그래프 탐색 기법을 이용한 효율적인 웹 크롤링 방법들
(Effective Web Crawling Orderings from Graph Search Techniques)

김 진 일 † 권 유 진 †† 김 진 욱 ††† 김 성 렬 †††† 박 근 수 †††††
(Jinil Kim) (YooJin Kwon) (Jin Wook Kim) (Sung-Ryul Kim) (Kunsoo Park)

요 약 웹 크롤러는 웹에서 링크를 따라다니며 웹 페이지들을 자동으로 다운로드하는 프로그램으로 주로 웹 환경을 연구하거나 검색 엔진을 만들기 위해 사용된다. 기존의 연구들에서는 웹 크롤러가 인기 있는 웹 페이지들을 먼저 크롤링 할 수 있도록 몇 가지 방법들이 제안되었으나 그래프 이론 분야에서 연구되어 온 몇몇 그래프 탐색 기법들은 아직 웹 크롤링 방법으로 고려되지 않았다. 이 논문에서는 잘 알려진 너비 우선 탐색, 깊이 우선 탐색 뿐 아니라 사전식 너비 우선 탐색, 사전식 깊이 우선 탐색 및 최대 크기 탐색을 웹 크롤링 방법으로 고려하여 이 중에서 선형적인 시간복잡도를 가지면서도 인기 있는 웹 페이지를 효율적으로 수집할 수 있는 웹 크롤링 방법을 찾는다. 특히 선형 구현이 단순하지 않은 최대 크기 탐색과 사전식 너비 우선 탐색에 대해서는 분할 정제 방법을 이용한 선형 시간 웹 크롤링 방법을 제시한다. 실험 결과는 최대 크기 탐색이 다른 그래프 탐색 방법에 비해 시간 복잡도 및 크롤링 된 페이지들의 질에 있어서 바람직한 성질을 가짐을 보여준다.

키워드 : 웹 크롤러, 검색 엔진, 그래프 탐색, PageRank

Abstract Web crawlers are fundamental programs which iteratively download web pages by following links of web pages starting from a small set of initial URLs. Previously several web crawling orderings have been proposed to crawl popular web pages in preference to other pages, but some graph search techniques whose characteristics and efficient implementations had been studied in graph theory community have not been applied yet for web crawling orderings. In this paper we consider various graph search techniques including *lexicographic breadth-first search, lexicographic depth-first search* and *maximum cardinality search* as well as well-known breadth-first search and depth-first search, and then choose effective web crawling orderings which have linear time complexity and crawl popular pages early. Especially, for *maximum cardinality search* and *lexicographic breadth-first search* whose implementations are non-trivial, we propose *linear-time* web crawling orderings by applying the partition refinement method. Experimental results show that *maximum cardinality search* has desirable properties in both time complexity and the quality of crawled pages.

Key words : web crawler, search engine, graph search, PageRank

· 본 연구는 기초기술연구회의 NAP 과제 지원으로 수행되었음
이 연구를 위해 서울대학교 컴퓨터 연구소에서 연구장비를 지원하고 공간을 제공함
이 논문은 2008학년도 건국대학교의 지원에 의하여 연구되었음

† 학생회원 : 서울대학교 컴퓨터공학부
jikim@theory.snu.ac.kr
†† 정 회 원 : IBM Korea Lab Ubiquitous Computing Lab S/W Engineer
kwonyjin@kr.ibm.com
††† 정 회 원 : 인하대학교 컴퓨터정보공학부 교수
gnugi@inha.ac.kr
†††† 종신회원 : 건국대학교 인터넷미디어공학부 교수
kimsr@konkuk.ac.kr
††††† 종신회원 : 서울대학교 컴퓨터공학부 교수
kpark@theory.snu.ac.kr

논문접수 : 2009년 5월 25일
심사완료 : 2009년 10월 15일

1. Introduction

Web crawlers are fundamental programs which iteratively download web pages by following links of web pages starting from a small set of initial URLs. They are mainly used to build web search engines or to study the Web structure. Every web crawler should determine the order of URLs to visit based on the information it has already obtained. It is desirable that they download popular web pages in preference to unpopular pages since no web crawler can crawl all the pages on the Web within limited time, space, and money [1-6].

The Web can be understood as a directed graph, where each web page is denoted by the corresponding vertex identified by a URL, and each link is denoted by the edge from the source vertex to the target vertex. As every URL has a unique domain name of the site as a prefix, we can represent each site by a virtual *super-vertex* which is the set of vertices in the site. Then, an edge is either an *internal edge* which connects vertices within the same *super-vertex* or an *external edge* which connects vertices across different *super-vertices*. They are consistent with the usual concepts of *internal link* and *external link*.

However, graph search theory has not been seriously considered in previous studies of web crawling ordering. For example, *back-link count* [1], which is useful but was implemented inefficiently, can be implemented in linear time by *maximum cardinality search* [6-8] as we show in this paper. Thus, some theoretical background would give ideas to improve web crawling efficiency as well as the insight to web crawling ordering.

In this paper we examine various graph search techniques and distinguish a useful one which visits popular pages early. We consider a wide range of traditional graph search techniques [7,8] including *lexicographic breadth-first search, lexicographic depth-first search, maximal neighborhood search, maximum cardinality search* as well as well-known *breadth-first search* and *depth-first search.* For *maximum cardinality search* and *lexicographic breadth-first search* whose implementations are non-trivial, we propose *linear-time* web crawling orderings by applying the partition refinement method [9].

We categorize links into three types and apply different strategies for each type: *external links, initial links*, and *non-initial links*. We define *initial links* as internal links of the sites of initial URLs, and *non-initial links* as internal links excluding *initial links*. Usually *external links* are more meaningful than *internal links* since they strongly contribute to the *popularity* of the target URLs. And *initial links* are important because they are contained in directory sites which have many links pointing to popular web pages. Hence, we determine web crawling ordering by *external links* and *initial links*, and we use *non-initial links* only to find new visitable URLs.

We implemented web crawlers using various graph search techniques and compared them by cumulative PageRank' [10] and the number of sites they crawled, which reflect the quality of the crawled web pages and the range of the crawling, respectively. Experimental results show that *maximum cardinality search* has desirable properties in both time complexity and the quality of crawled pages.

Throughout this paper, we assume that the Web is a static graph. Of course, it is not the case for the real situation where pages and links are added and removed every second. However, it greatly simplifies our discussion and sometimes it is more appropriate for one-time crawling or focused crawling.

The next section outlines related work in the subject of web crawlers and page importance metric. Section 3 explains backgrounds about graph search techniques. Section 4 introduces criteria for web crawling ordering and proposes web crawling algorithms based on the partition refinement method. Section 5 evaluates them by experiments and we conclude in Section 6.

2. Related work

2.1 Web crawling ordering

Several URL orderings for web crawlers are proposed so far. Cho et al. [1] conducted the first study on the web crawling ordering. They claimed that PageRank ordering crawls popular web pages better than *breadth-first search* ordering or *backlink-*

count. However, their experiments were done only in small scale data and it is hard to apply PageRank ordering to web crawlers directly because of its computation cost. Najork and Wiener [2] claimed that the *breadth-first search* is sufficient to collect high-quality pages. They actually crawled 328 million pages, but did not compare it with other ordering strategies. Abiteboul et al. [3] proposed an adaptive on-line PageRank computation method (OPIC) and web crawling ordering based on it. They defined the concept of *cash* which is distributed when crawling a page. Since they assumed that the Web is a dynamic graph, popular pages can be crawled many times. As a result, it is difficult to compare their work to ours. Boldi et al. [4] investigated several web crawling orderings and concluded that the set of pages crawled by the ordering which increases cumulative PageRank fast do not represent the Web properly. Baeza-Yates et al. [5] compared various web crawling strategies by PageRank and Kendall's τ. They recommended the OPIC ordering and the larger-site-first ordering for web crawlers.

Cho and Schonfeld [11] proposed *RankMass* ordering which guarantees lower bound of the cumulative PageRank of crawled web pages. It crawls the pages with highest lower bound of PageRank first. Except providing the guarantee, it is similar to the OPIC ordering of Abiteboul's in that each page has a temporary value which is distributed when crawling the page. Even though they also proposed a batch URL ordering and reported the running time of their orderings, it is not clear that their orderings can be implemented in linear time.

2.2 Page Importance Metric: PageRank

Page et al. [10,12] proposed famous PageRank which is a page quality measure based on the link structure. PageRank measures the popularity of a page regardless of query words. Intuitively, a page has high PageRank if many pages with high PageRank have links pointing to it, and the pages pointing to it do not have too many outgoing links pointing to other pages.

PageRank of a page is expressed mathematically as follows. Suppose there are N pages on the Web and let d be a constant[1] such that $0 < d < 1$. For any web page p_i, let $R(p_i)$ be PageRank of p_i, $C(p_i)$ be the number of links in p_i, and $I(p_i)$ be the set of pages which have links pointing to p_i. Then, PageRank satisfies the following equation.

$$R(p_i) = \frac{1-d}{N} + d \cdot \sum_{p_j \in I(p_i)} \frac{R(p_j)}{C(p_j)}$$

PageRank of a page can be interpreted in the random surfer model. Suppose a random walker who wanders on the Web by following links or by visiting random pages. At some step, if he is visiting page p_i, he will jump to a random page with probability $1-d$ or follow any link of p with probability d at the next step. After long walks, he may visit page p_i with probability $R(p_i)$ which is PageRank of p_i.

3. Background

In this section, we explain several graph search algorithms and their relationships. Corneil and Krueger [7,8] categorized graph search algorithms by the properties of vertex ordering and classified them into six categories from general to specific: *generic search*, *breadth-first search* (BFS), *lexicographic breadth-first search* (LexBFS), *depth-first search* (DFS), *lexicographic depth-first search* (LexDFS), and *maximal neighborhood search* (MNS). With the addition of maximum cardinality search (MCS), their relationships are shown in Figure 1.

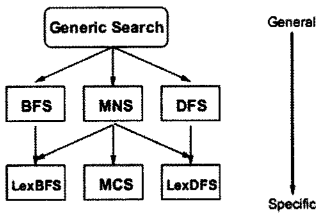


Figure 1 The relationship of graph search techniques

Generic Search - Generic Search is the most general graph search algorithm. It visits *any* vertex v from S which is the set of visitable vertices, and adds new neighbors of v to S if they haven't been visited yet. It repeats the same process until S is empty.

1) The value of d is usually chosen in the range $0.85 < d < 0.9$.

Breadth-First Search (BFS) - BFS is a well-known graph search algorithm which keeps the visitable vertices in a queue instead of the set in Generic Search. When it visits a vertex, it pushes new visitable vertices on the queue and pops the next vertex from the top of the queue. Then, the next vertex is one of the neighbors to the earliest visited vertex.

Lexicographic Breadth-First Search (LexBFS) - LexBFS is a special case of BFS. It keeps an ordered list of vertices called *label* for each unvisited vertex. When visiting the i-th vertex v, it appends $(n-i)$ to the labels of v's neighbors, where n is the total number of vertices in the graph. It compares labels of unvisited vertices one-by-one and selects a vertex with lexicographically largest label as the next vertex. Then, the next vertex is adjacent to as many earliest vertices as possible. When comparing only the first value of labels, LexBFS becomes BFS.

Depth-First Search (DFS) - DFS is also a well-known graph search algorithm which keeps visitable vertices in a stack. Then, the next vertex is one of the neighbors to the latest visited vertex whose neighbors are not visited completely yet.

Lexicographic Depth-First Search (LexDFS) - LexDFS is similar to LexBFS. They both construct labels for all vertices and traverse in lexicographical order. However, when visiting the i-th vertex v, LexDFS prepends i to labels of v's neighbor vertices. Then, the next vertex is adjacent to as many vertices chosen recently as possible.

Maximal Neighborhood Search (MNS) - Let the *visited neighborhood* of a vertex be the set of visited vertices which have edges pointing to it. MNS selects a vertex whose visited neighborhood is maximal.[2] LexDFS and MNS are introduced by Corneil and Krueger [7, 8] to categorize graph search algorithms completely.

Maximum Cardinality Search (MCS) - MCS [6,13] selects a vertex with the maximum cardinality of visited neighborhood as the next vertex. MCS is exactly the same with *back-link count* [1] if we treat *external links* and *internal links* equally.

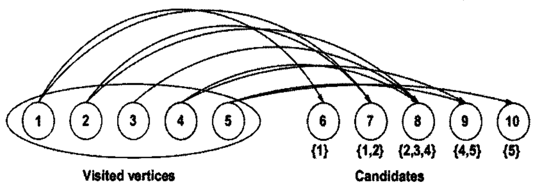


Figure 2 An example of graph search: 1, 2, 3, 4 and 5 are already visited in order and it's time to select the next vertex among vertices 6, 7, 8, 9 and 10.

Example - Consider an example shown in Figure 2. A graph search algorithm already traversed vertices 1, 2, 3, 4 and 5 in order. It now determines to visit one of vertices 6, 7, 8, 9 and 10. Generic Search chooses any vertex from candidates 6, 7, 8, 9 and 10. BFS selects vertex 6 or 7 which gets a link from vertex 1, the earliest visited vertex. LexBFS selects vertex 7 whose label (10-1, 10-2) = (9, 8) is the largest in lexicographical order. Likewise, DFS selects vertex 9 or 10 which gets a link from vertex 5, the latest visited vertex, and LexDFS selects vertex 9 which has the lexicographically largest label (5, 4). MNS selects one of the vertices 7, 8, and 9, whose visited

Table 1 The next vertex to visit for each graph search technique in the example of Figure 2.

Algorithm	Candidate	Justification for the selection process
Generic	6,7,8,9,10	Any neighbor of visited vertices
BFS	6,7	A neighbor of the earliest visited vertex 1
LexBFS	7	A vertex with the largest label (9, 8)
DFS	9,10	A neighbor of the latest visited vertex 5
LexDFS	9	A vertex with the largest label (5, 4)
MNS	7,8,9	Maximal sets {1,2},{2,3,4},{4,5}
MCS	8	*max* (|{1,2}|,|{2,3,4}|,|{4,5}|)

2) A set is *maximal* if it is not a subset of any other set.

neighborhood is maximal from the visited set. Lastly, MCS picks up vertex 8, which has the maximum cardinality of the visited neighborhood. Table 1 shows an overall description.

4. Web crawling algorithms

4.1 Criteria for web crawling orderings

We evaluate web crawling orderings by speed, quality, and the range of crawled pages. Specifically, we use the following three criteria.

1. Time Complexity should be linear (i.e., O(|V|+|E|)). Web crawlers should be sufficiently scalable since the entire Web is enormous. Therefore, we do not consider graph search algorithms which require more than linear time.
2. Web pages with high PageRank should be preferred in ordering.
   Although the page popularity is subjective, we can estimate it by measuring PageRank. Hence it is desirable to crawl pages with high PageRank early.
3. The range of crawling should be spread out widely on the Web.
   To measure the range of crawling, we focus on the number of sites crawled as compared to the amount of crawled web pages. Crawling a few sites in concentration may causes ethical problems and performance degradation.

The above criteria help us to rule out several unpromising orderings. DFS and LexDFS3) do not give preference to popular pages. We need not consider MNS because we can make a specific ordering of MNS by MCS in linear time. Therefore, the next two subsections are dedicated to propose web crawling algorithms using LexBFS and MCS.

4.2 Web Crawling Ordering using LexBFS

LexBFS can be implemented in linear time by the partition refinement method which was proposed by Habib et al. [9]. It does not maintain labels since they make LexBFS conceptually easy but inefficient.

The partition refinement method works as follows. First of all, it binds the URLs with the same labels into a list called class. All classes are connected by a doubly-linked list maintained in decreasing order of labels. When a web page is crawled, each class is divided into two classes: one containing the URLs which have incoming links from the crawled page, and the other containing the remaining URLs. New URLs extracted from the crawled page are added as a new class to the end of the class list. This operation can be done in time proportional to the number of outgoing links of the crawled page, hence the overall time complexity is O(|V|+|E|).

In the above method we distinguish different types of links. We use only *external links* and *initial links* for the purpose of moving URLs, i.e., *non-initial links* do not contribute to the priority of URLs but they are used only to add new URLs to crawl. We adopt such a policy because *external links* tend to have meaningful information in comparison with *internal links*. It improves the quality of ordering by removing effects of *internal links* unrelated to the page popularity.

Algorithm 1 shows how the partition refinement algorithm works for LexBFS. In lines 1-2, it creates an initial class and the class list for given initial URLs. In lines 4-5, it picks a URL from the first nonempty class of the class list, and then it performs crawling in line 6. In lines 7-13, it divides each class by external and initial links of the crawled page. In lines 14-17, it adds new URLs as the last class of the class list. It repeats this process until the end of crawling.

Algorithm 1 Web Crawling Algorithm using LexBFS

```
1: $C_{init}$ = a URL list which contains initial URLs
2: $L$ = the class list ($C_{init}$)
3: while $L$ has a nonempty class do
4:   $C_{first}$ = the first nonempty element of $L$
5:   $u$ = pop the first element of $C_{first}$
6:   $p$ = DOWNLOAD($u$)
7:   $links_p$ = external links and initial links of $p$
8:   for each class $C_i$ in $L$ do
9:     $Y$ = the list of URLs in $C_i$ which are the target URLs of $links_p$
10:    if $Y$ is not empty then
11:      divide $C_i$ into $Y$ and ($C_i - Y$)
12:    end if
13:  end for
14:  if $p$ has new URLs then
15:    $C_{new}$ = a URL list which consists of new URLs of $p$
16:    append $C_{new}$ to the end of list $L$
17:  end if
18: end while
```

3) LexDFS has no known O(|V|+|E|) algorithm.

4.3 Web crawling Ordering using MCS

We design a linear time web crawling algorithm using MCS by modifying the partition refinement method [9] which implements LexBFS in linear time. All URLs with the same number of incoming links are contained in a doubly-linked list called *class*. All classes are also connected by a doubly-linked list maintained in decreasing order of the number of incoming links.

When a web page is crawled, each URL pointed to by links from the crawled page moves to the adjacent class containing pages with one more incoming links. When the class does not exist, a new class is inserted to the class list, and the URL is moved to the new class. New URLs extracted from the crawled page are added to the last class of the class list. Then in the next step, each class has pages with the same number of incoming links again. This operation requires time proportional to the number of outgoing links of the crawled page, hence the overall time complexity is O(|V|+|E|).

As in LexBFS, we use only external links and initial links for the purpose of dividing classes, i.e., non-initial links do not contribute to class division but used only to add new URLs to crawl. Empirically, we observed that if MCS treats all links equally, it crawls only a few sites in concentration because of numerous non-initial links.

Algorithm 2 shows our algorithm for MCS. In the pseudo code, $C_{<k>}$ denotes the class which contains pages of k incoming links. In lines 1-3, it creates an initial class and the class list for given initial URLs. In lines 5-6, it picks a URL from the first nonempty class of the class list, and then it performs crawling in line 7. In lines 8-17, it moves target URLs of the extracted links to the adjacent classes with one more incoming links. In lines 18-20, it adds new URLs to $C_{<1>}$. It repeats this process until the end of crawling.

Note that our algorithm is essentially similar to the MCS algorithm proposed by Tarjan and Yannakakis [6], but the latter has a global initialization step. Our algorithm considers the graph incrementally, and thus it is more appropriate for web crawlers.

Algorithm 2 Web Crawling Algorithm using MCS

```
1: $C_{init}$ = a URL list which contains initial URLs
2: $C_{\langle 1\rangle}$ = an empty URL list
3: $L$ = the class list ($C_{init}, C_{\langle 1\rangle}$)
4: while $L$ has a nonempty class do
5:   $C_{first}$ = the first nonempty element of $L$
6:   $u$ = pop the first element of $C_{first}$
7:   $p$ = DOWNLOAD($u$)
8:   $links_p$ = external links and initial links of $p$
9:   for each link $x$ in $links_p$ do
10:     $t$ = the target URL of $x$
11:     if $t$ is in class $C_{\langle k\rangle}$ of $L$ then
12:       if $C_{\langle k+1\rangle}$ does not exist in $L$ then
13:         insert an empty class $C_{\langle k+1\rangle}$ to $L$
14:       end if
15:       move $t$ to class $C_{\langle k+1\rangle}$
16:     end if
17:   end for
18:   if $p$ has new URLs then
19:     add new URLs of $p$ to $C_{\langle 1\rangle}$
20:   end if
21: end while
```

5. Experiment

5.1 Experimental setup

We performed experiments in 5 machines with CPU Intel Pentium 2.8GHz and 2GB RAM under Linux (Redhat Fedora Core4). Within a set of 13 million crawled web pages (approximately 500GB), we virtually crawled by BFS, LexBFS, and MCS up to 250,000 pages. To evaluate different strategies, we used two performance measures: cumulative PageRank and the number of sites. Initial URLs are 3 famous portal sites in Korea.

5.2 Experimental results

Cumulative PageRank Figure 3 shows cumulative PageRank by the amount of crawled web pages on BFS, LexBFS, and MCS. In this figure, BFS and LexBFS are indistinguishable for the most part. When crawling starts, cumulative PageRank becomes nearly 0.3 and increases in two steps. Meanwhile, the plot of MCS increases rapidly from 30,000 pages, and after 50,000 pages it's above 0.75, finishes above 0.85 in overall crawling of 250,000 pages. MCS does collect pages that are 85% of PageRank when it just crawled 0.5% of the experiment set. It means MCS web crawler can crawl popular pages fast without being confined to some local sites.

Number of Sites Figure 4 demonstrates the number of sites by the amount of crawled web pages. In this figure, BFS and LexBFS are

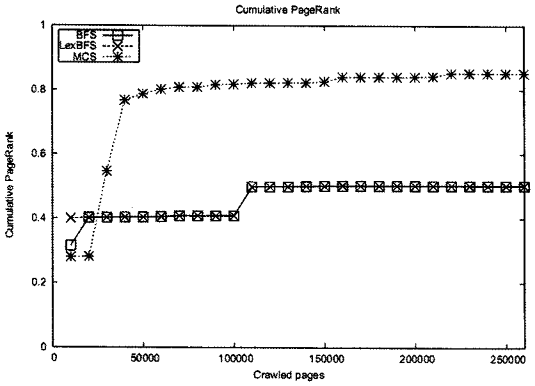


Figure 3 Cumulative PageRank by the amount of crawled web pages

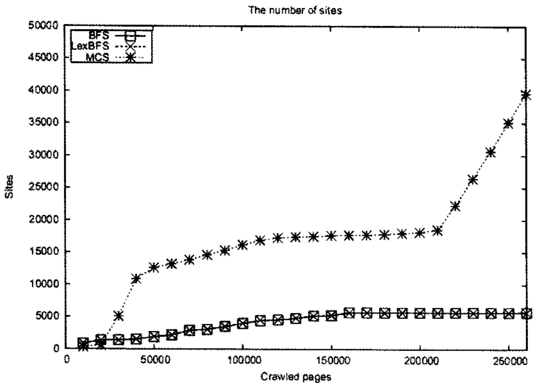


Figure 4 The number of sites by the amount of crawled web pages

indistinguishable again for the most part. BFS and LexBFS crawled 5,000 distinct sites while MCS crawled 3 times more than that of BFS and LexBFS at 100,000 pages. After 100,000 pages, the plots of BFS and LexBFS increase slightly. It means they do not spread out widely to crawl efficiently on the web. In MCS, however, the number of sites has increased rapidly from 18,000 to 35,000 sites after crawling 200,000 pages and finished in about 7 times larger than that of BFS and LexBFS. It means that MCS web crawler crawls the Web quite evenly and has much wider range compared to others.

6. Conclusion

In this paper we considered various graph search techniques for web crawling ordering and proposed linear time algorithms for LexBFS and MCS based on the partition refinement method. To improve efficiency, we use *external links* and *initial links* to determine priorities of URLs. We applied them to web crawlers and evaluated by cumulative PageRank and the number of sites. Experimental results show that MCS has desirable properties in both time complexity and the quality of crawled pages.

We hope that our work helps to connect two different areas: graph theory and web crawling. Expanding our work, other graph search techniques not covered in this paper can be considered for web crawling ordering. A careful investigation of the Web would lead to discover web crawling orderings that might be better than MCS.

References

[1] J. Cho, H. Garcia-Molina, L. Page. Efficient crawling through URL ordering *In Proceedings of 7th World Wide Web Conference*, 1998.

[2] M. Najork, K. Wiener. Breadth-first search crawling yields high-quality pages. *In Proceedings of 10th World Wide Web Conference*, 2001.

[3] S. Abiteboul, M. Preda, and G. Cobena. Adaptive On-Line Page Importance Computation. *In WWW 2003*, 2003.

[4] P. Boldi, M. Santini, and S. Vigna. Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations. In *International Workshop on Algorithms and Models for the Web-Graph (WAW), LNCS*, vol.3, 2004.

[5] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez. Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering. In A. Ellis and T. Hagino, editors *WWW (Special interest tracks and posters)* ACM, 2005.

[6] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566 579, Aug. 1984.

[7] D. Corneil, R. Krueger, A unified view of graph searching, *ICGT'05: Proceedings of 7th French International Colloquium on Graph Theory*, Hyeres, France, 2005.

[8] R. Krueger, Graph searching, PhD thesis, University of Toronto, 2005.

[9] M. Habib, R. McConnell, C. Paul, L. Viennot, LexBFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, *Theoretical Computer Science*, 234(1-2): 59-84, March 2000.

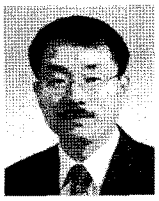
[10] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Pro ject, Stanford University, Stanford, CA, USA, Nov. 1998.
[11] J. Cho, U. Schonfeld. Rankmass crawler: a crawler with high personalized PageRank coverage guarantee. *In VLDB Endowment*, 2007.
[12] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine. *In Proceeding of 7th World Wide Web Conference*, 1998.
[13] A. Berry, J. R. S. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287-298, 2004.

김 진 일
2005년 서울대학교 전기공학부 학사. 2007년 서울대학교 전기컴퓨터공학부 석사. 2007년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 암호학, 웹 검색, 컴퓨터이론

권 유 진
2007년 고려대학교 정보통신대학 컴퓨터학과 학사. 2009년 서울대학교 전기컴퓨터공학부 석사. 2009년~현재 IBM Korea 연구소 Ubiquitous Computing Lab. 관심분야는 컴퓨터이론, 알고리즘, 생물정보학, 의료정보시스템

김 진 욱
1998년 서울대학교 수학과 학사. 2000년 서울대학교 컴퓨터공학과 석사. 2006년 서울대학교 전기컴퓨터공학부 박사. 2006년~2009년 ㈜에이치엠연구소 책임연구원. 2009년~현재 인하대학교 컴퓨터정보공학부 연구교수. 관심분야는 컴퓨터이론, 알고리즘, 웹 검색, 생물정보학, 암호학

김 성 렬
1993년 서울대학교 컴퓨터공학과 학사
1995년 서울대학교 컴퓨터공학과 석사
2000년 서울대학교 컴퓨터공학과 박사
2000년~2002년 WiseNut Inc., Engineering Manager. 2002년~현재 건국대학교 인터넷미디어공학부 교수. 관심분야는 알고리즘, 컴퓨터이론, 웹 검색, 분산처리


박 근 수
1983년 서울대학교 컴퓨터공학과 학사
1985년 서울대학교 컴퓨터공학과 석사
1992년 미국 Columbia 대학교 전산학 박사. 1991년 11월~1993년 8월 영국 런던대학교 King's College 조교수. 1993년 8월~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 컴퓨터이론, 생물정보학, 암호학, 웹 검색