순서를 고려하는 k-키워드 근접도 문제를 위한 빠른 알고리즘

(A Fast Algorithm for the k-Keyword Ordered Proximity Problem)

김 진 욱 †

(Jin Wook Kim)

요 약 웹 검색 엔진들은 질의에 대한 문서의 적합성을 판단하기 위한 방법의 하나로 근접도를 사용한다. 근접도는 키워드의 순서를 고려하지 않는 방식과 순서를 고려하는 방식이 모두 연구되어왔다. 본 논문에서는 k개 키워드의 순서를 모두 고려하는 근접도 문제를 위한 $O(n)$ 시간 알고리즘을 제시한다. 이때, n은 k개의 키워드가 문서에 나타난 전체 횟수이다. 또한 실험을 통해 이전 연구 결과보다 $k=2$인 경우는 약 1.2배의 속도 향상을, $k=5$인 경우는 3배 이상의 속도 향상이 있음을 보인다.

키워드 : 근접도, 순서 유지, 문자열 알고리즘

Abstract In the web search engines, the proximity is used to compute the relevance of a document to the given query. There exist various research results about the proximity problems and the ordered proximity problems. In this paper, we present $O(n)$ time algorithms for the k-keyword ordered proximity problems where n is the total number of occurrences of the k keywords in a document. Experimental results show that the proposed algorithms are about 1.2 times and over 3 times faster than the previous results when $k=2$ and $k=5$, respectively.

Key words : proximity, order preserving, string algorithms

1. 서 론

웹 검색 엔진들은 일반적으로 다양한 방식의 검색을 지원한다[1-4]. 가장 기본적인 방식은 질의(query)에 포함된 키워드를 모두 포함하는 문서를 찾는 것인데, 이때 키워드의 순서를 고려하지 않는 방식과 고려하는 방식이 있다. 예를 들어, '과학 기술 정보'를 포함한 문서 A와 '정보 과학'을 포함한 문서 B가 있을 때 질의가 '과학 정보'라면, 순서를 고려하지 않는 방식은 두 문서를 모두 찾지만 순서를 고려하는 방식에서는 문서 A만 찾게 된다.

† 정 회 원 : 인하대학교 컴퓨터정보공학부 교수
gnugi@inha.ac.kr

논문접수 : 2009년 12월 2일
심사완료 : 2010년 1월 1일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제3호(2010.3)

순서를 고려하지 않는 방식과 고려하는 방식은 서로 장단점이 존재한다. 순서를 고려하지 않는 방식은 순서를 고려하는 방식보다 다양한 문서를 찾기 때문에 광범위한 결과를 원하는 사용자에게 적합하고, 반대로 순서를 고려하는 방식은 구체적인 결과를 원하는 사용자에게 적합하다. 예를 들어, 만약 사용자가 '한국과학기술정보연구원'을 찾기 위해 '과학 정보'로 질의했다면 위의 예에서 '정보 과학'을 포함한 문서 B는 사용자에게 찾아주지 않는 것이 좋다.

이렇게 찾은 문서가 주어진 질의와 얼마나 연관이 있는지 적합성(relevance)을 판단하기 위해 근접도(proximity), PageRank[5], Hub & Authority[6] 등의 방법이 사용된다. 이 중 근접도는 주어진 키워드들이 문서상에서 가까이에 위치할수록 해당 문서에 높은 점수를 주는 방법이다. 위의 예에서, '과학 기술 정보'는 세 단어 안에 질의 '과학 정보'가 모두 나타났고 '정보 과학'은 두 단어 안에 모두 나타났으므로 문서 B가 문서 A보다 더 높은 근접도 점수를 갖게 된다. 물론 순서를 고려하는 방식에서는 문서 B의 근접도를 구하지 않는다.

근접도는 순서를 고려하지 않는 경우에 대한 연구 결

표 1 기호 정리

기호	설명
T	N개의 단어들이 나열된 문서. $T=T[1..N]$
w_i	질의를 구성하는 키워드. $1 \le i \le k$
s_i	w_i가 T에 나타난 횟수
o_{ij}	w_i의 오프셋들 중 j 번째 오프셋. $1 \le j \le s_i$
K_i	w_i에 대한 역 인덱스. $K_i=\{o_{i1},o_{i2},...,o_{is_i}\}$
n	k개의 키워드가 나타난 전체 횟수. $n=\sum_{i=1}^{k}s_i$
$[l,r]$	오프셋 l 부터 r 까지의 영역

과들[7-11]과 순서를 고려하는 경우에 대한 연구 결과들[12-14]이 모두 존재한다. 두 경우 모두 알려진 가장 효율적인 알고리즘들의 동작 시간은 $O(n\log k)$이다[10-14]. 이때 k는 질의에 포함된 키워드의 개수이고 n은 문서에서 키워드가 나타난 전체 횟수이다.

본 논문에서는 순서를 고려해서 근접도를 구하는 문제를 해결하는 보다 효율적인 알고리즘을 제시한다. 제시하는 알고리즘의 시간 복잡도는 $O(n)$으로 키워드가 문서에 나타난 횟수에만 비례한다. 또한 실험을 통해 이전 연구 결과[13,14]보다 $k=2$인 경우는 약 1.2배의 속도 향상을, $k=5$인 경우는 3배 이상의 속도 향상이 있음을 보인다.

본 논문의 구성은 다음과 같다. 우선 2장에서는 기본적인 용어 및 배경 지식을 살펴보고, 3장에서는 관련 연구를 살펴본다. 4장에서 순서를 고려하는 근접도에 대한 알고리즘을 제시하고, 5장에서는 실제 구현을 통한 실험 결과를 살펴본다. 마지막으로 6장에서 결론을 맺는다.

2. 용어 및 배경 지식

먼저 몇 가지 용어들을 정의한다. 문서 $T=T[1..N]$는 N개의 단어들이 나열된 것이고, 질의는 하나 이상의 키워드 w_1, w_2, ..., w_k로 구성된다. 키워드 w_i의 오프셋(offset)은 문서 T의 처음에서부터 w_i가 나타난 위치까지의 거리로 정의된다. 예를 들어, 문서가 I am tom일 때 tom은 세 번째 위치한 단어이므로 tom의 오프셋은 $3-1=2$가 된다. o_{ij}는 T에서 w_i의 오프셋들 중 j번째 오프셋을 의미한다. $K_i=\{o_{i1}, o_{i2},...,o_{is_i}\}$는 T에서 w_i의 s_i개 오프셋들을 정렬된 형태로 가진 리스트이다. 즉, $o_{i1}<o_{i2}<\cdots<o_{is_i}$를 만족한다. K_i를 T에서 w_i에 대한 역 *인덱스(inverted index)*라고 부른다. n은 $\sum_{i=1}^{k}s_i$, 즉 T에서 주어진 k개의 키워드가 나타난 전체 횟수이다.

일반적으로 검색 엔진들은 문서를 역 인덱스 구조로 저장한다. 즉, 각 키워드에 대해 해당 키워드가 나타나는 문서의 목록과 각 문서에서 해당 키워드가 나타나는 오프셋의 리스트를 저장하고 있어서, 질의가 들어오면 해당 키워드가 포함된 문서와 각 오프셋을 바로 찾을 수 있다.

예 1. T_1=한국 과학 기술 정보 연구원 정보……, T_2=정보 과학 저널……이라면 역 인덱스 구조는 그림 1과 같다. 만약 '과학 정보'라는 질의가 들어오면 w_1=과학, w_2=정보가 되고, 역 인덱스 구조에서 w_1과 w_2가 공통으로 가진 문서를 찾아보면 T_1과 T_2임을 알 수 있다. 또한, 문서 T_1에 대해서 $K_1=\{1\}$이고 $K_2=\{3,5\}$이며 $n=3$이 된다.

한편, 영역 $[l,r]$은 오프셋 l부터 오프셋 r까지 연속된 구간을 의미하고, 영역 $[l,r]$의 크기는 $r-l+1$로 정의된다.

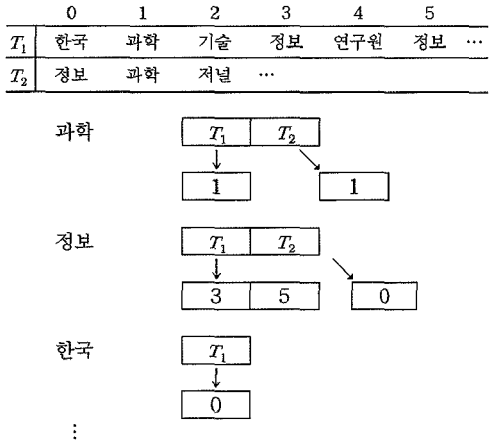

그림 1 역 인덱스 구조의 예

정의 1. *후보영역*이란 서로 다른 k개의 키워드가 모두 순서대로 나타나는 영역, 즉 $1 \le i < j \le k$이면 영역에 속하는 모든 오프셋 o_{ip}, $o_{jq}(1 \le p \le s_i, 1 \le q \le s_j)$에 대해 $o_{ip}<o_{jq}$를 만족하는 영역이다.

정의 2. *극소(minimal) 영역*이란 자신 이외의 다른 후보영역을 포함하지 않는 후보영역이다.

예 2. 예 1에 이어서, w_1=과학, w_2=정보에 대한 T_1에서의 후보영역은 [1,3]과 [1,5]가 되고, 극소영역은 [1,3]이 된다. 즉, T_1에는 '과학'과 '정보'를 순서대로 포함하는 크기 3(=3-1+1)인 영역이 존재하고 그 영역의 시작 오프셋은 1이다.

본 논문에서 다룰 순서를 고려하는 근접도 문제는 다음과 같다.

문제 1. k개의 서로 다른 키워드 w_1, w_2,..., w_k와 그

들의 역 인덱스 집합 $K=\{K_1, K_2,...,K_k\}$가 주어질 때, 크기가 가장 작은 극소영역, 즉, k개의 키워드가 모두 순서대로 나타나는 영역 중 가장 크기가 작은 영역을 찾으시오.

3. 관련 연구

우선 순서를 고려하지 않는 근접도와 관련된 연구 결과들은 다음과 같다. Gonnet, Baeza-Yates, Snider[7]는 문서에서 두 개의 키워드가 거리 d 이내에 나타나는 위치를 $O(n)$의 시간에 찾는 알고리즘을 제안하였다. Baeza-Yates와 Cunto[8]는 $O(n^2)$의 시간에 Proximity란 자료구조를 생성한 후 이를 이용하여 두 개의 키워드가 거리 d 이내에 나타나는 위치를 $O(\log n)$의 시간에 찾는 알고리즘을 제안하였다. Manber와 Baeza-Yates[9]는 $O(dn)$의 공간을 요하는 자료구조를 이용하여 두 개의 키워드가 거리 d 이내에 나타나는 위치를 역시 $O(\log n)$의 시간에 찾는 알고리즘을 제안하였다.

이상의 결과들은 키워드 개수가 두 개로, 그리고 최대 거리가 d로 제한되었지만, 다음 결과들은 이러한 제한이 없다. 먼저 Sadakane와 Imai[10]는 k-키워드 근접도 문제를 정의하고 $O(n\log k)$의 시간에 구하는 알고리즘을 제시하였다. *k-키워드 근접도 문제*는 $k(k\geq 2)$개의 서로 다른 키워드에 대한 근접도를 구하는 문제로, Sadakane와 Imai 알고리즘은 k개의 역 인덱스에서 하나씩 오프셋을 뽑아 이들을 포함하는 영역을 만든 뒤, 영역의 가장 왼쪽에 있는 오프셋을 빼고 그 키워드의 역 인덱스에서 다음 오프셋을 뽑아 영역에 추가하는 방식으로 근접도를 구한다. Kim, Lee, Park[11]은 *k-키워드 근접도 문제*를 확장하여 일반화된 k-키워드 근접도 문제를 정의하고 역시 $O(n\log k)$의 시간에 구하는 알고리즘을 제시하였다. 일반화된 k-키워드 근접도 문제는 k개의 서로 다른 키워드가 질의에 중복해서 나타날 수 있는 경우에 $k'(k'\leq k)$개의 키워드에 대한 근접도를 구하는 문제로, k개의 역 인덱스를 정렬한 뒤 영역을 찾는 방식으로 해결한다.

순서를 고려하는 근접도에 대한 연구도 일부 진행되었다. Lee와 Kim[12]은 일반화된 k-키워드 근접도 문제를 특정 두 키워드의 순서가 유지되도록 제한한 문제를 $O(n\log k)$의 시간에 구하는 알고리즘을 제시하였다. Gupta[13]는 k-키워드 근접도 문제를 k개 키워드의 순서가 유지되는 경우로 제한한 문제, 즉 문제 1을 $O(n\log k)$의 시간에 구하는 알고리즘을 제시하였다. Gupta의 알고리즘은 k개의 역 인덱스를 정렬한 뒤 k개 키워드가 순서대로 나오는 영역을 찾는 방식으로 근접도를 구한다. 앞으로 문제 1을 *k-키워드 순서 고려 근접도 문제*로 부르겠다. 한편, Gupta, Ozsoyoglu, Ozsoyoglu[14]는 k개 키워드의 순서가 유지되면서 각 키워드를 하나씩만 포함하도록 제한한 문제를 $O(n\log k)$의 시간에 구하는 알고리즘을 제시하였는데, 이는 Gupta[13]의 알고리즘을 거의 그대로 사용하고 있다. 이 문제는 제한된 *k-키워드 순서 고려 근접도 문제*로 부르겠다.

4. 알고리즘

이번 장에서는 k-키워드 순서 고려 근접도 문제를 푸는 알고리즘과 제한된 k-키워드 순서 고려 근접도 문제를 푸는 알고리즘을 제시한다. 두 알고리즘의 큰 틀은 역 인덱스를 보며 문제의 조건을 만족하는 극소영역을 차례대로 찾고 그 중 최소 크기를 갖는 영역을 최종 결과로 선택하는 것이다.

알고리즘 제시에 앞서 한 가지 용어를 더 정의한다.

정의 3. $t\leq k$인 t개의 서로 다른 키워드 w_1, w_2,..., w_t에 대해 문제의 조건을 만족하는 극소영역을 부분극소영역이라 부른다.

4.1 k-키워드 순서 고려 근접도 문제

k-키워드 순서 고려 근접도 문제를 푸는 알고리즘은 주어진 k개의 역 인덱스 $K_i(1\leq i\leq k)$를 각각 차례대로 보며 w_1부터 w_k까지 k개의 키워드가 순서대로 위치하는 영역 $[l,r]$을 찾는다. 이때 영역 $[l,r]$은 처음에는 w_1만 포함하는 크기 1인 부분극소영역이지만(즉, $r=l$), 알고리즘이 진행되면서 다음은 w_1과 w_2가 순서대로 위치하는 부분극소영역으로 확장되고 그 다음은 w_1부터 w_3까지가 순서대로 위치하는 부분극소영역으로 확장된다. 즉, 만약 현재 K_i를 보고 있다면 영역 $[l,r]$은 w_1부터 w_{i-1}까지 $i-1$개의 키워드가 순서대로 위치된 부분극소영역이 된다. 마지막으로 K_k를 본 후 영역 $[l,r]$이 확장되어 w_k까지 포함되면 하나의 극소영역을 찾은 것이다.

만약 w_{i-1}까지 포함된 부분극소영역 $[l,r]$이 w_i를 포함하는 부분극소영역으로 확장이 되지 않는다면, K_1에서 오프셋 l 다음의 오프셋 l'을 뽑아서 새로운 부분극소영역 $[l',l']$으로 만든 뒤 다시 확장 과정을 반복한다.

예 3. 그림 2에서, 현재 A와 B에 대한 부분극소영역 [0,1]이 있다면, 이 영역은 C에 대한 오프셋 3이 추가됨으로써 A, B, C에 대한 부분극소영역 [0,3]으로 확장이 된다.

예 4. 그림 2에서, 현재 A와 B에 대한 부분극소영역 [4,7]이 있다면, 이 영역은 C에 대한 오프셋 6에 의해 B와 C의 순서가 바뀌게 되어 확장되지 못한다. 이 경우, K_1에서 4의 다음 오프셋인 8을 기준으로 새로운 부분극

w_1=A, w_2=B, w_3=C

K_1	0	4	8
K_2	1		7
K_3		3	6

T = A B ? C A ? C B A

그림 2 부분극소영역이 확장되는 경우와 확장되지 못하는 경우의 예

소영역 [8,8]을 만들고 다시 확장 과정을 반복하게 된다. 구체적인 알고리즘은 다음과 같다.

1. $1 \le j \le k$인 j에 대해 $next_j$는 K_j의 첫 오프셋을 가리킨다. 최종 결과가 될 최소 크기의 극소영역 $MinRange$를 $[0,N]$으로 초기화한다.
2. $1 \le j \le k$인 j에 대해 하나라도 $next_j$가 존재하지 않으면 10번으로 간다. 그 외의 경우 i를 1로 두고, 스택 st를 비운다.
3. ind_i를 $next_i$로 두고, K_i에서 $next_i$의 다음 오프셋을 새로운 $next_i$로 둔다. 만약 더 이상의 오프셋이 K_i에 없다면 4번으로 간다. 만약 스택 st가 비어 있거나 $next_i$가 st의 top보다 작다면 $next_i$를 st에 push한다.
4. 만약 $i=k$라면 $[ind_1,ind_k]$가 극소영역이 되고, 극소영역의 크기가 $MinRange$의 크기보다 작다면 $MinRange$를 $[ind_1,ind_k]$로 대체한다. 이후 2번으로 간다.
5. 만약 $i<k$라면 i를 1 증가시킨다.
6. $next_i$가 ind_1보다 작은 동안 $next_i$의 다음 오프셋을 새로운 $next_i$로 업데이트한다. 만약 더 이상의 오프셋이 K_i에 없다면 10번으로 간다.
7. 만약 $next_i$가 부분극소영역 $[ind_1,ind_{i-1}]$에 포함된다면 2번으로 간다.
8. 만약 $next_i$가 스택 st의 top보다 크다면 st를 pop한다. Pop한 오프셋이 K_j에 속한다고 할 때, $j<i-1$이거나 $j=1$이거나 st의 top이 다시 $next_i$보다 작다면 2번으로 간다. 그 외의 경우, $next_j$가 $next_i$보다 작은 동안 $next_j$의 다음 오프셋을 새로운 $next_j$로 업데이트한다. 만약 더 이상의 오프셋이 K_j에 없다면 9번으로 간다. 만약 $next_j$가 st의 top보다 작다면 $next_j$를 st에 push한다.
9. 3번으로 간다.
10. $MinRange$를 출력하고 종료한다.

위 알고리즘의 정확성은 다음 보조정리들과 따름정리를 통해 보일 수 있다.

보조정리 1. 3번 단계가 끝났을 때 $[ind_1,ind_i]$는 i개의 키워드에 대한 부분극소영역이 된다.

증명. 먼저 $i=1$인 경우, $[ind_1,ind_i]=[ind_1,ind_1]$이므로 w_1의 오프셋을 하나 갖는 크기 1인 부분극소영역이 된다. 이때 스택 st에는 $next_1$이 저장된다.

$i=q-1$일 때 $[ind_1,ind_{q-1}]$이 $q-1$개에 대한 부분극소영역인 경우, $i=q$가 되면 w_q의 오프셋 $next_q$와 ind_1, ind_{q-1}, 스택 st와의 관계에 따라 여덟 가지 경우로 나누어 생각할 수 있다.(이때 st에는 $next_1$부터 $next_{q-1}$ 중 오프셋이 가장 작은 값이 top에 저장되어 있다.)

먼저 $next_q < ind_1$인 경우(그림 3(a)) $next_q$가 부분극소영역 $[ind_1,ind_{q-1}]$의 시작 오프셋보다 작으므로 K_q에서 그 다음 오프셋을 $next_q$로 해서 다시 경우를 고려한다.

두 번째, $next_q$가 $[ind_1,ind_{q-1}]$에 속하는 경우(그림 3(b)) w_{q-1}과 w_q의 순서가 뒤집히게 되어 $[ind_1,ind_{q-1}]$은 q개에 대한 부분극소영역이 될 수 없다. 따라서 현재 영역을 버리고 $i=1$부터 다시 시작한다.

세 번째, $next_q > ind_{q-1}$이고 $next_q < st.top$인 경우(그림 3(c)) 3번 단계로 가서 $ind_q = next_q$로 해서 $[ind_1,ind_{q-1}]$을 $[ind_1,ind_q]$로 확장하면 q개의 키워드에 대한 순서를 모두 만족하는 부분극소영역이 된다.

네 번째, $next_q > st.top$이고 $st.top$이 K_{q-1}의 오프셋이며 $next_q$가 st의 두 번째 top보다 작은 경우(그림 3(d)) 3번 단계로 가서 $ind_q = next_q$로 해서 $[ind_1,ind_{q-1}]$을 $[ind_1,ind_q]$로 확장하면 q개의 키워드에 대한 순서를 모두 만족하는 부분극소영역이 된다. 즉, w_{q-1}의 오프셋이 영역에 여러 번 포함되어도 q개 키워드의 순서에 위배되지 않는다.

다섯 번째 $next_q > st.top$이고 $st.top$이 K_{q-1}의 오프셋이며 $next_q$가 st의 두 번째 top보다 큰 경우(그림 3(e))와 여섯 번째 $next_q > st.top$이고 $st.top$이 K_{q-1}의 오프셋이 아닌 경우(그림 3(f))는 둘 다 $next_q$를 포함하도록 영역을 확장하면 w_{q-1}의 오프셋 다음에 w_p $(p<q-1)$의 오프셋이 오게 되어 순서에 위배된다. 따라서 현재 영역을 버리고 $i=1$부터 다시 시작한다.

일곱 번째, $next_q > st.top$이고 $q-1=1$인 경우(그림 3(g)) $next_q$를 포함하도록 영역을 확장하면 $[ind_1,next_q]$는 또 다른 후보영역인 $[next_1,next_q]$를 포함하게 되어 부분극소영역이 되지 않는다. 따라서 현재 영역을 버리고 $i=1$부터 다시 시작한다.

여덟 번째, 더 이상 w_q의 오프셋이 없는 경우(그림 3

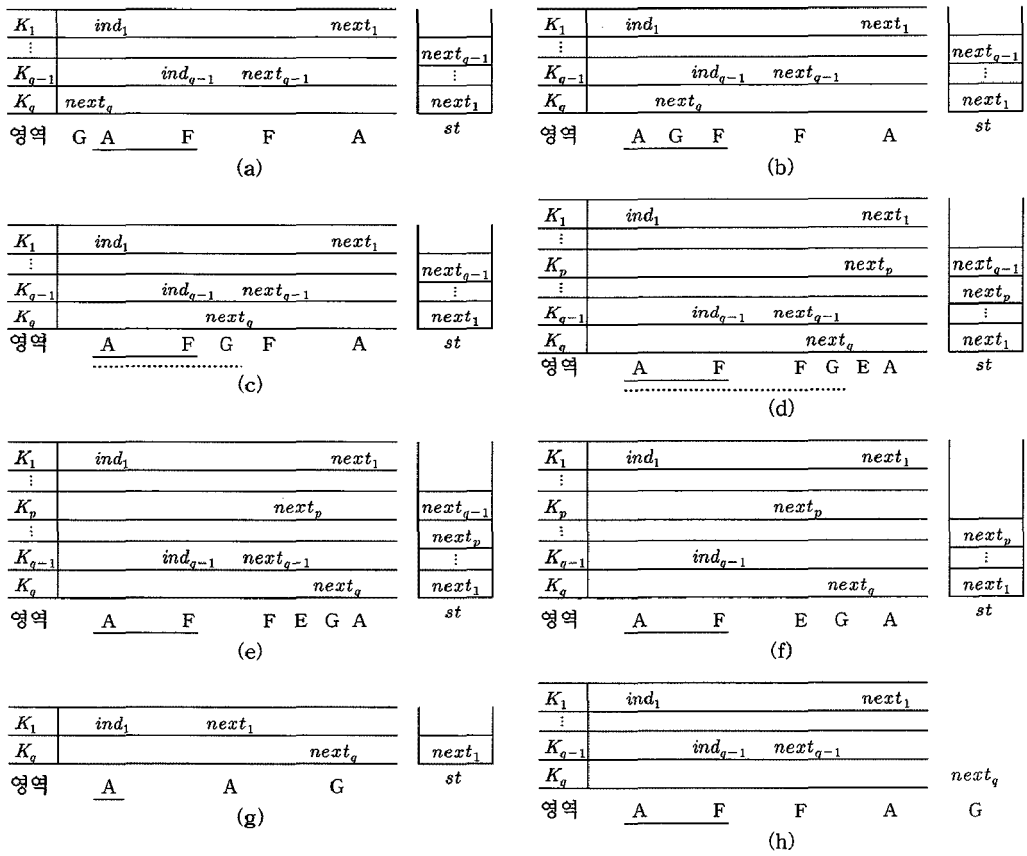

그림 3 보조정리 1의 증명. w_q의 오프셋 $next_q$와 ind_1, ind_{q-1}, 스택 st와의 관계에 따라 (a)부터 (h)까지 여덟 가지의 경우로 나뉨. 각 경우에서 실선 밑줄은 부분극소영역 $[ind_1, ind_{q-1}]$을 나타내고 (c)와 (d)의 점선 밑줄은 $[ind_1, ind_q]$를 나타냄. (h)는 더 이상 w_q의 오프셋이 없음을 의미함(w_1=A, w_{q-2}=E, w_{q-1}=F, w_q=G라고 가정)

(h)) q개에 대한 영역으로 확장할 오프셋이 존재하지 않으므로 q개 이상에 대한 부분극소영역이 존재하지 않는다. 이 경우, 더 이상 극소영역을 찾을 필요가 없으므로 알고리즘을 종료한다.

이상과 같이 $i=q$가 된 후 3번 단계에서 ind_i가 찾아지는 세 번째와 네 번째의 경우 모두 영역 $[ind_1, ind_i]$가 i개의 키워드에 대한 부분극소영역임이 증명되었다. □

보조정리 2. 4번 단계는 모든 극소영역을 찾는다.

증명. 4번 단계에서 찾는 영역이 극소영역임을 먼저 보인다. 보조정리 1에 의해 $[ind_1, ind_i]$는 i개의 키워드에 대한 부분극소영역이므로, $i=k$인 경우라면 결국 k개의 키워드에 대한 부분극소영역이 되고 정의에 의해 4번 단계에서 찾은 영역 $[ind_1, ind_k]$는 극소영역이 된다.

이제 4번 단계에서 찾지 못하는 극소영역이 없음을 보인다. 보조정리 1의 증명에 의해, 1개의 키워드에 대한 부분극소영역 $[ind_1, ind_1]$이 찾아지면 ind_1을 시작 오프셋으로 갖는 k개의 키워드에 대한 극소영역 $[ind_1, ind_k]$이 존재한다면 찾을 수 있음이 보장된다. 모든 극소영역은 순서 관계에 의해 항상 ind_1을 시작 오프셋으로 가지므로, K_1에 속하는 모든 오프셋이 1개의 키워드에 대한 부분극소영역으로 찾아짐만 보이면 증명이 끝난다. 그런데 알고리즘의 2번과 3번 단계를 보면 $i=1$인 경우에는 언제나(오프셋만 있다면) 부분극소영역 $[ind_1, ind_1]$을 찾고 있으며, $next_1$은 3번 단계에서만 변경되므로 빠지는 오프셋도 없다. 따라서 4번 단계에서는 모든 극소영역을 찾게 된다. □

따름정리 3. 위 알고리즘의 출력 *MinRange*는 k-키워드 순서 고려 근접도 문제의 조건을 만족하는 최소 크기의 극소영역이다.

증명. 보조정리 2에 의해 4번 단계는 모든 극소영역

을 찾고 그 중 가장 작은 크기의 영역을 *MinRange*로 유지하므로, 결국 *MinRange*는 k-키워드 순서 고려 근접도 문제의 조건을 만족하는 최소 크기의 극소영역이 된다. □

위 알고리즘의 복잡도는 다음 정리와 같다.

정리 4. k개 키워드의 오프셋의 개수의 합이 n일 때, 위 알고리즘은 $O(n+k)$의 공간을 사용하여 $O(n)$의 시간에 최소 크기의 극소영역을 찾는다.

증명. 먼저 공간 복잡도를 살펴본다. 역 인덱스 집합 K를 위해 $O(n)$의 공간이 필요하고, $1 \le i \le k$인 모든 i에 대해 ind_i와 $next_i$를 위해 $O(k)$의 공간이 필요하다. 스택 st에는 최대 k개가 들어갈 수 있으므로 $O(k)$의 공간이 필요하고, *MinRange*를 위해서는 상수공간이 필요하다. 따라서 전체 공간 복잡도는 $O(n+k)$가 된다.

다음 시간 복잡도를 살펴본다. 기본적으로 $next_i$가 K_i의 오프셋을 순서대로 한 번 씩 보게 되며, 모두 보고나면 알고리즘이 종료된다. 따라서 한 번의 오프셋을 보는 동안 상수시간의 작업이 필요하다면 전체 수행 시간은 $O(n)$이 된다.

알고리즘을 단계별로 살펴보자. 단계 1은 k의 시간이 걸리지만 k번의 오프셋을 보고 있으므로 오프셋 별로 상수시간이 걸린다. 단계 3은 상수 시간동안 한 번의 오프셋을 본다. 단계 4, 5도 상수시간 작업이다. 단계 6은 오프셋을 보는 횟수만큼 반복되므로 오프셋 별로 역시 상수시간이다. 단계 8은 조건에 따라 여러 상황이 발생하지만 오프셋을 업데이트하는 작업을 제외하고는 모두 상수시간 작업이다. 오프셋을 업데이트하는 작업도 단계 6과 마찬가지로 오프샛을 보는 횟수만큼만 반복되므로 상수시간이다. 결국 알고리즘에서 오프셋 별로 필요한 작업은 상수시간을 넘지 않는다. □

4.2 제한된 k-키워드 순서 고려 근접도 문제

제한된 k-키워드 순서 고려 근접도 문제를 푸는 알고리즘은 4.1에서 제시한 k-키워드 순서 고려 근접도 문제를 푸는 알고리즘과 거의 동일하다. 제한된 k-키워드 순서 고려 근접도 문제에서 극소영역은 각 키워드를 하나씩만 포함해야하므로, $ind_i\,(1 \le i \le k)$를 포함하도록 부분극소영역을 확장할 때 $\min_{1 \le j \le i}\{next_j\}$는 절대로 넘지 않도록 해야 한다. 이를 위해 4.1에서 제시한 알고리즘의 8번 단계를 아래와 같이 바꾸면 된다.

8. 만약 $next_i$가 스택 st의 top보다 크다면 2번으로 간다.

제안하는 알고리즘은 4.1의 알고리즘과 마찬가지로 $O(n+k)$의 공간을 사용하여 $O(n)$의 시간에 동작한다.

5. 실 험

순서를 고려하는 근접도에 대해 본 논문에서 제시하는 결과와 이전 연구 결과를 실험을 통해 비교해 본다. k-키워드 순서 고려 근접도 문제에 대해서는 4.1절의 알고리즘과 Gupta[13]의 알고리즘을 비교하고, 제한된 k-키워드 순서 고려 근접도 문제에 대해서는 4.2절의 알고리즘과 Gupta, Ozsoyoglu, Ozsoyoglu[14]의 알고리즘을 비교한다. 참고로, 순서를 고려하는 근접도와 고려하지 않는 근접도의 비교는 이전 연구 결과[13,14]를 참고하기 바란다.

실험 환경은 다음과 같다. 알고리즘들을 모두 C 언어로 직접 구현 후 gcc version 3.4.5로 컴파일 하였다. 하드웨어는 3.0GHz Intel Pentium 4 프로세서에 메모리는 1GB이며 운영체제는 Windows XP Service Pack 3이다.

실험은 다음과 같은 세 가지 경우로 진행하였다. 첫 번째는 각 키워드별로 5만개의 오프셋을 갖는 하나의 문서에 대해 키워드 개수를 2개부터 7개까지 바꿔가며 시간을 측정하였다. 두 번째는 각 키워드별로 5개의 오프셋을 갖는 1만개의 문서에 대해 키워드 개수를 2개부터 7개까지 바꿔가며 시간을 측정하였다. 세 번째는 키워드 개수가 5개인 경우에 대해 각 키워드별로 5개의 오프셋을 갖는 문서를 1만개부터 10만개까지 바꿔가며 시간을 측정하였다. 모든 데이터는 랜덤하게 역 인덱스를 생성하였다. 표 2는 각 실험에서 존재하는 극소영역의 개수를 나타낸다. 모든 실험은 1,000번 반복 후 평균 시간을 결과로 사용하였다.

표 2 실험별로 존재하는 극소영역의 개수. A는 k-키워드 순서 고려 근접도 문제에 대해, B는 제한된 k-키워드 순서 고려 근접도 문제에 대한 개수임

#키워드	키워드별 5만개 오프셋 × 1개 문서		키워드별 5개 오프셋 × 1만개 문서	
	A	B	A	B
2	25008	25008	21277	21277
3	8245	5466	7760	4500
4	1380	775	1182	645
5	150	88	141	77
6	6	4	13	4
7	0	0	0	0

5.1 k-키워드 순서 고려 근접도 문제

그림 4, 5, 6은 k-키워드 순서 고려 근접도 문제에 대한 세 가지 실험 결과는 각각 나타낸다. 그림 4는 키워드 별로 5만개의 오프셋을 갖는 하나의 문서에 대해 키워드 개수를 2개부터 7개까지 바꿔가며 시간을 측정한 결과로, 모든 경우에 Gupta의 알고리즘보다 빠르게 수행됨을 알 수 있다.

그림 5는 키워드 별로 5개의 오프셋을 갖는 1만개의 문서에 대해 키워드 개수를 2개부터 7개까지 바꿔가며 시간을 측정한 결과로, 역시 모든 경우에 Gupta의 알고리즘보다 빠르게 수행됨을 알 수 있다. 수행시간이 가장

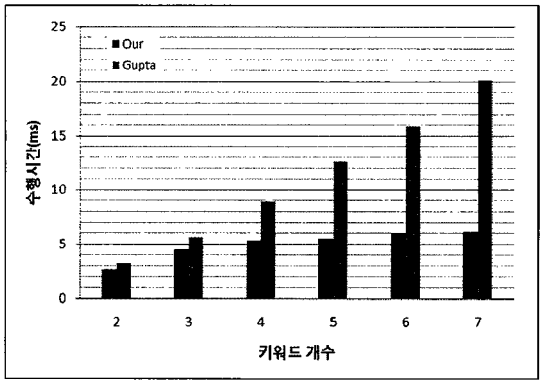

그림 4 키워드 별로 5만개의 오프셋을 갖는 한 문서에 대한 k-키워드 순서 고려 근접도 문제 실험

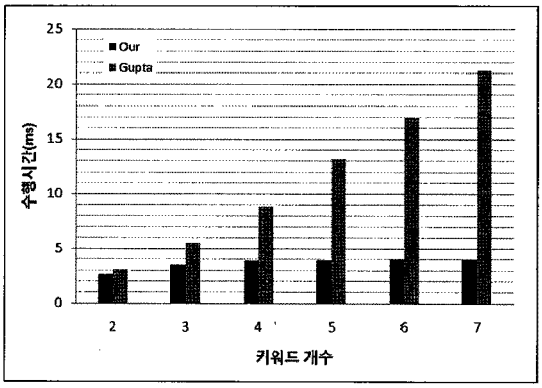

그림 5 키워드 별로 5개의 오프셋을 갖는 1만개 문서에 대한 k-키워드 순서 고려 근접도 문제 실험

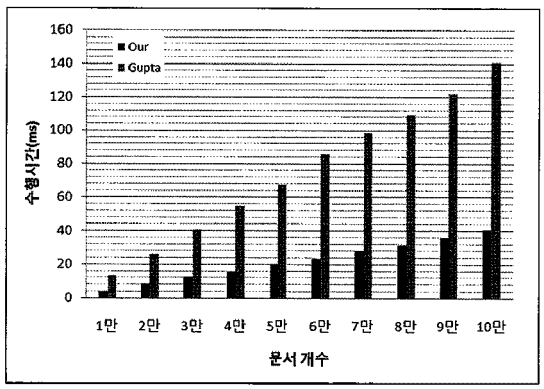

그림 6 키워드 개수가 5개인 경우에 대한 k-키워드 순서 고려 근접도 문제 실험

근접한 키워드 개수가 2개인 경우에도 약 1.2배의 속도 향상이 있다. 이 실험은 많은 문서를 처리하는 검색 엔진의 상황을 시뮬레이션 한 것으로, 주어진 키워드를 모두 포함하는 1만개의 문서에 대해 5ms 이내의 시간에 수행됨을 보인다.

특이한 점은 제안하는 알고리즘의 시간 복잡도는 전체 오프셋의 개수에 비례하지만 실제로는 키워드 개수가 늘어나도 수행 시간의 변화가 미미하다는 것이다. 실험데이터의 특성상 키워드 개수가 늘어나면 전체 오프셋 개수도 비례해서 늘어나는데도 수행 시간에 큰 변화가 없는 이유는, 키워드 개수가 늘어나도 문제의 조건을 만족하는 부분극소영역의 개수는 비례해서 늘어나지 않기 때문이다. 표 2에서 1만개 문서의 경우를 보면, 5개 키워드에 대한 극소영역이 141개임을 알 수 있다. 즉, 1만개 문서 중 최소 9859개의 문서는 5개 키워드에 대한 부분극소영역이 존재하지 않으므로, 키워드 개수가 6개, 7개로 늘어도 제안하는 알고리즘은 최소 9859개 문서에 대해서는 여섯 번째 키워드와 일곱 번째 키워드의 역인덱스를 전혀 보지 않는다.

그림 6은 키워드 개수가 5개로 고정한 경우에 키워드 별로 5개의 오프셋을 갖는 문서를 1만개부터 10만개까지 바꿔가며 시간을 측정한 결과로, 두 알고리즘 모두 문서 개수에 선형으로 비례함을 알 수 있다. 제안하는 알고리즘은 Gupta의 알고리즘보다 3배 이상 빠르게 동작하며, 주어진 키워드를 모두 포함하는 문서의 개수가 10만개인 경우에도 40ms에 수행됨을 알 수 있다.

5.2 제한된 k-키워드 순서 고려 근접도 문제

그림 7, 8은 제한된 k-키워드 순서 고려 근접도 문제에 대한 실험 결과는 나타낸다. 제한된 k-키워드 순서 고려 근접도 문제에 대한 알고리즘이 k-키워드 순서 고려 근접도 문제에 대한 알고리즘과 거의 동일한 만큼 실험 결과도 거의 유사하게 나왔다.

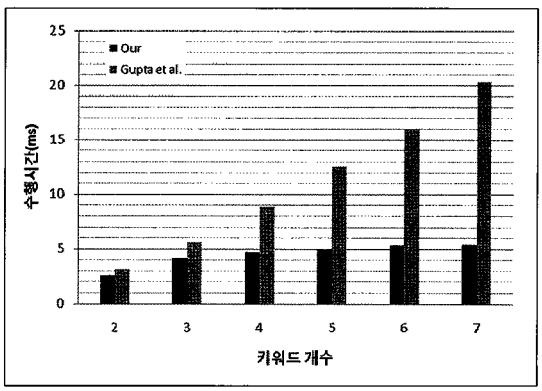

그림 7 키워드 별로 5만개의 오프셋을 갖는 한 문서에 대한 제한된 k-키워드 순서 고려 근접도 문제 실험

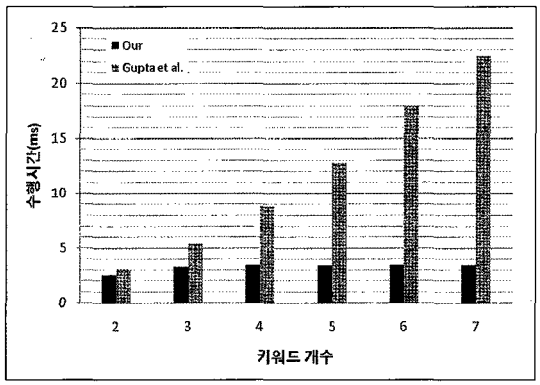

그림 8 키워드 별로 5개의 오프셋을 갖는 1만개 문서에 대한 제한된 k-키워드 순서 고려 근접도 문제 실험

6. 결론 및 향후 연구

본 논문에서는 k-키워드 순서 고려 근접도 문제와 제한된 k-키워드 순서 고려 근접도 문제를 해결하는 선형 시간 알고리즘들을 제시하였다. 이는 알려진 이전 연구 결과인 Gupta[13]의 $O(n\log k)$ 시간 알고리즘과 Gupta, Ozsoyoglu, Ozsoyoglu[14]의 $O(n\log k)$ 시간 알고리즘을 각각 $O(n)$ 시간으로 개선한 결과이다.

향후 연구로 본 논문의 알고리즘을 k개의 서로 다른 키워드가 질의에 중복해서 나타날 수 있는 경우에 k' ($k' \leq k$)개의 키워드에 대한 근접도를 구하는 문제인 일반화된 k-키워드 근접도 문제로 확장하는 것을 고려하고 있다.

참 고 문 헌

[1] Google, http://www.google.com.
[2] Naver, http://www.naver.com.
[3] Daum, http://www.daum.net.
[4] Yahoo, http://www.yahoo.com.
[5] S. Brin, L. page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, *Computer Networks and ISDN Systems*, 30(1-7), pp.107-117, 1998.
[6] J. Kleinberg, Authoritative Sources in a Hyperlinked Environment, *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.668-677, 1998.
[7] G.H. Gonnet, R. Baeza-Yates, T. Snider, New indices for text: PAT trees and PAT arrays, in *Information Retrieval: Algorithms and Data Structures*, ed. W. Frakes and R. Baeza-Yates, pp. 66-82. Prentice-Hall, 1992.
[8] R. Baeza-Yates, W. Cunto, The ADT proximity and text proximity problems, *Proc. IEEE String Processing and Information Retrieval Symposium*, pp.24-30, 1999.
[9] U. Manber, R. Baeza-Yates, An algorithm for string matching with a sequence of don't cares, *Information Processing Letters*, 37, pp.133-136, 1991.
[10] K. Sadakane, H. Imai, Fast algorithms for k-word proximity search, *IEICE Trans. Fundamentals*, E84-A(9), pp.312-319, 2001.
[11] S.-R. Kim, I. Lee, K. Park, A Fast Algorithm for the Generalized k-keyword Proximity Problem Given Keyword Offsets, *Information Processing Letters*, 91(3), pp.115-120, 2004.
[12] I. Lee, S.-R. Kim, An Algorithm for the Generalized k-Keyword Proximity Problem and Finding Longest Repetitive Substring in a Set of Strings, *Proc. of the 6th International Conference on Computational Science*, LNCS, 3994, pp.289-292, 2006.
[13] C. Gupta, Efficient k-Word Proximity Search, *MS Thesis*, CWRU, EECS Department, 2008.
[14] C. Gupta, G. Ozsoyoglu, Z.M. Ozsoyoglu. Efficient k-Word Proximity Search. *Proc. of the 24th International Symposium on Computer and Information Sciences*, pp.123-128, 2009.

김 진 욱
1998년 서울대학교 수학과 학사. 2000년 서울대학교 컴퓨터공학과 석사. 2006년 서울대학교 전기·컴퓨터공학부 박사. 2006년~2009년 ㈜에이치앰연구소 책임연구원. 2009년~현재 인하대학교 컴퓨터정보공학부 연구교수. 관심분야는 컴퓨터이론, 알고리즘, 웹검색, 생물정보학, 암호학