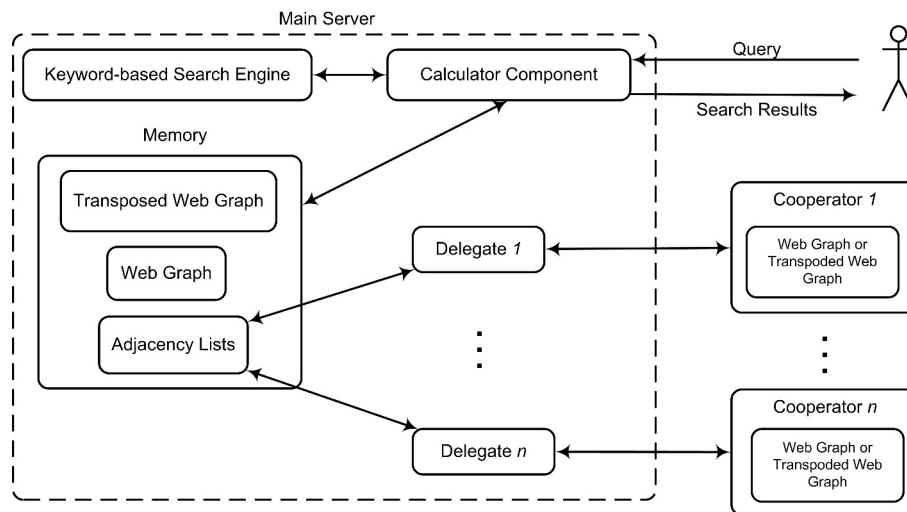


Fig. 4. An Architecture for Implementing the PHITS Algorithm

REFERENCES

- [1] J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, Sep 1999.
- [2] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. of the Thirteenth International World Wide Web Conference*, Manhattan, USA, 2004, pp. 595–601.
- [3] P. Boldi and et al., "The WebGraph framework II: Codes for the world wide web," in *Data Compression Conference, 2004. Proceedings*, Snowbird, Utah, United States of America, 2004.
- [4] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.
- [5] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proc. 7th International WWW Conference*, 1998.
- [6] S. Chakrabarti, B. Dom, P. Raghavan, and S. Rajagopalan, "Automatic resource compilation by analyzing hyperlink structure and associated text," in *WWW7 / Computer Networks*, 1998.
- [7] Clever Project Home Page. [Online]. Available: <http://www.almaden.ibm.com/cs/k53/clever.html/>
- [8] B. D. Davison, A. Gerasoulis, K. Kleisouris, Y. Lu, H. ju Seo, WeiWang, and B. Wu, "Discoweb: Applying link analysis to web search," in *proceedings of the Eighth International World Wide Web*, 2000.
- [9] Teoma Search Engine. [Online]. Available: <http://www.teoma.com/>
- [10] The best search engines. [Online]. Available: <http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/SearchEngines.html>
- [11] Google Search Engine. [Online]. Available: <http://www.google.com/>
- [12] Yahoo Search Engine. [Online]. Available: <http://www.search.yahoo.com/>
- [13] M. J. C., R. G., and S. F., "Modifications of kleinbergs hits algorithm using matrix exponentiation and web log records," in *Proc. of SIGIR01*, New Orleans, September 2001, pp. 444–445.
- [14] R. Lempel and S. Moran, "Introducing regulated bias into co-citation ranking schemes on the web," *Proceedings of the American Society for Information Science and Technology*, vol. 39, no. 1, pp. 425–435, May 2002.
- [15] A. Ng, A. Zheng, and M. Jordan, "Stable algorithms for link analysis," in *Proc. 24th Annual Intl. ACM SIGIR Conference*, 2001.
- [16] S. Nomura, S. Oyama, T. Hayamizu, and T. Ishida, "Analysis and improvement of hits algorithm for detecting web communities," in *Systems and Computers in Japan*, 2004.
- [17] R. Lempel and S. Moran, "Salsa: The stochastic approach for link-structure analysis," *ACM Transactions on Information Systems*, vol. 19, no. 2, pp. 131–160, 2001.
- [18] WebGraph. [Online]. Available: <http://webgraph.dsi.unimi.it/>
- [19] Laboratory for web algorithms. [Online]. Available: <http://law.dsi.unimi.it/>