링크 정보를 활용한 확장된 웹 검색 방법론

김승[O*] 임태수[**] 이우기[***] 강석호[*]
[*]서울대학교 산업공학과 {seung2@netopia[O], shkang}.snu.ac.kr
[**]성결대학교 컴퓨터공학부 tshou@sungkyul.edu
[***]인하대학교 산업공학과 wookey@gmail.com

A Rich Web Search Mechanism using Linkage Information

Seung Kim[O*] TaeSoo Lim[**] Wookey Lee[***] Suk Ho Kang[*]
[*]Dept. of Industrial Engineering, Seoul National University
[**]Dept. of Computer Engineering, Sungkyul University
[***]Dept. of Industrial Engineering, Inha University

요 약

기존의 웹 검색 시스템은 주어진 키워드들을 모두 포함한 웹 페이지의 도출을 목표로 하기 때문에, 검색어의 popularity가 떨어지는 경우나, 키워드의 한정 정도가 과도하거나 많아지는 경우, 또는 키워드가 길어지는 경우 등에는 검색결과가 크게 줄어드는 결과희소성 문제(scarcity problem)를 겪게 된다. 본 연구에서는 이를 해결하기 위해서 검색결과를 개별 웹페이지 집합이 아닌 관련 웹페이지들간의 링크 구조로 도출해주는 확장된 검색 방법을 제시한다. 확장된 웹 검색은 링크기반 분석을 활용하여 개별 질의들에 대한 검색 결과 페이지들간의 링크 관계를 탐색하고, 도출된 검색 결과의 순위 측정을 통해 이루어진다. 본 연구에서 제안하는 확장된 웹 검색 방법은 결과희소성 문제에 효과적임을 보였다.

1. 서 론

인터넷의 폭발적인 성장에 따라, 웹은 현재 전세계 규모의 다종다양한 정보에 대해 실시간적 접근을 가능하게 해주는 거의 유일한 정보원천이 되고 있다. 역설적으로 정보량이 방대해 지면서 관련성이 높은 정확한 정보를 선별해내는 데에는 더욱더 어려움이 가중되고 있다. 즉, 정보량 확대의 역기능으로 웹 사용자는 '정확한 정보의 선택'이라는 어려움을 겪게 되었다. 그러므로 웹 정보의 효과적, 효율적인 선택 문제는 나날이 그 중요성과 시의성이 커지고 있는 주제인 것이다.

일반적으로 웹 사용자가 검색엔진을 사용하여 필요한 정보를 검색하고자 할 때 여러 가지 문제점은 전통적인 정보검색 분야의 수행도 측정지표인 precision과 recall의 관점에서 파악해 볼 수 있다. 우선 Kleinberg[1]가 분류한 웹 검색 질의의 종류를 살펴보면 다음과 같다.

▶ 구체성 질의(Specific query)
예) "Does Netscape support the JDK 1.1 code-signing API?"
▶ 보편성 질의(Broad-topic query)
예) "Find information about the Java programming language."
▶ 유사성 질의(Similar-page query)
예) "Find pages 'similar' to java.sun.com."

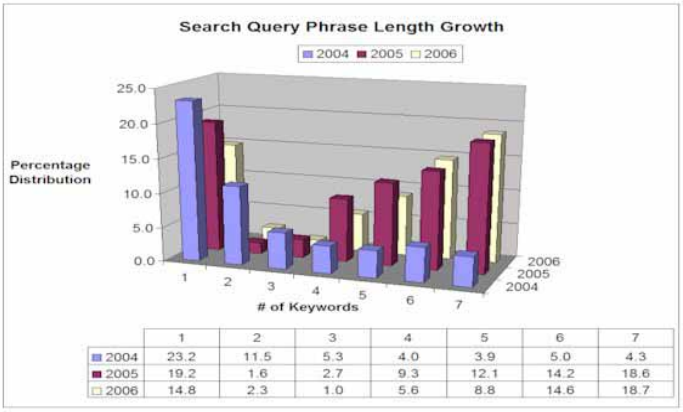
질의의 상세화가 많이 이루어지는 구체성 질의의 경우에는 검색 결과의 precision은 높일 수 있으나, recall이 낮아지는 결과로 이어질 가능성이 크고, 이는 제시되는 검색 결과의 수가 매우 적어지는 결과희소성 문제(scarcity problem)로 귀결된다. 이와 반대로 질의의 범위가 넓은 보편성 질의의 경우에는 결과 폭증 문제(abundance problem), 즉, 검색결과의 recall이 커져서 발생하는 문제로서, 지나치게 과다한 검색 결과에 대한 선택의 문제를 겪을 가능성이 커지게 된다. 또한 '*java.sun.com과 유사한 페이지를 검색하라.*'와 같은 특정 사이트와 의미적으로 유사한 페이지를 검색하는 유사성 질의 문제 역시 전통적인 VSM(Vector Space Model) 기반의 웹 검색엔진이 처리하기에는 특히 어려운 문제이다.

전통적인 VSM 기반의 웹 검색에서 겪을 수 있는 또 다른 문제들로는 검색어의 동음이의어(homonym), 동음다의어(polysemy) 관계 때문에 발생하는 topic-drift 문제가 있다. 이는 AltaVista[2]나 Google[3][4]과 같은 정보수집기(crawler) 기반 검색엔진이 자동색인을 사용할 때에 생길 수 있다. Topic-drift 문제는 주어진 질의에 대한 관련문서를 찾지 못하여 비정상적으로 검색 결과 집합의 precision이 낮은 문제를 의미하며, 이를 해결하기 위해 하이퍼링크 기반 분석(Linkage Analysis)을 활용한 방법이 다수 제안되고 있다[5].

하이퍼링크는 웹 환경에서만 존재하는 정량적인 정보로서, 웹페이지를 분석하면 쉽게 얻어낼 수 있다. 이러한 하이퍼링크는 웹페이지의 제작자 혹은 웹 마스터가 관련성을 부여한 근거로 사용될 수 있으며, 구글폭탄(Google Bomb)의 경우와 같은 인위적인 조작을 제외한 대부분의 경우에 해당된다.

웹 분석 연구조직인 OneStat.com[6]에 따르면 웹 사용자가 사용하는 질의 길이가 최근 들어 더욱 길어지고 있는 추세에 있다([그림1] 참조). 질의 길이가

길어지면 더욱 정확한 검색이 가능해 질 수 있지만 그만큼 결과희소성 문제를 겪게 될 가능성도 커진다. 질의를 구성하는 단어들의 popularity 또한 이러한 가능성에 영향을 준다. 즉, popularity가 떨어지는 일부 특정 도메인에만 국한된 질의 단어가 입력될 때 결과희소성 문제를 겪게 될 확률이 높아지는 것이다.



	1	2	3	4	5	6	7
2004	23.2	11.5	5.3	4.0	3.9	5.0	4.3
2005	19.2	1.6	2.7	9.3	12.1	14.2	18.6
2006	14.8	2.3	1.0	5.6	8.8	14.6	18.7

[그림 1] 검색 쿼리 길이의 변화추이 [6]

이러한 문제점은 현재의 거의 모든 웹 검색 엔진의 결과가 개별 질의 키워드를 모두 포함하는 단일 웹 페이지의 검색을 목표로 하는데 기인한다. 어떤 질의 σ^1, σ^2, σ^3가 주어졌을 때, 대부분의 검색 엔진은 그 빈도수 등을 계산하여 '$\sigma^1 \wedge \sigma^2 \wedge \sigma^3$'를 포함하는 웹 페이지(들)를 결과로 제시하기 마련이다. 그 세 단어 모두를 포함하는 웹페이지가 없는 경우 결과가 없다고 리턴 한다. 예컨대, '이교수님(P: Professor Lee)의 강의안(L: Lecture Notes)에서 웹구조화(W: Web Structuring)'을 알고 싶은 경우 이는 σ^1= P, σ^2= L, σ^3= W라고 표현될 수 있다.

일반적으로 결과희소성 문제는 다음 3가지 종류로 분류할 수 있다. 첫째, 키워드의 개수가 증가하면서 검색 도메인이 급격히 줄어들어 결국 결과희소성 문제에 봉착하게 되는 경우, 둘째, 질의가 상하관계 혹은 포함관계를 가지는 경우, 셋째, 사용자가 관련된 여러 페이지를 원하는 경우이다.

이러한 문제에 대한 대안은 기존 검색엔진을 사용하는 경우 질의를 분해하여, 해를 좁혀나가는 분할정복방식(divide & conquer)이다. 즉, 질의 부분집합을 조합하여 수행하는데, $\sigma^1 \wedge \sigma^2$ 결과로부터 σ^3을 검색하거나, 혹은 σ^1을 수행한 다음 $\sigma^2 \wedge \sigma^3$를, 혹은 σ^2와 σ^3에 대해 개별적으로 혹은 대화식으로 검색을 시도하게 된다. 그러나 이러한 방식은 질의 입력순서에 영향을 크게 받거나, 이러한 예에서와 같이 개별 질의가 보편성이 큰 경우 분할의 의미가 적다. 가장 문제가 되는 경우는 세번째의 경우로서 사용자가 '일련의 웹 페이지'를 결과로 원하는 경우에는 페이지 단위의 결과를 제시하는 기존 검색방식으로는 이러한 문제를 해결할 수 없다.

본 연구의 동기는 질의를 구성하는 문자열 중 개별 단어를 포함한 일부 페이지와 또 다른 일부 개별 단어를 포함한 페이지 간에 링크 관계가 존재한다면 이러한 페이지들까지 주어진 질의의 검색 결과로 주어져야 한다는 데에서 출발한다. 물론 기존의 검색엔진은 이러한 웹페이지간의 링크가 존재하더라도 이러한 웹페이지들은 키워드의 모든 단어를 포함하지 않고 있기 때문에 검색결과로서 이를 도출해주지 않는다. 이를 위해, 본 연구에서는 검색 질의의 길이가 길어질 때, 즉 구체성 질의가 입력될 때, 생길 수 있는 결과 희소성 문제를 해결하기 위해 링크 기반 분석을 활용한 웹 검색 확장 알고리즘을 제안한다. 검색 결과는 질의에 대한 다수의 관련 웹페이지들과 이들간의 링크로 이루어지는 웹페이지 집합구조(네트웍)로 제공된다. 이러한 접근법을 활용하면 질의가 담고 있는 주제가 다수개인 다중 하부주제 질의(multiple sub-topic query)가 입력되더라도 이를 효과적으로 처리할 수 있게 된다.

2. 알고리즘

본 연구에서 질의는 키워드의 집합으로 보고, 검색대상 페이지들 사이에는 하이퍼텍스트 링크(이하 링크)를 가진다고 가정한다. 다음에 설명하는 알고리즘은 전체 질의 키워드 중, 단일 키워드에 대한 검색결과 페이지로부터 서로 간의 링크의 존재를 탐색하여 질의에 대한 검색 결과로 웹 페이지와 그들간의 링크 관계를 검색결과로 도출해주는 방식이다.

먼저, 질의에 대한 결과로 제시할 페이지 집합을 포함하는 검색 대상 집합 Sσ가 가져야 할 성질은 다음과 같다.

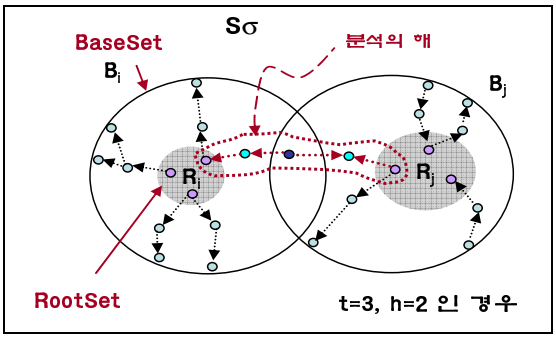
① Sσ의 크기가 상대적으로 작을 것
② Sσ의 내부에 개별 질의 문자열들을 포함한 페이지(relevant page)가 많을 것
③ Sσ의 내부에 존재하는 relevant page들간에 짧은 hop수로 연결되는 링크가 존재할 것

검색대상 Sσ는 다음과 같이 만들어진다. 먼저, k개의 질의어(query term)를 가지는 전체 질의집합 σ에서 각각 1부터 k까지에 해당하는 단일 질의 $\sigma^1,\ldots,\sigma^k$에 대해 t개의 가장 높은 관련성을 갖는 검색결과 페이지 집합을 기존의 검색엔진을 통해서 구하고 이를 RootSet $R_1,\ldots,R_k$로 칭한다. 다시 이 RootSet은 서로간의 링크가 존재할 때 까지 확장을 하며, 확장 횟수를 hop수 h로 두고, 이 확장된 페이지 집합을 BaseSet $B_1,\ldots,B_k$로 둔다. 이때 링크의 방향성(inbound, outbound)은 무시한다.

전체 BaseSet에 속하는 페이지들을 종류별로 나눠보면 다음과 같다.

종류① :단일 질의 σ^1~ σ^k에 대해 연관되어 검색된 페이지(RootSet)
종류② :RootSet으로부터 임의의 hop수의 링크로 연결되는 확장된 페이지(BaseSet)
종류③ : 종류②에 속하며, 둘 이상의 BaseSet간에 중복하여 존재하는 페이지
종류④ : 종류②에 속하며, 종류①과 종류③ 페이지를 링크로 연결해주는 중간 페이지
종류⑤ : 종류②중에서 종류③, ④에 포함 되지 않은 페이지

그림2는 이러한 5개 종류로 이루어지는 검색대상 Sσ를 예시하고 있다. 그림2의 예에서는 t=3 즉, 3페이지로 구성된 RootSet과 h=2 즉, RootSet을 2번 확장하여 만들어지는 BaseSet, 그리고 알고리즘의 결과로서 나타나게 되는, 질의에 대한 결과인 웹 페이지(노드)와 그들간의 링크관계가 나타나 있다.



[그림 2] 검색 대상집합 Sσ의 생성

전체 알고리즘은 크게 네 부분으로 이뤄진다. 첫번째 부분은 단일 질의 σ^1~ σ^k에 대한 k개의 RootSet을 만드는 과정이다(아래 그림3, **RICH-SEARCH**의 3~5행).

```
RICH-SEARCH (σ, t)
▷ Main search algorithm which input is array σ consisting of query terms
1   Graph[] ← [NodeURL[],NodeWeight[], Edge[]]
2         ▷ Storage structure of a Graph
3   for i ← 1 to |σ|
4       do R[i,1..t]← CUSTOM-SEARCH (σ[i])
5           ▷ for each query term, retrieve top t relevant
              pages using custom search engine like
              Google.
6   for i ← 1 to t
7       do Graph[i] ← MINIMUM-EXPAND(R[1,i])
8           ▷ for each seed page i in R[1,1..t], retrieve
              minimally expanded network i.
9   for i ← 1 to t
10      do quality[i] ← CENTROID-ESTIMATE (Graph[i], μ)
11  return maximum[quality[1]..quality[t]]
```

[그림 3] RICH-SEARCH 알고리즘

두번째 부분은 k개의 개별 RootSet들간에 짧은 링크 관계가 존재하는지를 탐색하기 위해 RootSet을 BaseSet으로 확장하고 BaseSet들간의 중복 페이지를 찾는 과정이다(부록 **MINIMUM-EXPAND**의 1~13행). 세번째 부분은 해당 시점까지 만들어진 BaseSet에 저장된 노드 배열과 링크(행렬)구조로부터 최종적인 해답으로 도출될 웹 페이지들과 이들간의 링크 관계를 담고 있는 인접 행렬(adjacency matrix)을 도출하는 과정이다(부록 **MINIMUM-EXPAND**의 14~23행). 마지막으로 네번째 부분은 구해진 다수의 웹페이지 집합구조(그래프)로부터 해당 구조의 중요도에 대한 측정에 관련된 부분이다(부록 **CENTROID-ESTIMATE**).

전체 알고리즘 **RICH-SEARCH**는 그림3과 같다.

그림4는 그림2와 같은 개별 RootSet의 일부가 주어졌다고 할 때 전체 알고리즘의 두번째 단계에서 이루어지는 BaseSet의 확장 및 관련 웹페이지 URL의 저장 배열과 인접행렬의 생성 과정을 보여준다(부록 **MINIMUM-EXPAND**의 1~13행).

먼저 질의 σ^i, σ^j 대한 검색엔진의 결과로 RootSet Ri, Rj가 생성된다. 그림4에서는 i=1, j=i+1이며, 이후 i=k-1까지 반복 수행된다.

BaseSet Bi와 Bj의 초기값은 Bi는 RootSet Ri의 1개 페이지, Bj는 Rj가 그대로 할당된다. 또한 관련 저장구조 NodeURL 배열에 Bi와 Bj간에 중복되는 페이지가 존재하지 않으면 각각 Bi와 Bj를 1hop씩 확장하면서 중복페이지의 존재 유무를 검사하고 이 과정은 중복페이지가 생성될 때까지 이루어 진다. 그림4에서는 2hop 확장에서 공통 페이지 *E*의 존재를 확인하는 과정을 보여준다. 이러한 과정은 알고리즘의 두번째 과정을 거쳐 생성되는 현단계까지의 해(A-C-E-h-b)를 다음번 Ri로 두고 i=k-1까지 반복 수행된다.

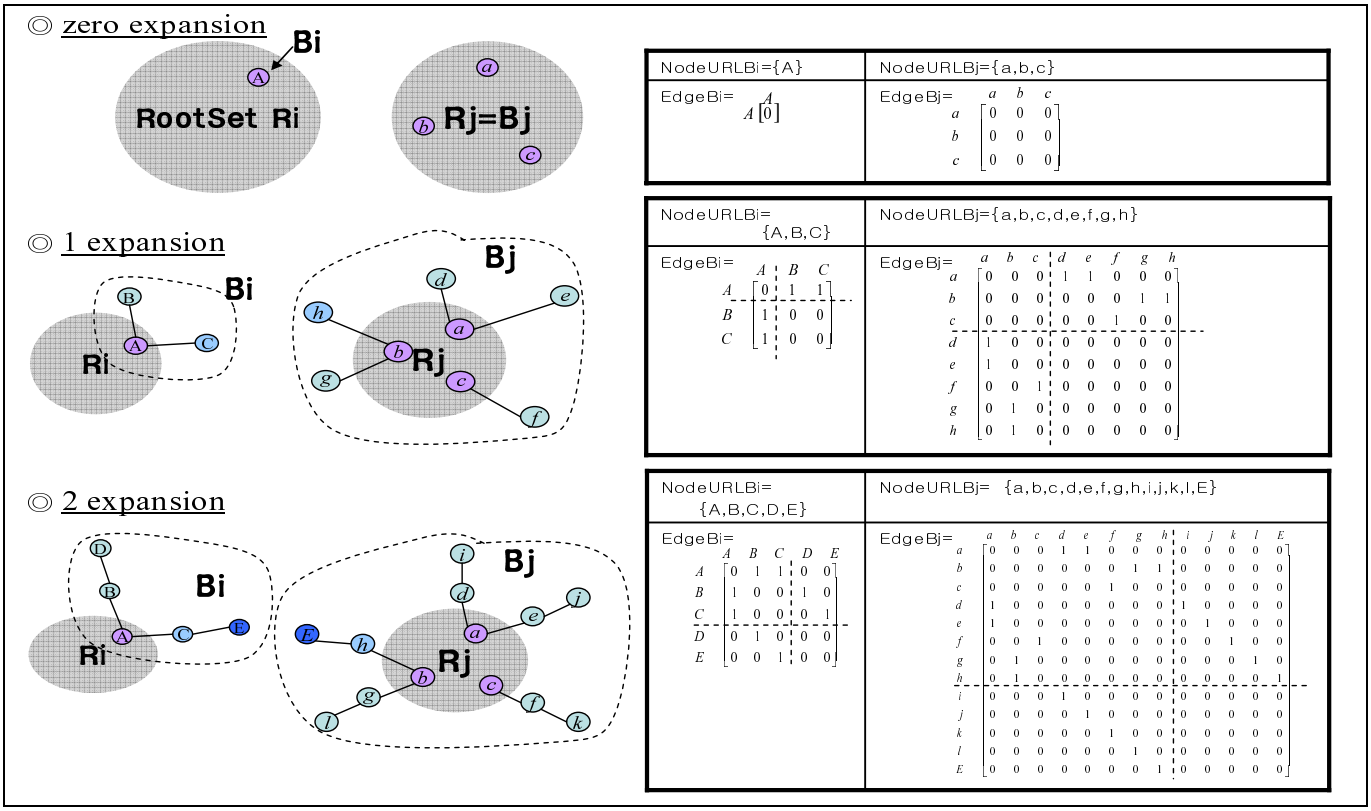
정의 1. 인접행렬(Adjacency Matrix)
0과 1을 요소로 갖는 $n \times n$의 대칭행렬인 (n은 웹 페이지의 개수) 인접행렬 **A**의 요소 a_{ij}는 다음과 같이 정의된다.

$$a_{ij} = \begin{cases} 1 & \text{웹페이지 } i\text{와 } j\text{간에 링크가 존재할때,} \\ 0 & otherwise. \end{cases} \quad ....(1)$$

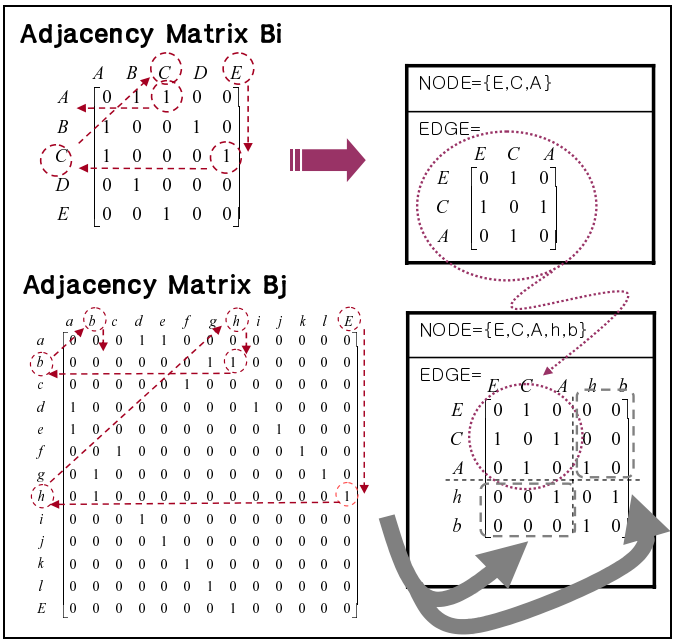
정의 2. 확장인접행렬(Adjacency Matrix Expansion)
매 확장 단계의 인접행렬은 다음 구조로 이뤄진다.

$$AdjacencyMatrix,\ \mathbf{A}^i = \begin{bmatrix} \mathbf{A}^{i-1} & \mathbf{E} \\ \mathbf{E}^T & \mathbf{0} \end{bmatrix} \quad .....(2)$$

단, $\mathbf{A}^{i-1}$은 현 단계 이전의 인접행렬,
E 는 해당 확장 단계에 포함되는 웹페이지와 이전 단계까지 포함된 웹페이지간의 인접행렬,
0 은 0 행렬.



[그림 4] RootSet의 BaseSet으로의 확장과 그에 따른 저장 구조의 변화



[그림 5] 최종 웹페이지와 관련 인접행렬의 도출

그림5는 알고리즘이 공통페이지를 찾을 때까지 확장된 BaseSet Bi, Bj의 인접행렬로부터 질의에 대한 해답으로 도출할 웹페이지와 그들간의 링크관계를 나타낼 인접행렬을 도출하는 과정을 보여준다. 그림 4에서의 예에서는 A-C-E-h-b의 웹페이지 및 이들의 링크 관계 이다.

공통 페이지 E를 찾을 때까지 확장된 인접행렬 Bi, Bj의 최우측열 요소는 E가 된다(그림 4의 최하단 테이블). 먼저 Bi 행렬로부터 최우측 열의 첫번째 행에서부터 1 요소를 찾을 때까지 행의 수를 증가시켜 가다가 1 요소를 만나면 해당 시점의 행의 수를 다음 번의 컬럼의 수로 놓고 다시 1 행부터 1 요소를 만날 때까지 행의 수를 반복시켜 가면서 1 요소를 찾아가고, 이 과정을 반복하면 RootSet Ri에 속한 페이지들로부터 웹페이지 E로 연결된 경로상의 웹페이지와 이들간의 링크관계를 얻을 수 있다. 이 과정을 다시 Bj의 확장된 인접행렬에 까지 반복하면 임의의 개별 질의 σ^i와 σ^j에 대응하는 관련 페이지 및 이들간의 링크관계를 찾을 수 있다.

3. 관련 연구

링크기반 분석은 특정 분야에서 핵심이 되는 논문이나 저자, 저널 등을 찾기 위한 인용 분석(citation analysis)이 그 시초이다. 여기에서는 저자나 저널 등의 객관적이고 정량적인 중요도를 측정하기 위해서 피인용 횟수를 기본적으로 사용한다. 이러한 중요도 중에 가장 잘 알려진 것이 Garfield[7]의 impact factor이다[5].

인용분석에서의 '피인용'은 웹환경에서는 웹페이지 간의 '링크'로 대비해 볼 수 있으며, 링크기반 분석은 이러한 링크 정보를 활용하여 이루어진다. 본 연구에서는 결과희소성 문제를 해결하는데 링크기반 분석을 활용하였고, 링크기반 분석은 이외에 기존의

VSM 기반 검색에서 겪을 수 있는 결과폭증문제 [1,4]나 유사성질의 문제[1]를 해결하는데에도 활용될 수 있다.

한편, 현재 이루어지고 있는 웹 검색의 질의와 그에 대한 결과는 각각 질의의 키워드들과 다수의 관련 웹 페이지 집합이다. 쿼리가 제한하는 검색범위가 좁아질수록 즉, 쿼리를 구성하는 문자열의 길이가 길어질수록 (쿼리가 구체성질의에 가까워질수록), 단일 웹페이지가 해당 단어들을 포함하는 경우는 기하급수적으로 줄어 들것이며 따라서, 전통적인 검색엔진의 검색 메커니즘으로는 결과희소성 문제를 피할 수가 없게 된다.

구체성질의 문제에 관련된 연구로 질의확장(Query expansion)[8]이 있다. 질의확장은 pseudo-relevance feedback 등의 방법으로 짧게 입력된 질의를 사용자의 개입 없이 자동으로 확장함으로써 검색의 질을 높이기 위한 방법이다. 그러나 이러한 질의확장 방법 역시 결과희소성 문제를 근본적으로 해결하지는 못하며 다중 하부주제 질의 문제에도 효과적으로 대응하기는 힘들다.

4. 결 론

본 연구에서는 기존 검색엔진의 문제점, 즉, 모든 질의 키워드를 포함한 웹 페이지의 검색이라는 단점을 해결하기 위한 방안을 제시하였다. 이를 위해 본 연구에서는 개별 질의 키워드의 검색결과들 간의 링크 관계를 탐색하는 알고리즘을 제시하였고 알고리즘으로 도출되는 네트웍들간의 순위 측정을 위한 척도를 제안하였다. 본 연구에서 제시한 방법의 공헌은 다음과 같다. 검색 질의가 길어지거나, 검색 질의의 popularity가 떨어질 때(일부 특정 도메인에만 국한된 질의 단어의 경우) 혹은 사용자가 연결된 다중의 페이지를 결과로 원하는 경우에 야기되는 결과희소성 문제에 효과적인 대안이 되는 것을 보였다.

5. 참고문헌

1. Kleinberg, M., Authoritative Sources in a Hyperlinked Environment, *The Journal of the ACM*, Vol.46, No.5, pp.604–632, 1998.
2. AltaVista, http://www.altavista.com/
3. Google, http://www.google.com/
4. Brin, S., Page, L., The anatomy of a large-scale hypertextual web search engine, *Computer Networks*, Vol.30, No.1-7, pp107–117, 1998.
5. Hou, J., Zhang, Y., Effectively Finding Relevant Web Pages from Linkage Information, *IEEE TKDE*, Vol.15, No.4, pp940–951, 2003.
6. OneStat.com, http://onestat.com/
7. Garfield, E., Citation Analysis as a Tool in Journal Evaluation, *Science*, Vol.178, pp.471–479, 1972.
8. Belkin, N.J., Cool, C., Kelly, D., Kim, G., Kim, J.-Y., Lee, H.-J., Muresan, G., Tang, M.-C., Yuan, X.-J., Query Length in Interactive Information Retrieval, In *Proc. SIGIR*, pp. 205-212, 2003

부록 - 알고리즘

```
MINIMUM-EXPAND(R[1,i])
1   do B1[1] ← R[1,i]
2      ▷ Also, update NodeURL_B1, NodeWeight_B1, Edge_B1 with R[1,i]
3   for j ← 1 to t
4       do B2[j] ← R[2,j]
5          ▷ construction of two Base set for expansion
6          ▷ Also, update NodeURL_B2, NodeWeight_B2, Edge_B2 with R[2,j]
7   for k ← 1 to |σ|-1
8       do index ← 0
9          Alternatively expand B1 and B2 at 1 hop until B1∩B2 ≠ NIL
10         ▷ Also, update NodeURL_B1, NodeWeight_B1, Edge_B1 with B1
11         ▷ Also, update NodeURL_B2, NodeWeight_B2, Edge_B2 with B2
12         TempNode ← B1∩B2
13         Graph[k] ← Graph[k] + [URL of TempNode, Weight of TempNode, 1]
14         column ← TempNode index at NodeURL_B1, size ← length[NodeURL_B1]
15         for row ← 1 to size
16             do if Edge_B1[row,column] = 1
17                then Graph[k] ← Graph[k] + [NodeURL_B1[row], NodeWeight_B1[row],1]
18                     column ← row, size ← column, index ← index+1
19         column ← TempNode index at NodeURL_B2, size ← length[NodeURL_B2]
20         for row ← 1 to size
21             do if Edge_B2[row,column] = 1
22                then Graph[k,index] ← Graph[k] + [NodeURL_B2[row], NodeWeight_B2[row],1]
23                     column ← row, size ← column, index ← index+1
24         for j ← 1 to |Graph[k]. NodeURL[j]|
25             do B1[j] ← Graph[k].NodeURL[j]
26                ▷ Also, update NodeURL_B1, NodeWeight_B1, Edge_B1 with Graph
27         for j ← 1 to t
28             do B2[j] ← R[k+2,j]
29                ▷ Also, update NodeURL_B2, NodeWeight_B2, Edge_B2 with R[k+2,j]
30  return Graph[]

CENTROID-ESTIMATE (Graph[i], μ)
1   do difference ← 999, CurStep ← 0
2   while difference > threshold
3       do CurStep ← CurStep + 1
4       for each node j in Graph[i]
5           for all the other nodes k (≠j)
6               do Graph[i].NodeWeight[j] ← Graph[i].NodeWeight[j] + Influence(k, j, μ)
7                  ▷ influence(k, j, μ) is calculated using Equation (4)
8           do TotalWeight[i] ← TotalWeight[i] + Graph[i].NodeWeight[j]
9       do difference ← 0, PrevStep ← CurStep
10      for each node j in Graph[i]
11          do Graph[i].NodeWeight[j] ← Graph[i].NodeWeight[j] / TotalWeight[i]
12      for each node j in Graph[i]
13          do difference ← difference + difference of Graph[i].NodeURL[j] between PreStep and CurStep
14  return maximum[NodeURL[i]]
```