http://dx.doi.org/10.3745/KIPSTD.2011.18D.6.415

비트맵 필터를 이용한 효율적인 역 리스트 탐색 기법

권 인 택[†] · 김 종 익[††]

요 약

텍스트 데이터는 표현 방식의 차이, 타이핑 오류 등을 포함하고 있어 정확히 일치하는 검색으로는 유용한 정보를 얻기 어렵다. 따라서 유사도 기반 검색 방법이 많이 연구되고 있으며 효율적인 유사도 기반 검색을 위해 텍스트 데이터에 대한 역 리스트를 구성한다. 그리고 이를 병합하여 질의와 일정 기준 이상 유사한 데이터를 찾는다. 본 논문에서는 Suffix 필터링 과정에서 역 리스트의 탐색 비용을 줄이기 위해 역 리스트의 통계 정보인 비트맵 필터를 사용하는 기법을 제안한다. 제안하는 기법은 비트맵 필터를 사용하여 Suffix 필터링 과정에서 역 리스트의 탐색 여부를 결정하여 불필요한 역 리스트 탐색을 회피함으로써 역 리스트 병합 비용을 줄인다. 실험을 통하여 제안된 기법이 기존의 연구에서 제안된 Suffix 필터링 알고리즘보다 더 효율적임을 보인다.

키워드 : 유사 문자열 검색, 유사 검색, 비트맵 필터

Efficient Inverted List Search Technique using Bitmap Filters

Inteak Kwon[†] · Jongik Kim[††]

ABSTRACT

Finding similar strings is an important operation because textual data can have errors, duplications, and inconsistencies by nature. Many algorithms have been developed for string approximate searches and most of them make use of inverted lists to find similar strings. These algorithms basically perform merge operations on inverted lists. In this paper, we develop a bitmap representation of an inverted list and propose an efficient search algorithm that can skip unnecessary inverted lists without searching using bitmap filters. Experimental results show that the proposed technique consistently improve the performance of the search.

Keywords : String Similarity Search, Similarity Search, Bitmap Filter

1. 서 론

최근 인터넷의 발달로 인해 데이터의 양이 급격히 증가하고 있는 추세이며 인터넷 상에 존재하는 대부분의 정보는 취급하기 쉬운 텍스트 형태로 표현되어 있다. 텍스트 데이터는 표현방식의 차이, 타이핑 오류 등을 포함하고 있어 정확히 일치하는 검색만을 수행하는 경우 유용한 정보를 얻어내기가 매우 어렵다. 따라서 텍스트 데이터로부터 원하는 정보를 추출하기 위해 유사도를 기반으로 하는 검색 방법이 많이 연구되고 있다.

유사도를 기반으로 하는 텍스트 검색은 주어진 문자열 집합 내에서 질의 문자열과 유사한 모든 문자열을 찾아내는 문제로 정의할 수 있다. 유사 문자열 검색을 위해서는 서로 다른 두 문자열의 유사도를 측정할 수 있어야 한다. 문자열의 유사도를 측정하기 위해 편집거리, 코사인 유사도, 자카드 유사도와 같은 다양한 유사도 기준이 제안되었다. 기존의 연구에서는 문자열을 집합으로 변환한 후 이 유사도 기준들을 두 집합의 교집합 크기로 변환하여 사용한다[2]. 문자열을 집합으로 변환하기 위해 문자열을 특정 길이의 부분 문자열로 분해하는 방법이 가장 많이 사용된다[2, 8, 9, 10, 11]. 예를 들어 "information"라는 문자열을 길이 4인 부분 문자열로 분해하면 {info, nfor, form, orma, rmat, mati, atio, tion}의 부분 문자열 집합을 얻을 수 있다. 이때, 길이 q인 부분 문자열을 q-그램이라 부른다.

유사 문자열 검색은 주어진 문자열 집합 내의 각 문자열과 질의 문자열을 q-그램 집합으로 변환한 후에 각 문자열과 질의 문자열 사이의 교집합의 크기를 직접 계산함으로써 수행할 수 있다. 하지만, 주어진 문자열 집합이 매우 큰 경우 이러한 방법은 많은 연산을 필요로 하여 매우 비효율적이다.

※ 이 논문은 2011년 한국정보처리학회 춘계학술대회에 "비트맵 필터를 이용한 효율적인 유사 문자열 검색"[1]의 제목으로 발표된 논문을 확장한 논문임.
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초사업임(No.2011-0005411).
† 준 회 원 : 전북대학교 전자정보공학부 석사과정
†† 정 회 원 : 전북대학교 컴퓨터공학부 조교수(교신저자)
논문접수 : 2011년 8월 4일
수 정 일 : 1차 2011년 9월 15일
심사완료 : 2011년 9월 20일

효율적인 유사 문자열 검색을 위해 문자열 집합의 각 문자열들을 q-그램으로 분해하고 각 q-그램들에 대해 자신을 포함하는 모든 문자열들의 ID를 정렬된 리스트로 나타내게 한다. 이때, 이 리스트를 역 리스트라고 한다. 그리고 질의 처리 시 질의 문자열을 분해하여 q-그램들을 생성하고 이 q-그램들에 해당하는 역 리스트들을 병합한다. 병합된 리스트에서 문자열 ID가 나타난 횟수는 질의 문자열과 문자열 집합의 각 문자열들의 교집합의 크기가 된다. 따라서 역 리스트들을 병합함으로써 질의 문자열과의 교집합의 크기가 일정 이상인 문자열들을 효율적으로 찾을 수 있다.

본 논문에서는 역 리스트에 포함된 문자열 ID들에 대한 통계 정보인 비트맵 필터를 구성하는 기법과 이 비트맵 필터를 이용하여 불필요한 역 리스트 탐색을 회피하는 기법을 제안한다. 제안하는 기법은 불필요한 역 리스트 탐색을 회피함으로써 역 리스트의 탐색 시간을 감소시키고 이로 인해 유사 문자열 검색의 질의 처리 시간을 감소시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 배경 지식 및 관련 연구들을 소개하며 3장에서는 제안하는 역 리스트의 비트맵 필터에 대해 설명한다. 그리고 4장에서는 비트맵 필터를 이용한 역 리스트 탐색 기법에 대해 설명하고 5장에서는 비트맵 필터의 공간 비용을 감소시키는 기법에 대해 설명한다. 6장에서는 실험을 통하여 기존의 방법과 제안하는 방법의 성능을 비교하고 7장에서는 결론과 향후 연구에 대해 기술한다.

2. 배경 지식 및 관련 연구

본 장에서는 유사 문자열 검색과 관련된 연구들을 살펴본다. 먼저 2.1에서는 유사도 기반 질의 처리에 관한 내용들을 살펴보며 2.2에서는 그 외의 관련된 연구들을 살펴본다.

2.1 유사도 기반 질의 처리

유사도 기반 질의 처리에서는 다양한 유사도 판별 기준을 지원하기 위하여 여러 유사도 판별 기준 대신 교집합 크기를 유사도 기준으로 사용하며 이를 Overlap 유사도라고 한다. <표 1>은 유사도가 t 이상일 때 유사도 판별 기준에 따른 Overlap 유사도를 나타낸 것이다[2].

유사도 기반 질의 처리 과정은 먼저 일정 Overlap 유사도를 만족하는 문자열들을 선별하는 후보 생성 단계와 각각의 후보들이 일정 기준 이상의 유사도를 갖는지 확인하는 검증 단계를 거쳐 질의 결과를 얻는다. 자카드와 코사인 유사도는 두 집합 간의 Overlap 유사도가 α 이상일 때 반드시 t 이상의 유사도가 되어야 한다. 따라서 자카드와 코사인 유사도의 경우 검증 단계를 거치지 않아도 질의 결과를 생성할 수 있다.

〈표 1〉 유사도 판별 기준에 따른 Overlap 유사도

유사도	정의	Overlap유사도(α)
자카드	$\frac{(\lvert s_1 \cap s_2 \rvert)}{(\lvert s_1 \cup s_2 \rvert)}$	$\frac{t(\lvert s_1 \rvert + \lvert s_2 \rvert)}{1+t}$
코사인	$\frac{\lvert s_1 \cap s_2 \rvert}{\sqrt{\lvert s_1 \rvert \cdot \lvert s_2 \rvert}}$	$t\sqrt{\lvert s_1 \rvert \lvert s_2 \rvert}$
편집거리	편집연산수	$\max(\lvert s_1 \rvert, \lvert s_2 \rvert) - q \cdot t$

최근 연구들[3, 4, 5, 6, 7]에서는 적은 수의 후보를 생성하기 위해 Prefix 필터링[4]을 이용하고 있으며 Prefix 필터링의 결과를 Prefix를 제외한 나머지 부분인 Suffix를 이용하여 필터링하여 더 적은 수의 후보를 생성하고 있다.

특성 1. (Prefix 필터링 원리) q-그램들이 어떤 정렬 기준 O에 의해서 정렬된 두 집합 S_1과 S_2를 고려해보자. 집합의 제일 앞에 위치한 q-그램 k개를 k-Prefix라고 하면, $|S_1 \cap S_2| \geq \alpha$ 일 때 S_1의 $(|S_1|-\alpha+1)$-Prefix와 S_2의 $(|S_2|-\alpha+1)$-Prefix는 적어도 하나의 q-그램을 공유한다.

최근 연구에서는 Prefix 필터링에서 사용하는 정렬 기준 O를 문자열 집합 내에서 q-그램의 발생 빈도에 대한 오름차순으로 하여 Prefix 필터링으로 생성되는 후보의 수를 줄인다.

Prefix 필터링을 적용하기 위하여 먼저 문자열 집합에 대한 역 색인을 구성한다. 그리고 질의의 유사도 기준을 이용하여 질의의 Prefix를 조사하고 Prefix에 포함되는 q-그램들이 가리키는 역 리스트들을 추출한다. 이렇게 추출된 역 리스트에 포함된 문자열들은 Prefix에서 하나 이상의 q-그램들이 질의 문자열의 q-그램들과 공통이므로 후보로 선별된다. 이렇게 선별된 후보들은 검증 단계에서 유사도를 측정하여 질의로 주어진 유사도 기준을 만족하는지 확인된다.

Suffix를 이용하여 Prefix 필터링을 통하여 생성된 후보들 중에서 Overlap 유사도 기준을 만족하는 문자열들을 선별함으로써 더 작은 크기의 후보 집합을 생성할 수 있다. 이때 Suffix를 이용하여 Prefix 필터링에 의해 생성된 후보 집합의 크기를 더 줄이는 과정을 Suffix 필터링이라 한다. Prefix 필터링을 통하여 생성된 후보들을 Suffix에 포함된 역 리스트에 탐색하여 포함 여부를 확인하고 Prefix와 Suffix에서의 발생 빈도를 이용하여 Overlap 유사도를 만족하는 후보를 선별한다. 이때 Suffix에 포함된 역 리스트들은 길이가 길기 때문에 이진 탐색하여 역 리스트의 탐색 비용을 줄인다[8, 9 10, 11].

(그림 1)은 유사 문자열 검색의 후보 생성 과정을 의사 코드로 기술한 것이다. 05~08번 줄은 질의 문자열의 Prefix를 이용하여 질의 결과에 대한 후보를 생성하는 과정이다. 먼저 06번 줄에서는 질의 문자열의 Prefix가 가리키는 역 리스트들을 역 색인에서 추출하며 07~08번 줄에서는 질의 결과가 될 수 있는 후보 문자열들을 선별한다. 09~13번 줄은 Overlap 유사도를 이용하여 Prefix 필터링을 통하여 생성된 후보 집합의 크기를 줄인다. 먼저 Suffix로 분류된 q-그램들이 가리키는 역 리스트들을 이진 탐색하여 특정 문자열 ID가 포함되어 있는지 확인한다. 그 후 특정 문자열 ID가 질의 문자열의 q-그램들이 가리키는 역 리스트에서 α번 이상 나타나는지 확인하여 Overlap 유사도를 충족하는지 확인한다.

알고리즘1 후보 생성 알고리즘

input : Query string qs and overlap threshold α
output : result set

```
01. Convert query string qs into q-gram set Q ;
02. Let Cands be the empty map from record id to int;
03. Let R be the empty set of ids ;
04. Calculate prefix size ps ;
05. for  i := 1 to ps
06.     Let PrefixList be the inverted list pointed by Q[i];
07.     foreach id in PrefixList
08.         Cands[id] := Cands[id] + 1;
09. foreach id in Cands
10.     for  j := ps + 1 to length(Q)
11.         Let SuffixList be the inverted list pointed by Q[j];
12.         if binarySeach( SuffixList, id ) then Cands[id]++;
13.     if Cands[id] >= α  then put id into R ;
14. RETURN R ;
```

(그림 1) 후보 생성 알고리즘

2.2 기타 관련 연구

[2, 8]에서는 유사 문자열 집합의 질의 결과 후보를 빠르게 생성하기 위해 효율적으로 역 리스트를 병합하는 알고리즘을 제안하였다. 또한 [7]에서는 문자열 집합에 대한 역 색인의 공간 비용을 줄이기 위한 방법을 제안하고 있으며 문자열 집합의 q-그램 발생 빈도를 이용하여 q의 범위를 특정 범위로 확장하는 VGRAM[10, 11]이 제안되었다. ED-Join[3]에서는 Prefix 필터링을 수행하는데 필요한 Prefix의 길이를 줄여 더 작은 크기의 후보 집합을 생성하는 방법은 제안하고 있다.

VGRAM은 q를 특정 범위로 확장하기 때문에 질의 문자열의 VGRAM을 얻기 위하여 별도의 Trie형태의 그램 사전이 필요하며 또한 편집거리를 Overlap 유사도로 변환하기 위해 별도의 알고리즘을 수행하여야 한다. 하지만 질의 문자열에 의해 추출되는 역 리스트의 수가 q-그램을 사용하는 경우보다 줄어들기 때문에 역 리스트를 병합하는 비용을 감소 할 수 있다. ED-Join의 경우 Prefix의 길이를 줄이기 때문에 상대적으로 Suffix의 길이가 길어지게 되며 Suffix에 포함되는 역 리스트의 수가 많아져 Suffix 필터링 과정에서 역 리스트를 탐색하는 비용이 상대적으로 증가하게 된다. 따라서 질의 처리 시간 중 Suffix 필터링 시간이 증가하게 되고 이를 효율적으로 감소시키는 방법이 필요하다.

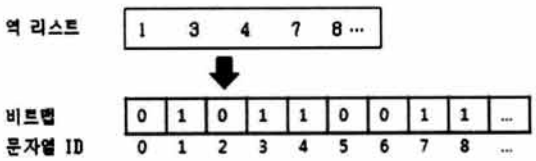
3. 역 리스트의 비트맵 필터

유사 문자열 검색의 경우 Prefix 필터링을 수행하여 생성된 후보 집합의 크기를 Suffix 필터링을 통하여 더 줄일 수 있다. ED-Join[3]과 같이 Prefix의 길이를 줄이게 되면 상대적으로 Suffix가 길어지게 되므로 이진 탐색해야 하는 역 리스트의 수가 많아져 Suffix 필터링의 비용이 증가하게 된다.

본 장에서는 Suffix 필터링의 비용을 줄이기 위하여 역 리스트에 대한 통계 정보인 비트맵 필터를 구성하는 방법에 대해 설명한다. 먼저 3.1절에서는 역 리스트를 비트맵으로 나타내는 방법에 대해 설명하며 3.2절에서는 역 리스트에 대한 비트맵 필터를 구성하는 방법에 대해 설명한다.

3.1 역 리스트의 비트맵 표현

Suffix 필터링에서는 Suffix로 분류된 역 리스트들이 특정 문자열 ID를 포함하고 있는지 확인하기 위하여 이진 탐색을 이용한다. 이 때 Prefix 필터링에 의해 생성된 모든 후보들을 이진 탐색하여 Suffix로 분류된 역 리스트에 포함되어 있는지 확인하기 때문에 Suffix 필터링의 비용이 매우 크다. 하지만 역 리스트를 길이가 문자열 집합의 크기와 같은 비트맵으로 나타낼 경우 역 리스트에 특정 문자열 ID가 포함되어 있는지 확인하기 위해 이진 탐색 대신 비트 연산을 사용할 수 있다. 비트 연산은 상수 시간에 수행되기 때문에 이진 탐색보다 더 효율적이다. 따라서 이진 탐색 대신에 비트연산을 사용하면 Suffix 필터링의 비용을 줄일 수 있다.



(그림 2) 역 리스트의 비트맵 표현

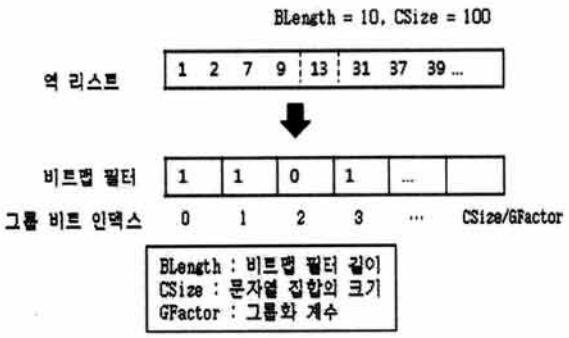
비트 연산을 사용하여 Suffix 필터링을 수행하기 위해서는 역 리스트를 비트맵으로 표현해야 하고 (그림 2)와 같이 비트맵으로 표현할 수 있다. (그림 2)의 역 리스트가 문자열 ID 1, 3, 4, 7, 8을 포함하고 있으므로 비트맵의 1, 3, 4, 7, 8번째 비트만 1로 표시한다. 이때 비트맵의 길이가 문자열 집합의 크기가 되며 각 비트는 해당하는 문자열 ID가 역 리스트에 포함되어 있는지를 나타낸다. 역 리스트 대신 이러한 비트맵을 구성하면 비트맵의 해당 비트가 1인지 아닌지를 판별하여 역 리스트에 해당 문자열 ID가 포함되어 있는지 확인할 수 있다. 문자열 ID가 해당 비트의 위치 정보가 되므로 상수 시간 내에 포함 여부를 확인할 수 있다. 따라서 이진 탐색을 통하여 문자열 ID가 역 리스트에 포함되어 있는지 확인하는 것보다 더 효율적이다.

임의의 질의를 처리하기 위해서는 이와 같은 비트맵을 모든 역 리스트에 대해 구성해야 한다. 하지만 일반적으로 문자열 집합의 크기가 매우 크기 때문에 비트맵의 길이 또한 매우 길어지게 된다. 또한 문자열 집합에 대한 역 색인을 구성하면 매우 많은 수의 역 리스트가 생성된다. 따라서 모든 역 리스트들에 대하여 비트맵을 구성하는 것은 많은 공간비용을 요구하게 된다.

3.2 역 리스트의 비트맵 필터

3.1에서 설명한 비트맵은 그 길이가 문자열 집합의 크기와 같으므로 역 리스트를 비트맵으로 표현하는데 많은 공간 비용이 든다.

본 절에서는 이러한 비트맵의 공간 비용을 줄이기 위해 비트맵의 길이를 고정하고 문자열 ID를 비트맵 길이만큼의



(그림 3) 역 리스트에 대한 비트맵 필터

그룹으로 나누어 각 그룹에 하나의 비트만 할당하는 비트맵 필터에 대해 설명한다.

비트맵 필터의 길이를 BLength라고 하고 문자열 집합의 크기를 CSize라고 할 때 문자열 ID들은 BLength만큼의 그룹으로 나누어진다. 이 때 한 그룹으로 나누어지는 문자열 ID의 수를 그룹화 계수 GFactor라 하고 CSize/BLength로 계산된다. 따라서 그룹화 계수만큼의 문자열 ID들은 역 리스트에 포함 여부를 나타내기 위해 하나의 비트를 공유하게 된다. 이때 공유되는 비트를 그룹 비트라고 부른다. 각 그룹 비트는 그룹 비트를 공유하는 문자열 ID들 중 한 개 이상이 역 리스트에 포함되어 있는 경우 그룹 비트는 1로 표시되며 그렇지 않은 경우는 0으로 표시된다.

(그림 3)과 같은 경우 CSize가 100이고 BLength가 10이므로 GFactor는 10이 된다. 따라서 10개의 문자열 ID들이 하나의 비트를 공유하게 된다. (그림 3)처럼 비트맵 필터의 0번째 비트는 문자열 ID 0부터 9까지의 포함 여부를 나타내며 1,2,7,9가 포함되어 있으므로 1로 표시된다. 하지만 2번째 비트의 경우 문자열 ID 20부터 29까지의 포함 여부를 나타내지만 해당하는 문자열 ID들이 모두 역 리스트에 포함되어 있지 않으므로 0으로 표시된다.

(그림 4)는 앞서 설명한 비트맵 필터의 구성 방법을 기술한 알고리즘이다. 2번 줄에서는 각각의 역 리스트에 BLength만큼의 길이를 갖는 비트 배열을 할당하고 3번 줄에서는 모든 비트를 0으로 초기화한다. 그리고 나서 04~06번 줄에서 0으로 초기화된 비트맵 필터를 역 리스트에 포함된 문자열 ID들의 그룹 비트들을 1의 값을 갖도록 해준다. 이 과정 중에서 05번 줄은 특정 문자열 ID의 그룹 비트를 결정하며 06번 줄은 해당하는 그룹 비트를 1의 값을 갖도록 만든다.

문자열 집합의 크기가 고정되어 있으므로 그룹화 계수는 오직 비트맵 필터의 길이의 영향만 받는다. 비트맵 필터의 길이가 짧으면 그룹화 계수는 커지게 되고 하나의 그룹 비트를 공유하는 문자열 ID가 많아진다. 하나의 그룹 비트를 공유하는 문자열 ID가 많아지면 비트맵 필터가 표현하는 정보의 신뢰성이 떨어지게 된다. 따라서 불필요한 역 리스트 탐색을 효율적으로 회피 할 수 없다. 이러한 이유로 비트맵 필터의 길이는 Suffix 필터링 시간과 질의 처리 시간에 영향을 미치게 된다.

알고리즘2 비트맵 필터 구성 I

```
input : Set of Inverted list I
        and Length of Bitmap Filter BLength
01. foreach inverted list List in I
02.     BFilter := bit array of size BLength;
03.     Initialize BFilter to 0's
04.     foreach ID in List
05.         Group := ID / CSize * BLength ;
06.         Set BFilter[Group] to 1;
07.     Attach BFilter to List ;
```

(그림 4) 역 리스트에 대한 비트맵 필터 구성

알고리즘3 비트맵 필터를 이용한 Suffix 필터링

```
input  : candidate list C, q-gram list SUFFIX, overlap a,
         and inverted lists I
output : ids that appear at least  times on lists
01. Initialize a result set R to be empty;
02. Initialize a inverted lists set L  to be empty;
03. Put inverted lists such that pointed by elements
    of SUFFIX  into L ;
04. foreach c in candidate list C
05.     Let count be the number of occurrence of c
        in PREFIX;
06.     Lon : = 0;
07.     foreach list in L
08.         if group bit of c . id is 1 then
09.             increase Lon by 1;
10.     if count + Lon < a  then continue;
11.     foreach list in L
12.         Check if c . id appear on this list
            using binary search;
13.         if c . id appears a or more times among
            the all lists, add c.id to R ;
14. RETURN R ;
```

(그림 5) 비트맵 필터를 이용한 Suffix 필터링 알고리즘

4. 비트맵 필터를 이용한 Suffix 필터링

앞서 언급한 비트맵 필터를 구성하면 그룹 비트가 0일 경우에는 역 리스트에 문자열 ID가 포함될 가능성이 없으므로 그룹 비트를 공유하는 문자열 ID들은 역 리스트를 탐색하지 않아도 된다. 따라서 비트맵 필터를 이용하여 불필요한 역 리스트의 탐색을 회피할 수 있다.

비트맵 필터를 이용하여 문자열 ID가 역 리스트에 포함되어 있는지 확인하기 위해서 그 문자열 ID가 속한 그룹을 찾아야 하며 문자열 ID/GFactor번째 그룹에 포함된다. 문자열 ID가 속한 그룹을 찾은 후에 비트맵 필터에서 그룹의 비트를 확인한다. 만약 그룹 비트가 1인 경우에는 역 리스트에 문자열 ID가 포함될 가능성이 있으므로 역 리스트를 이진 탐색하여 해당 문자열 ID의 포함 여부를 확인한다. 그렇지 않은 경우에는 역 리스트에 문자열 ID가 포함되어 있지 않으므로 역 리스트를 탐색하지 않는다.

(그림 5)는 비트맵 필터를 이용하는 역 리스트 탐색 기법을 적용한 Suffix 필터링 알고리즘을 기술한 것이다. (그림 5)의 07~09번 줄에서는 Suffix에 포함된 역 리스트들 중 비

트맵 필터의 그룹 비트가 1인 역 리스트들의 수를 확인하여 문자열 ID가 역 리스트에 포함될 가능성이 있는 역 리스트의 수를 확인한다. 10번 줄에서 비트맵 필터를 통해 확인된 문자열 ID가 포함될 가능성이 있는 역 리스트의 개수와 Prefix에서 문자열 ID가 발생된 횟수의 합은 질의 문자열의 q-그램이 가리키는 역 리스트들 중에서 문자열 ID가 발생할 수 있는 최대값이 된다. 그러므로 이 값이 α보다 작은 경우에는 남아있는 역 리스트들을 탐색하지 않아도 후보가 될 수 없음을 알 수 있다. 따라서 남아있는 역 리스트들을 탐색하지 않음으로써 불필요한 역 리스트 탐색을 회피한다. 11~13번 줄에서는 비트맵 필터를 이용하여 역 리스트 탐색을 회피하지 못하였을 경우 Suffix에 해당하는 역 리스트들을 이진 탐색하여 문자열 ID가 역 리스트에 포함되어 있는지 확인한다. 그리고 Overlap 유사도를 만족하는 문자열 ID를 후보 집합에 포함시킨다.

5. 비트맵 필터의 공간 비용 감소 기법

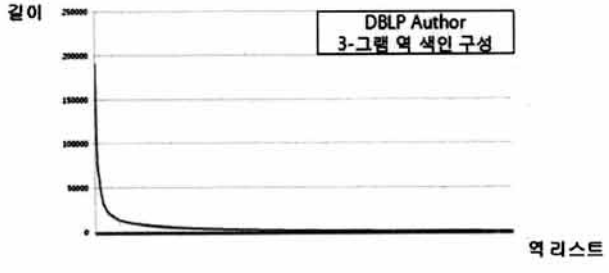
본 장에서는 문자열 집합에 대한 역 리스트 길이 분포를 이용하여 비트맵 필터의 공간 비용을 감소시키는 기법에 대해 설명한다.

〈표 2〉 데이터 집합의 역 리스트 수

데이터 집합	역 리스트의 수
DBLP Author	24624
DBLP Title	59314
Google Web Corpus	69909

비트맵 필터를 이용하는 역 리스트 탐색을 위해서는 문자열 집합에 대한 모든 역 리스트가 비트맵 필터를 가지고 있어야 한다. 하지만 <표 2>와 같이 문자열 집합에 의해 생성되는 역 리스트의 수는 매우 많다. 따라서 문자열 집합에 대한 모든 역 리스트들에 대해 비트맵 필터를 구성하는 것은 많은 공간 비용이 든다.

일반적으로 문자열 집합에 대한 역 리스트들의 길이는 Power-law 분포를 따른다. (그림 6)은 DBLP Author 데이터 집합을 3-그램으로 역 색인을 구성하였을 때 역 리스트들의 길이 분포를 나타낸 것이다. (그림 6)을 살펴보면 일부의 역 리스트들은 매우 길고 나머지 대부분의 역 리스트들은 길이가 매우 짧음을 알 수 있다.



(그림 6) 역 리스트의 길이 분포

문자열 집합에 대한 역 리스트들 중 길이가 긴 역 리스트들이 질의 처리 과정에서 Suffix에 포함될 가능성이 높으며 비트맵 필터는 Suffix에 포함된 역 리스트들을 탐색할 때만 이용된다. 따라서 길이가 긴 일부의 역 리스트들만 비트맵 필터를 구성하여 비트맵 필터의 공간 비용을 감소시킬 수 있다. 이때 전체 역 리스트의 수에 대한 비트맵 필터를 구성하는 역 리스트의 수의 비율을 비트맵 필터 구성 비율 β이라 한다.

(그림 7)은 비트맵 필터 구성 비율이 β일 때 역 리스트의 비트맵 필터를 구성하는 알고리즘을 기술한 것이다. (그림 7)의 03번 줄에서는 가장 긴 역 리스트부터 우선적으로 비트맵 필터를 구성하기 위하여 역 리스트들을 길이 순으로 정렬한다. 04~10번 줄에서는 길이가 가장 긴 N개의 역 리스트들을 선정하여 비트맵 필터를 구성하는 과정이다. 이 과정 중에서 06~10번 줄은 그림4와 동일하다.

알고리즘5 비트맵 필터 구성 II

input : Set of Inverted list I and bitmap filter construction rate β

```
01. Let B empty map for inverted list and bitmap filter pair;
02. N := |I| * β
03. sort inverted lists in I in decreasing order of their length;
04. for i := 1 to N
05.     List := i-th inverted list in I;
06.     Create a bitmap filter BFilter and Initialize BFilter
        to 0's;
07.     foreach ID in List
08.         Group = ID / CSize * BLength ;
09.         BFilter[Group] := 1;
10.     Attach BFilter to List ;
```

(그림 7) 역 리스트에 대한 비트맵 필터 구성

(그림 7)과 같이 역 리스트의 비트맵 필터를 구성하는 경우 역 리스트 병합 과정에서 비트맵 필터를 가지지 않은 역 리스트가 Suffix로 분류 될 수 있다. 이러한 경우 역 리스트를 탐색할 때 비트맵 필터를 이용할 수 없기 때문에 이진 탐색에 의해 특정 문자열 ID가 역 리스트에 포함되었는지 판별해야 한다. 따라서 비트맵 필터를 가지지 않은 역 리스트가 Suffix로 분류되면 모든 그룹 비트가 1인 비트맵 필터를 가진 것으로 간주해야 한다.

(그림 8)은 비트맵 필터를 가지지 않은 역 리스트들을 고려한 Suffix 필터링을 위한 알고리즘3의 수정 부분이다. (그림 8)의 02번 줄에서는 역 리스트에 대한 비트맵 필터가 구성되었는지를 확인하며 만약 비트맵 필터가 구성되어 있지 않은 경우 03번 줄에서 이진탐색을 통하여 특정 문자열 ID의 포함 여부를 확인해야 하므로 해당 그룹 비트가 1인 비트맵 필터를 가진 것처럼 간주한다. 04번 줄에서는 비트맵 필터를 구성 되었을 경우 특정 문자열 ID의 그룹 비트를 확인하고 05번 줄에서 그룹 비트가 1인 경우 탐색해야 하는 역 리스트의 수를 증가 시킨다.

알고리즘6 알고리즘3의 07~09번 줄 교체 부분

```
01.  foreach list in L
02.      if list has no bitmap filter then
03.          increase Lon by 1;
04.      else if group bit of c.id is 1 then
05.          increase Lon by 1;
```

(그림 8) 비트맵 필터를 가지지 않은 역 리스트를 고려한 Suffix 필터링 I 알고리즘

6. 실 험

본 장에서는 기존의 역 리스트 병합 알고리즘과 제안한 비트맵 필터를 사용하는 역 리스트 병합 알고리즘의 성능을 실험을 통해 비교한다. Intel Dual-Core 3GHz, 2GB RAM, Ubuntu 10.04 LTS 32bit의 환경에서 수행하였고 3-그램을 사용하여 역 리스트들을 구성하였고 Prefix의 크기는 ED-Join [3]에서 제안한 방법으로 계산하였다. 질의로 주어지는 유사도는 편집거리 1~3으로 실험하였으며 1000개의 질의를 처리하는 시간을 측정하였다. 실험에 사용된 데이터는 <표 3>과 같다.

〈표 3〉 실험 데이터 집합

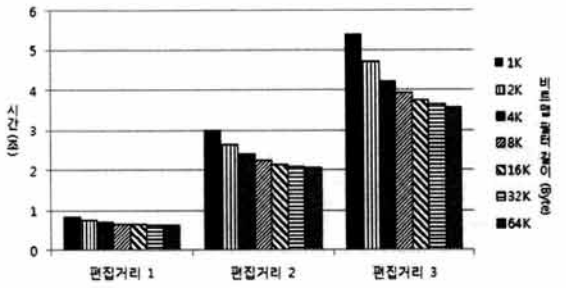
데이터 집합	문자열 개수	문자열 평균 길이
DBLP Author	2948930	15
DBLP Title	1158649	68
Google Web Corpus	3000000	21

6.1 비트맵 필터 길이에 따른 질의 처리 시간

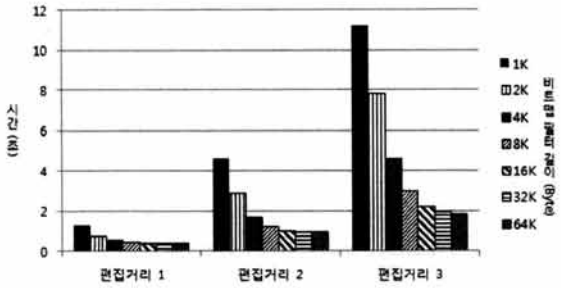
(그림 9)는 비트맵 필터의 길이가 질의 처리 시간에 미치는 영향을 보여주고 있다. 비트맵 필터의 길이가 짧은 경우에는 그룹화 계수가 너무 크기 때문에 비트맵 필터가 나타내는 정보의 신뢰도가 낮아진다. 따라서 비트맵 필터를 이용하여 불필요한 탐색의 회피가 효과적으로 이루지지 않고 오히려 비트맵 필터를 확인하는 비용이 추가되므로 비트맵 필터를 사용하지 않은 경우보다 질의 처리 시간이 더 길다. 하지만 비트맵 필터의 길이가 충분히 클 경우에는 비트맵 필터가 나타내는 정보의 신뢰도가 향상되므로 비트맵 필터에 의한 불필요한 탐색 회피가 효과적으로 이루어진다. 따라서 질의 처리 시간이 감소하게 된다. (그림 9)를 통하여 편집거리가 커질수록 비트맵 필터에 의한 불필요한 탐색 회피 효과가 커지는 것을 알 수 있다.

6.2 비트맵 필터 구성 비율에 따른 비트맵 필터 이용률

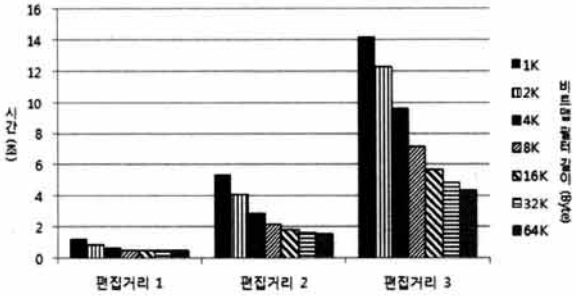
비트맵 필터의 공간 비용을 줄이기 위하여 질의 처리 과정에서 Suffix에 포함될 가능성이 높은 일부의 역 리스트들에 대해서만 비트맵 필터를 구성하는 방법을 제안하였다. (그림 10)은 비트맵 필터를 구성하는 비율에 따른 Suffix 필터링의 비트맵 필터 이용률을 측정한 것이다. (그림 10)은 편집거리가 커질수록 낮은 비트맵 필터 구성 비율에서도 비트맵 필터 이용률이 1에 도달함을 보여주고 있다. 따라서 비트맵 필터 구성 비율을 편집거리 1일 때 비트맵 필터 이용률을 1이 되는 지점으로 선택하여도 편집거리가 1 이상인 경우 비트맵 필터 이용률이 낮아지지 않는다. (그림 10)에서 모든 실험 데이터 집합에 대하여 비트맵 필터 구성 비율 β가 0.11일 때 비트맵 필터 이용률이 1에 도달하고 있으므로 전체 역 리스트들 중 가장 긴 11%만 비트맵 필터를 구성하여 비트맵 필터의 공간 비용을 줄일 수 있다.



(a) DBLP-Author



(b) DBLP-Title



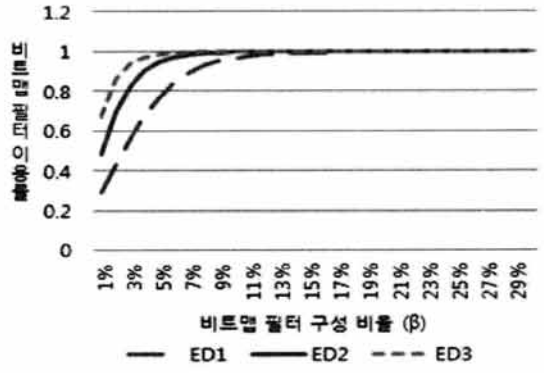
(c) Google Web Corpus

(그림 9) 비트맵 필터 길이에 따른 Suffix 필터링 시간 (β = 0.11)

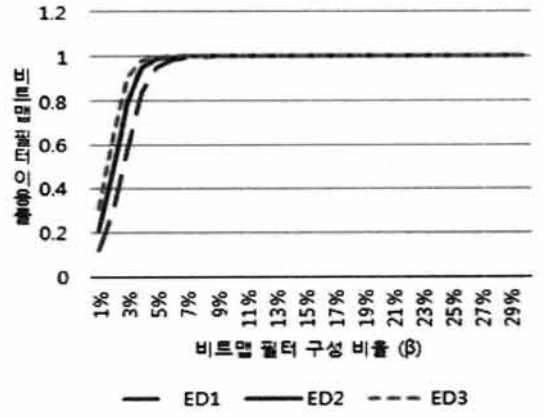
6.3 기존 방법과의 성능 비교

본 절에서는 [15]에서 제안되었던 방법과 본 논문에서 제안한 방법의 질의 처리 성능을 비교한다. [15]에서 제안된 방법은 MergeOpt[2]의 변형된 형태로 기존의 연구들에서 제안한 알고리즘 중 가장 좋은 질의 처리 성능을 보여주고 있다.

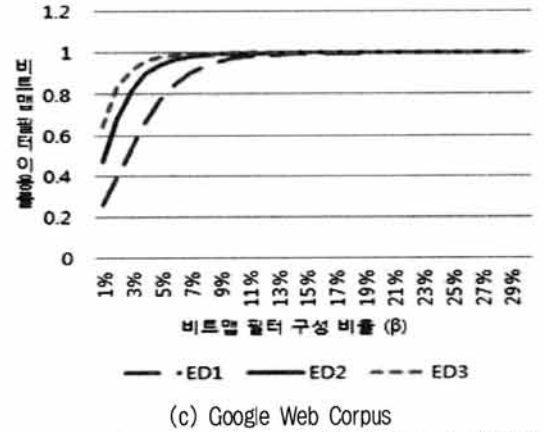
(그림 11)은 기존 방법과 제안한 방법의 질의 처리 시간을 나타낸 것이다. 이 실험의 비트맵 필터의 길이와 비트맵



(a) DBLP-Author



(b) DBLP-Title

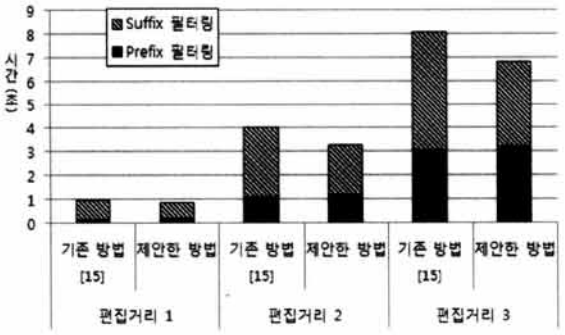


(c) Google Web Corpus

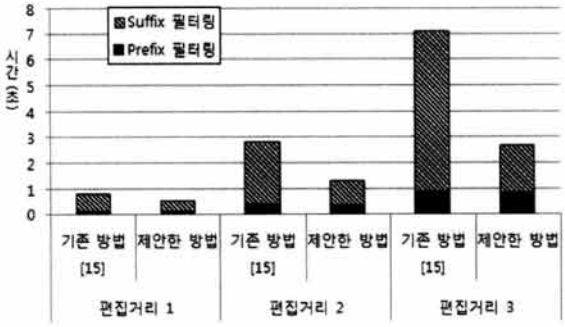
(그림 10) 비트맵 필터 구성 비율에 따른 비트맵 필터 이용률

필터 구성 비율 β는 6.1과 6.2의 결과에 따라 각각 64KB와 0.11로 하였다. (그림 11)에서 제안한 방법이 기존 방법보다 질의 처리 시간이 감소하였음을 알 수 있다. Prefix 필터링 시간은 제안한 방법과 기존의 방법이 비슷하며 제안한 방법이 기존 방법보다 Suffix 필터링 시간이 감소 한 것을 알 수 있다. 따라서 Suffix 필터링 시간이 감소하여 질의 처리 시간이 감소한 것을 알 수 있다.

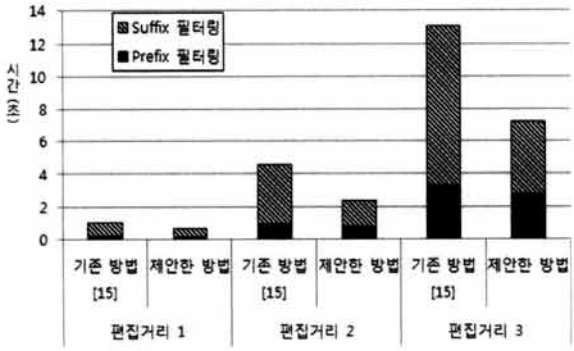
(그림 11)의 (a)의 경우 편집거리 1, 2, 3에 대하여 제안한 방법이 기존 방법보다 각각 11%, 18%, 15% 정도의 질의 처리 시간 개선 효과가 있음을 보이고 있다. (b)의 경우 각각 21%, 44%, 56% 정도의 질의 처리 시간 개선 효과가 있음을 알 수 있으며 (c)의 경우 각각 15%, 56%, 43%의 질의 처리 시간 개선 효과가 있음을 알 수 있다.



(a) DBLP-Author



(b) DBLP-Title



(c) Google Web Corpus

(그림 11) 질의 처리 시간 (β = 0.11, 64KB 비트맵 필터)

7. 결 론

본 논문에서는 효율적인 유사 문자열 검색을 위한 비트맵 필터를 이용한 역 리스트 탐색 기법을 제안하였다. 제안한 기법은 Suffix로 분류된 역 리스트들을 탐색하기 이전에 비트맵 필터를 확인하여 불필요한 역 리스트 탐색을 회피함으로써 리스트 병합 비용을 절감하였다.

실험을 통하여 비트맵 필터 구성 비율을 0.05로 하였을 경우와 그 이상으로 하였을 경우 질의 처리 성능의 차이가

없음을 확인하였으며 비트맵 필터의 길이가 길어짐에 따라 질의 처리 시간이 감소함을 확인하였다. 그리고 제안된 Suffix 필터링 기법이 기존 방법의 질의 처리 시간 개선 효과가 있음을 실험을 통하여 확인하였다.

향후에는 비트맵 필터의 공간 비용을 더 줄이기 위해 역 리스트의 길이나 역 리스트내의 문자열 ID의 분포에 따라 비트맵 필터의 길이를 다르게 하는 역 리스트 탐색 방법에 대해 연구할 예정이다.

참 고 문 헌

[1] 권인택, 김종익, "비트맵 필터를 이용한 효율적인 유사 문자열 검색 기법", 제 35회 한국정보처리학회 춘계학술대회 논문집, 제 18권 제 1호, pp.1298-1301, 2011.

[2] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," SIGMOD, pp743-755, 2004.

[3] C. Xiao, W. Wang, and X. Lin, "Ed-Join: an efficient algorithm for similarity joins with edit distance constraints," VLDB, 2008.

[4] S. Chaudhuri, V. Ganti, and R. Kaushik, "A Primitive Opeartor for Similarity Joins in Data Cleaning," ICDE, pp.5-5, 2006.

[5] C. Xiao, W. Wang, X. Lin, and Jeffrey Xu Yu, "Efficient Similarity Joins for Near Duplicate Detection", WWW, 2008.

[6] Roberto J. Bayardo, Y. Ma, and R. Crikant, "Scaling Up All Pairs Simialrity Search", WWW, 2007.

[7] Leonardo Andrade Ribeiro, and Theo Harder, "Generalizing prefix filtering to improve set similarity joins", Information Systems, 2010.

[8] C. Li, J. Lu, and Y. Lu, "Efficient Merging and Filtering Algorithms for Approximate String Searches," ICDE, pp.257-266, 2008.

[9] A. Behm, S. Ji, C. Li, and J. Lu, "Space-Constrained Gram-Based Indexing for Efficient Approximate String Search," ICDE, pp.604-615, 2009.

[10] C. Li, B. Wang, and X. Yang, "VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams," VLDB, pp.303-314, 2007.

[11] X. Yang, B. Wang, and C. Li, "Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently," SIGMOD, 2008.

[12] A. Arasu, V. Ganti, and R. Kaushik, "Efficient Exact Set-Similarity Joins," VLDB, pp.918-929, 2006.

[13] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, "An Efficient Filter for Approximate Membership Checking," SIGMOD, 2008.

[14] S. Chaudhuri, K. Ganjam, V. Ganti, R. Kapoor, Vivek R. Narasayya, Theo Vassilakis, "Data cleaning in microsoft SQL server 2005," SIGMOD, pp.918-920, 2005.

[15] N. Okazaki and J. Tsujii, "Simple and Efficient Algorithm for Approximate Dictionary Matching," In proc. of the 23rd International Conference on Computational Linguistics, pp.851-859, 2010.

[16] J. Barbay and C. Kenyon, "Adaptive intersection and t-threshold problems," SODA, pp.390-399, 2002.

[17] N. Koudas, S. Sarawagi, and D. Srivastava, "Record linkage: Similarity measures and algorithms," SIGMOD, 2006.

권 인 택

e-mail : kit0123@jbnu.ac.kr

2010년 전북대학교 전자정보공학부(학사)

2010년~현 재 전북대학교 전자정보공학부 (석사과정)

관심분야 : 유사 문자열 검색, 유사 조인 등

김 종 익

e-mail : jongik@jbnu.ac.kr

1998년 한국과학기술원 전산학과(학사)

2000년 한국과학기술원 전산학과 (공학석사)

2004년 서울대학교 컴퓨터공학부 (공학박사)

2004년~2007년 한국전자통신연구원 선임연구원

2007년~현 재 전북대학교 컴퓨터공학부 조교수

관심분야 : 유사 질의 처리, 스트림 데이터베이스, XML 데이터베이스