부분 문자열 선택도 추정을 위한 서픽스트리 변환 기법

(A Suffix Tree Transform Technique for Substring Selectivity Estimation)

이 홍 래 † 심 규 석 †† 김 형 주 †††
(Hongrae Lee) (Kyuseok Shim) (Hyoungjoo Kim)

요 약 선택도 추측은 관계형 데이타베이스에서 질의 최적화의 한 중요한 요소이다. 숫자 데이타에 대한 조건식에 대하여 이 주제는 많은 연구가 되어 왔으나 부분문자열에 대한 조건식은 최근에 이르러서야 관심의 초점이 되고 있다. 우리는 이 논문에서 이 문제를 위한 새로운 서픽스 트리 변환 알고리즘을 제시한다. 제안하는 기법은 서픽스 트리의 노드들을 단순히 잘라 없애 버리기 보다는 기본적으로 비슷한 카운트를 갖는 노드들을 구조적 정보를 유지하면서 병합하여 전체 크기를 줄인다. 본 논문은 여러 제약 사항 하에서 서픽스 트리를 그 크기를 줄이도록 변환을 하는 알고리즘을 제시하고 실생활 데이타를 대상으로 실험을 수행하여 우리가 제안하는 알고리즘이 기존의 알고리즘들보다 우수한 평균 상대 에러와 에러 분포 특성을 지니고 있음을 보인다.

키워드 : 질의 최적화, 부분문자열, 선택도 추측, 서픽스 트리

Abstract Selectivity estimation has been a crucial component in query optimization in relational databases. While extensive researches have been done on this topic for the predicates of numerical data, only little work has been done for substring predicates. We propose novel suffix tree transform algorithms for this problem. Unlike previous approaches where a full suffix tree is pruned and then an estimation algorithm is employed, we transform a suffix tree into a suffix graph systematically. In our approach, nodes with similar counts are merged while structural information in the original suffix tree is preserved in a controlled manner. We present both an error-bound algorithm and a space-bound algorithm. Experimental results with real life data sets show that our algorithms have lower average relative error than that of the previous works as well as good error distribution characteristics.

Key words : query optimization, substring, selectivity estimation, suffix tree

1. 서 론

인터넷이 발달하면서 텍스트 데이타는 현재 관계형 데이타베이스에서 매우 큰 비중을 차지하게 되었다. 이러한 문자값을 갖는 데이타에서 가장 많이 사용되는 조건식의 하나가 질의 문자열이 특정 칼럼 내에 존재하는 지를 테스트 하는 SQL의 like 구문이다. 예를 들어 "select lab from noticeboard where content like '%stream%'"과 같은 SQL 구문은 content 칼럼이 'stream'이라는 문자열을 포함하는 모든 연구실들을 출력한다. 이런 질의를 처리하기 위하여 질의를 최적화 하는 데 있어서 주어진 조건을 만족하는 튜플들의 선택도를 정확하게 추측하는 문제는 매우 중요하다. 선택도 추측의 문제는 숫자의 조건식에 대해서는 많은 연구가 진행되어 왔고 상용 데이타베이스에 적용되었으나 위의 예와 같은 부분문자열이 조건으로 들어가 있는 질의에 대하여는 최근에 들어서 연구가 시작되고 있다. 이들 중 많은 연구[1-4]는 카운트가 추가된 서픽스 트리[5]에 기반하고 있다. 모든 정보를 담고 있는 전체 서픽스 트리는 선택도를 정확하게 예측할 수 있는 반면에 그 크기는 원 데이타의 크기보다 커질 수 있어 문제가 된다. 이 공간을 줄이기 위하여 단순히 서픽스 트리의 노드들을 잘라내는 방법이 가장 많

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음
† 학생회원 : 서울대학교 컴퓨터공학과
hrlee@oopsla.snu.ac.kr
†† 정 회 원 : 서울대학교 전기 컴퓨터 공학부 교수
shim@ee.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부
hjk@snu.ac.kr
논문접수 : 2005년 11월 30일
심사완료 : 2006년 12월 27일

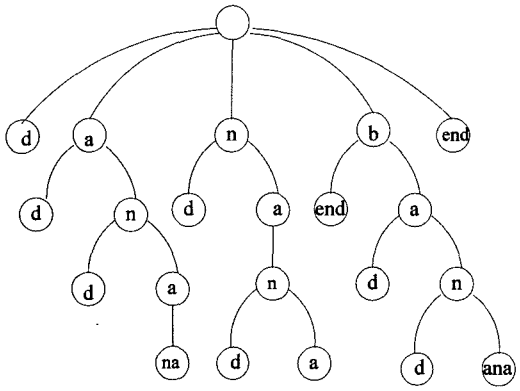


그림 1 예제 서픽스 트리

이 쓰이는데 이렇게 서픽스 트리의 사이즈를 줄인 후, 이 부분적 정보에 추측 알고리즘을 적용하는 방식으로 방식으로 부분문자열의 선택도를 추측한다.

이러한 알고리즘들의 정확도는 일반적으로 만족스럽지 못하다. 우리는 이 선택도 추측 문제에 대하여 다른 접근 방식을 취한다. 제안하는 방법은 기본적으로 서픽스 트리 안의 많은 노드들이 매우 비슷하다는 관찰에서 시작한다. 그림 1에서 볼 수 있는 바와 같이 서픽스 트리 내의 노드들은 많이 중복되어 있고 이러한 사실을 서픽스 트리의 크기를 줄이는 데 이용한다. 즉, 원래의 서픽스 트리를 간결하게 표현한, 압축된 서픽스 그래프로 변환하는 것이다. 우리는 노드들을 단순히 없애 버리지 않고 비슷한 노드들끼리 병합한다. 기본적으로 어떤 에러 한도 ε내의 비슷한 카운트를 갖는 노드들을 병합한다.

본 논문의 기여사항을 요약하면 다음과 같다.

- 부분문자열 선택도 추측을 위하여 새로운 서픽스 트리 변환 기법을 제안한다. 단순히 노드들을 없애는 이전 기법들과는 달리 노드들을 병합하여 노드 수를 줄이고 결과적으로 원래의 서픽스 트리는 서픽스 그래프로 변환된다.
- 실제 데이타 내에 존재하는 질의들(positive query)에 대하여 제한된 에러와 공간 한도 내에서 변환을 하는 알고리즘을 제시한다. 제한하는 알고리즘은 실 데이타 상에서 매우 정확하게 선택도를 추측한다.
- 실생활 데이타와 생성된 데이타 집합(synthetic data set)을 대상으로 실험을 하여 제안한 알고리즘이 크게 선택도 추측의 정확도를 향상시킴을 보인다.

논문의 이후 장들은 다음과 같이 구성되었다. 2장에서는 가지치기된-서픽스 트리 기반의 알고리즘을 제시하고 제안 알고리즘을 직관적으로 알 수 있는 예제를 본 후 관련된 몇 가지 기법들을 설명한다. 각 변환 기법들은 3장에서 자세하게 기술되어 있고, 변환알고리즘들은 4장에서 설명되어 있다. 기존의 방법들과의 융합도 이 장에 설명되어 있다. 5장은 실험 결과를 포함하고 있는데 기존의 알고리즘들과의 성능 비교를 하고 있다. 6장에서는 관련 연구들을 살펴보고, 7장에는 전체 내용이 정리되어 있다.

2. 예비 지식

2.1 서픽스 트리를 이용한 선택도 추측

2.1.1 서픽스 트리

서픽스 트리[5]는 문자열을 색인하기 위한 흔히 사용되는 트라이(trie)와 비슷한 자료 구조이고 주어진 문자열의 모든 접미사(서픽스)를 트리에 삽입하여 얻을 수 있다. 노드의 *레이블*은 각 노드가 가지고 있는 문자열이고, 노드에 *해당*하는 문자열은 루트에서 해당 노드까지의 경로 상에 존재하는 모든 노드의 레이블을 차례대로 붙인 문자열이다. 그림 1은 {"banana", "bad", "nand", "bed", "bend"}의 데이타에 해당하는 서픽스 트리이다. *카운트 서픽스 트리*(count suffix tree)는 노드들에 해당하는 문자열의 빈도가 부가적으로 붙은 서픽스 트리를 말한다. c(α)는 해당하는 문자열이 α인 노드의 카운트를 나타낸다. 이후에 우리는 서픽스 트리를 카운트 서픽스 트리와 같은 의미로 사용한다. 이러한 서픽스 트리가 있으면 어떤 부분문자열의 선택도를 정확하게 얻을 수 있다. 하지만 여기에 문제점은 전체 서픽스 트리는 사이즈가 커서 데이타베이스에 전체를 저장하기 힘들다는 점이다. 따라서 자연스러운 하나의 해결 방법은 단순히 서픽스 트리를 잘라(노드들을 삭제하여) 사이즈가 작게 만든 이후에 이 부분적인 정보를 가지고 있는 서픽스 트리에 추측 알고리즘을 이용하여 서픽스 트리의 원래 정보를 추측하는 것이다. 다음 절에서는 이러한 접근 방식에서 사용하는 추측 알고리즘들을 기술한다.

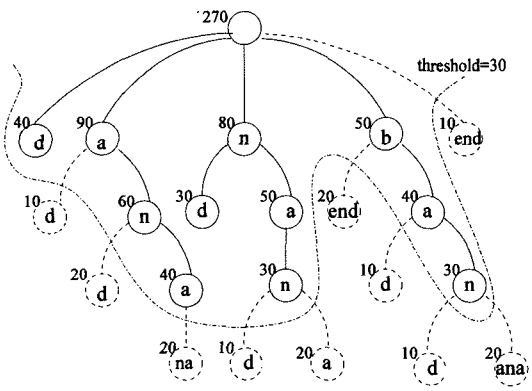


그림 2 가지치기 된 서픽스 트리의 예

2.1.2 KVI, MO 알고리즘

서픽스 트리를 작게 만드는 가장 간단한 알고리즘은 트리의 노드들을 잘라 내는 것이다. 이 방법은 어떤 규

칙에 의하여 노드들을 잘라 내는데, 이 결과로 생기는 서픽스 트리를 *가지치기된 서픽스 트리 PST*(pruned suffix tree)로 부른다. (이와 대비하여 원래의 서픽스 트리를 전체 *서픽스 트리 FST*(full suffix tree)라 한다.) 보통 줄이는 규칙으로는 특정 한계 값 이하의 카운트를 갖는 모든 노드들을 없애는 방법을 사용한다. 그림 2는 한 PST의 예이다. 각 노드 원 위의 숫자는 노드의 카운트를 나타내고, 점선의 원으로 표시된 노드들을 자른 기준 카운트가 30일 때 잘려지는 노드를 표시한다. N을 모든 튜플의 수, *Sel*을 실제 선택도, *Est*를 추측한 선택도라고 정의할 때, 어떤 부분문자열 조건식 σ가 PST 안에서 발견되었으면 $Est(\sigma)=Sel(\sigma)=c(\sigma)/N$이다. 부분문자열 질의어가 $\sigma=\sigma_1\cdots\sigma_l$라고 가정하자. 그러면 $Est(\sigma)$은 다음과 같이 나타낼 수 있다[3].

$$Est(\sigma)=Est(\sigma_1)\times\cdots\times Est(\sigma_l\mid\sigma_1\ldots\sigma_{l-1})=$$
$$Est(\sigma_1)\times\prod_{i=2}^{l}Est(\sigma_i\mid\sigma_1\ldots\sigma_{i-1})$$

예를 들면, 원래의 서픽스 트리에서 다음과 같이 추측을 한다.

$$Est(bad)=Est(b)\times Est(a\mid b)\times Est(d\mid ba)=$$
$$=c(b)/N\times c(ba)/c(b)\times c(bad)/c(ba)$$
$$=50/N\times 40/50\times 10/40=10/N\approx 3.7\%$$

PST는 가지치기된 서픽스 트리이므로 $Sel(\alpha_l|\alpha_1\ldots\alpha_{l-1})$을 이처럼 정확하게 구할 수가 없다는 것이 문제이다. 그림 2의 PST에서 "bad"에 해당하는 노드가 잘려 나가 $Sel(bad|ba)$를 정확하게 구할 수 없다.

Algorithm KVI

Krishnan 등은 [1]에서 해당하는 문자열을 찾지 못하는 상황을 처리하기 위한 여러 가지의 방법들을 제안하였다. 찾을 수 있는 가장 긴 문자열을 찾은 다음, 나머지 문자열을 다시 반복적으로 찾고 이 부분 문자열들이 서로 독립적이라고 가정한다. 따라서 전체 선택도는 각 부분 문자열의 선택도들을 곱하여 얻는다.

예제 2.1: KVI 추측

그림 2의 PST에서 문자열 σ="band"는"ban"과 "d"로 분해 되므로 KVI 알고리즘은 다음과 같이 선택도를 추측한다.

$$Est(band)=Est(ban)\times Est(d\mid ban)$$
$$\approx Est(ban)\times Est(d)$$
$$=c(ban)/N\times c(d)/N\approx 1.6\%$$

MO 알고리즘

최신의 알고리즘인 MO는 Jagadish 등에[3] 의해 제안되었다. MO는 조건식의 문자열을 겹치는 부분이 많도록 분해하고 겹치는 부분의 조건부 확률을 이용한다. 만약 $\sigma_1=\alpha\beta$, $\sigma_2=\beta\gamma$ 이고 β가 둘의 가장 길게 겹치는 보조 문자열이라고 하면 MO는 $\sigma_1\sigma_2$의 선택도를 $Est(\sigma_1\sigma_2)=Est(\alpha\beta)*Est(\gamma|\beta)$으로 예측한다. 이는 일종의 Markov 추측알고리즘이라고 볼 수 있다.

예제 2.2: MO 추측

그림2의 PST에서, MO는 τ="band"을 "ban"과 "nd"으로 분리하여 다음과 같이 선택도를 추측한다.

$$Est(band)=Est(ban)\times Est(d\mid ban)$$
$$\approx Est(ban)\times Est(d\mid n)$$
$$=c(ban)/N\times c(nd)/c(n)\approx 4.2\%$$

2.2 제안하는 노드 병합 기술

2.2.1 예제

앞서 기술한 두 알고리즘은 트리의 노드를 잘라 낸 후 질의 처리 시에 없어진 정보를 추측한다는 공통점이 있다. 실 데이타로 많은 실험을 수행한 후 우리는 그러한 접근 방법이 많은 경우에 성능이 나쁘게 나온다는 것을 발견하였다. 우리는 이 문제를 위하여 기존의 방법과는 다른 방법으로 접근을 하였다. 가장 기본이 되는 관찰은 서픽스 트리 안의 많은 노드들이 중복된 정보를 가지고 있다는 것이다. 이는 최근에 제안된 SPINE 색인[6]의 관찰과 비슷하다. 그림1의 서픽스 트리를 보면 많은 노드들이 같은 레이블을 가지고 있는 것을 볼 수 있다. 따라서 제안하는 알고리즘은 단순히 노드들을 잘라 없애 버리지 않고 기본적으로 노드들을 병합하여 중복을 없애 서픽스 트리를 작게 만든다. 다음의 예는 직관적으로 이를 보여준다.

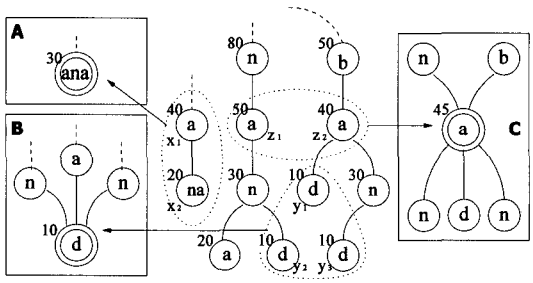


그림 3 노드 병합의 예

그림 3은 그림 2의 서픽스 트리의 일부분이다. 위에서 기술한 바와 같이 많은 노드들이 중복되어 있고 노드들을 병합하여 이 트리를 작게 만들 수 있다. 기본적인 노드 병합의 방법은 레이블이 같고 카운트가 비슷한 노드들을 합하는 것이다. 병합한 노드의 카운트는 노드 카운트의 평균을 사용한다. 노드들을 병합하는 데는 여러 가지 방법이 가능한데 이 중 세 가지 흥미로운 경우에 초점을 맞추어 보자. 각 경우들은 각각 A, B, C로 표시되어 있고 병합이 되는 노드들은 x,y,z로 첨자가 붙어 있

다. 이중선의 원으로 표시된 노드들은 병합의 결과로 생긴 노드를 의미한다.

2.2.2 서픽스 트리 변환에 의한 선택도 추측

앞 절의 예제에서처럼 본 논문은 서픽스 트리의 정보를 축약하여 그 크기를 줄여서 선택도 추측에 이용한다. 노드 병합은 서픽스 트리의 정보를 축약하는 가장 중요한 수단으로 이 과정에서 원래의 서픽스 트리는 서픽스 그래프로 변환된다. 이 변환을 병합되는 노드들의 상대적 위치를 기준으로 하여 예제 A의 경우처럼 단일 경로에 연달아 존재하는 노드들을 합하여 하나의 노드로 만드는 **V-변환**(Vertical)과 예제 B나 C의 경우처럼 모든 노드가 단일 경로에 있지 않은 경우의 **H-변환**(Horizontal)으로 나누어 볼 수 있다. 기본이 되는 위의 두 변환에서 카운트에 관한 정보는 에러가 생기지만 노드 레이블은 보존된다. 이를 확장하여 여러 가지 노드 레이블을 간략하게 표현할 수 있다면 더 효율적으로 트리의 크기를 줄일 수 있을 것이다. 다음 절에 설명하는 블룸 필터는 집합을 효율적으로 나타낼 수 있는 자료 구조로 여러 레이블을 효과적으로 축약하여 변환의 효율을 높이기 위하여 사용하였다.

2.2.3 블룸 필터(Bloom Filter)

블룸 필터는 n개의 원소를 가진 집합 $S=\{s_1,s_2,...,s_n\}$를 소속 질의(임의의 원소가 해당 집합에 속하는지의 여부를 묻는 질의)에 답할 수 있도록 크기가 m=O(n)인 비트 벡터 V를 이용하여 나타내는 방법이다. 블룸 필터는 Burton Bloom[7]에 의하여 제안되었으며 프록시 서버 내의 캐쉬 정보를 요약하여 표현하거나[8], 최장 접두사 매칭에 사용하거나[9], XML 경로 질의의 선택도를 추측하는[10] 등의 다양한 분야에서 널리 사용되고 있다.

기본 아이디어는 각 원소 $s \in S$를 k개의 독립적인 해시 함수를 사용하여 V 내의 k개의 위치에 대응시키는 것이다. 처음에 V의 모든 비트들은 0으로 시작한다. s에 대하여 $h_1(s)$, $h_2(s)$, ..., $h_k(s)$에 해당하는 위치의 비트들이 1로 설정된다. 어떤 원소 t가 S에 있는 지를 테스트하기 하려면 모든 $h_i(t), 1 \le i \le k$의 위치의 비트들이 1로 설정되어 있는지 보면 된다. 만약 하나라도 1이 아닌 비트가 있으면 t는 S에 있지 않다. 모두 1일 경우 t가 S 안에 있다고 가정하지만 이것이 위양성(false positive)일 가능성도 있고 이 확률은 다음과 같이 나타낼 수 있다[7].

$$p_f = (1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{nk}{m}})^k$$

블룸 필터를 이용한 변환

제안하는 서픽스 트리 변환에서 블룸 필터는 여러 레이블을 축약하여 나타내는 데 사용한다. 즉 문자열이 원소가 되어 블룸필터는 여러 문자열을 비트 벡터를 이용하여 축약하여 표현하는 것이다. 병합된 노드의 레이블은 블룸필터의 비트 벡터가 된다. 본 논문에서는 서브트리 전체를 하나의 노드로 표현하는 데에 사용하고 이를 **B-변환**이라 한다.

3. 서픽스 트리 변환 기법

3.1 H-변환

정의 3.1: 노드에 레이블이 있고 루트 노드가 있는 방향성 그래프 G=(V, E, r) (r은 루트 노드, $V \subseteq V_I \cup V_L$, V_I는 내부 노드(*internal node*)의 집합, V_L은 단말 노드(*leaf node*)의 집합이고 $E = V \times V$는 간선의 집합)에 대하여 *H-변환*은 모두 같은 레이블을 갖는 임의의 노드들의 집합 $U=\{u_i \in V \mid 1 < i < k\}$를 병합하여 하나의 노드 u로 변환한다. r은 변환되지 않고 변환에서 레이블은 보존된다. 즉 $label(u) = label(u_1) = ... = label(u_k)$이다. 간선 $e \in E$도 노드가 변환됨에 따라 관여된 노드만 변할 뿐 보존된다.

H-변환은 U의 모든 노드들이 단말 노드인 경우와 그렇지 않은 경우 두 가지가 다른 특성을 지님을 관찰 할 수 있다. 모든 노드들이 단말 노드인 경우는 변환을 하더라도 경로 정보가 그대로 보존되나 내부 노드가 관여된 경우는 거짓 경로 정보가 생겨난다. 그림 4에서 (a)는 변환 전의 그래프이다. (b)는 레이블이 d인 두 단말 노드가 H-변환된 모습이고 루트에서의 두 경로 *abcd, xbyd*는 변환 전 후에 보존됨을 볼 수 있다. (c)는 레이블이 b인 두 내부 노드가 H-변환된 모습인데 변환 전의 그래프에 있던 *abcd, xbyd* 외에 *abyd, xbcd* 등 원래의 그래프에는 존재 하지 않았던 경로를 볼 수 있다.

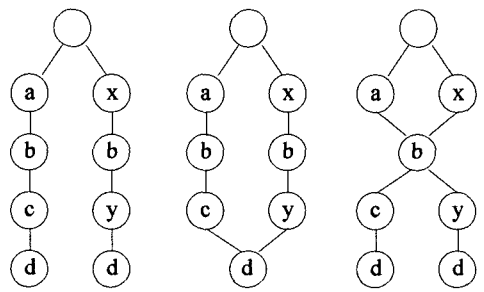


그림 4 H 변환 예

3.1.1 소수 번호 레이블링에 의한 경로 정보의 보전

앞 절의 예에서 볼 수 있는 바와 같이 H-변환을 사용하면 중복된 정보를 가지고 있는 노드들이 병합되어 서픽스 트리의 크기를 크게 줄일 수 있는 반면에 원래

의 트리에는 존재하지 않았던 경로들이 생겨나서 부분 문자열 선택도 추측의 정확도를 낮추는 결과를 가져 온다. 본 논문에서는 이렇게 노드들을 변환하면서 사라지는 경로 정보를 축약하여 보전하는 방법을 제안한다. 이에 앞서 변환에 관여되는 노드가 단말 노드인지 아닌지에 따라 다른 특성을 나타내므로 H-변환을 이를 기준으로 변환되는 노드들이 모두 단말 노드인 경우의 *HL-변환*과 그렇지 않은 경우의 *HI-변환*으로 세분할 수 있다.

HL-변환은 경로 정보가 문제 되지 않기 때문에 HI-변환에 초점을 맞추어서 생각한다. 제안하는 기법은 소수 번호 레이블링을 기반으로 한다. 즉 각 노드들에 노드 번호 NID을 부여하되 소수로 하는 것이다. H-변환 시 경로 정보 보전은 정수론적 성질을 이용하여 변환되는 노드들의 경로를 축약하여 저장함으로써 가능하다. HI-변환 시 새로이 생성되는 노드를 *HI-노드*라고 한자. 이 노드는 약간의 부가적 정보를 가져 변환되기 이전의 경로 정보를 축약하여 저장하는 것이다. 이 부가적인 정보는 SID(start, end)의 두 숫자의 쌍으로 이 후 경로 탐색을 하다 HI-노드에 도착하면 그 노드의 SID를 보고 어떤 경로가 원래의 서픽스 트리에 있었던 지 판단하는 것이다. 추가적으로 HI-노드는 자식들의 참조, 즉 나가는 간선들의 순서 정보를 가지고 있다고 가정한다.

HI-노드

노드들이 변환될 때 HI-노드는 병합되는 노드들의 자식 노드들을 연속된 범위에 저장한다. 따라서 만약 k개의 노드가 변환되어 병합될 때 그 노드들의 자식 노드들의 개수가 총 n개라면 그 n개의 간선들은 k개의 범위들로 분할 할 수 있다. 간 범위 내에서 자식 노드(간선)들은 문자숫자(alphanumeric) 순서로 저장된다. 따라서 만약 들어오는 경로를 보고 해당 경로에 대하여 원래의 서픽스 트리에 존재하였던 범위의 경계를 알 수 있다면 우리는 정확하게 어떤 자식 노드들을 검색하여야 알 수 있다.

예를 들어 그림 5에서 레이블이 a인 세 노드가 변환되어 HI-노드가 생성되고 연관된 경로들은 $u_1u_2v_1w_1w_2w_3$

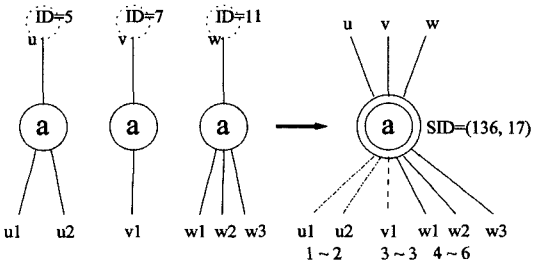


그림 5 HI-노드의 생성

처럼 순서대로 저장된다. 만약 탐색을 하다가 u노드를 거쳐 HI-노드에 도착하였다고 하면 이후의 6개의 경로 중 어떤 경로가 원래의 서픽스 트리에 있는 *유효한* 경로인 지 알 수 없다. 이 경우 실제 유효한 경로는 첫 번째에서 두 번째 경로로 u1과 u2 뿐이다. 들어오는 경로에 따른 유효한 범위를 판단하기 위하여 다음의 정리를 사용한다.

정리 3.2: 중국인의 나머지 정리[10]

$m_1, m_2, ..., m_r$이 상대적으로 서로 소인 양의 정수라고 하자. 그러면 합동식의 시스템

$$x \equiv a_1 (\mathrm{mod}\, m_1)$$
$$\cdots$$
$$x \equiv a_r (\mathrm{mod}\, m_r)$$

은 모드 $M=m_1m_2...m_r$로 유일한 해를 갖는다. (≡는 합동식 기호)

예제 3.3: x ≡ 1 (mod 3), x ≡ 2 (mod 4), x ≡ 3 (mod 5)은 모듈로 60(=3·4·5)으로 x=58을 유일한 해로 갖는다. 58 ≡ 1 (mod 3), 58 ≡ 2 (mod 4), 58 ≡ 3 (mod 5)임을 확인할 수 있다.

정리 3.2을 사용하여 SID의 start는 각 범위의 시작 색인을 인코딩하고 end는 끝 색인을 인코딩한다. 따라서 두 개의 합동식 시스템이 생기는데 시작 색인의 경우에 변환되는 노드들의 부모의 NID가 정리 3.2의 식에서 m_i, 각 범위의 시작 색인들이 a_i가 되고 이 합동식의 유일한 해를 SID의 start로 저장한다. 노드를 검색할 때 HI-노드에 도달하면 바로 직전 노드의 NID로 HI-노드의 start를 나눈 나머지가 유효한 경로의 시작 색인이 된다. 끝 색인의 경우도 마찬가지이다. 노드들은 문자숫자 순서로 저장하고 유효한 시작과 끝 색인을 구하면 이 범위 내에서는 이진검색을 하면 경로 탐색을 계속할 수 있다.

예제 3.4: 그림 5에서 변환되는 세 노드의 부모 노드 u, v, w는 소수 레이블이 각 5, 7, 11이다. 만약 자식 노드를 $u_1u_2v_1w_1w_2w_3$순서로 저장한다면 u,v,w의 시작 색인들은 1, 3, 4이고 끝 색인들은 2, 3, 6이다. 예를 들면 u에서 오는 경로는 순서대로 저장된 자식 노드 중 첫 번째에서 두 번째까지가 유효한 경로인 것이다. 이 경우 SID의 start는 합동식 x ≡ 1 (mod 5), x ≡ 3 (mod 7), x ≡ 4 (mod 11)의 해 x이고 136이다. 마찬가지로 SID의 end는 17이 된다. 만약 노드 u를 통하여 HI-노드 z에 도달하였다면 여섯 개의 경로 중 유효한 경로를 식별하기 위하여 start와 end값을 u의 NID인 소수 5로 나누면 된다. 136 ≡ 1 (mod 5)이고 17 ≡ 2 (mod 5)이므로 유효한 경로는 범위는 1~2임을 알 수 있다.

3.2 H(k)-변환

위의 SID를 이용한 H-변환은 변환된 노드에 이르면

경로에서 바로 이전 노드의 정보(NID)를 이용하여 유효한 다음 경로를 판단한다. 하지만 만약 연달아 두 개의 노드가 HI-변환된다면 두 번째 HI-노드에서는 이전 노드가 같을 수 있기 때문에 다음 유효한 경로를 결정하지 못하는 경우도 발생할 수 있다. 이처럼 H-변환은 전체 경로 정보를 완전히 보전하는 변환이 아니라 제한된 부분 경로 정보를 보전하는 *손실 변환*이라 할 수 있다. 이 손실의 정도를 다음과 같이 일반화 하여 H-변환을 다시 정의할 수 있다.

정의 3.5: H(k)-변환

변환 H가 그래프 G=(V, E, r)를 G′=(U, A, r)로 변환하였다고 하자. G′에 존재하는 k+1개의 노드로 이루어진 길이가 k인 임의의 경로 가 원래 G에 존재하는 경로인지를 판단할 수 있다면 이 변환을 *H(k)-변환*이라 한다.

H(k)-변환은 길이가 k인 경로를 보전하고 위의 소수 레이블링에 의한 H-변환은 H(2)-변환이다. 그러나 위의 방법을 확장하여 간단히 임의의 k에 대한 H(k)-변환을 구현할 수 있다. 방법은 경로에서 HI-노드 바로 직전에 있는 노드의 가 아니라 직전 k-1개의 노드의 *NID*를 이용하여 *HI*-노드의 *SID*를 만드는 것이다. 경로를 탐색하다 HI-노드에서 다음 유효한 경로를 찾을 때는 바로 직전 노드 하나가 아니라 이전 k-1개의 노드의 NID를 곱하여 SID의 start와 end를 나누어 유효한 범위를 결정하여 찾으면 된다.

3.3 V-변환

정의 3.6: V-변환

그래프 G=(V, E, r)에 대한 *V-변환*은 $1 \le i \le k-1$을 만족하는 각 u_i가 하나의 자식 노드 u_{i+1}를 가지고 있는 노드들의 집합 $U=\{u_1,...u_k\} \subseteq V$에 대하여 노드 $u_i \subseteq U$를 하나의 노드 u로 병합하는 변환이다. $label(u)=label(u_1)\cdot ... \cdot label(u_k)$이 되고 $1 \le j \le k-1$인 간선 $e=(u_j, u_{j+1})$는 없어지고 $e=(u_i,v) \in E$는 $e'=(u,v)$로 변환된다. 결과로 생긴 노드 u는 *V-노드*라 한다.

V-변환은 서픽스 트리가 트라이(trie)를 축약한 것으로 생각할 때와 비슷한 방법으로 카운트가 비슷한 노드들을 축약하는 변환으로 볼 수 있다. 예를 들어 그림 4의 (a)에서 왼쪽 경로 a/b/c/d의 노드들의 카운트가 에러 범위 안에 있다면 4개 노드는 V-변환되어 레이블이 abcd인 하나의 노드가 될 것이다.

3.4 B-변환

앞 절에서 간략하게 설명했던 바와 같이 블룸 필터를 이용하면 서로 레이블이 다른 노드들을 병합할 수 있다. H-변환의 경우도 블룸 필터를 이용하면 레이블이 같지 않은 노드들에 대해서도 변환할 수 있도록 적용할 수 있지만 서로 다른 내부 노드들을이 변환되면 위 쪽 경로의 거짓해답 확률이 아래로 전파되므로 H-변환에서는 블룸필터를 사용하지 않고 서브 트리 단위로 적용할 있는 변환을 정의한다.

정의 3.7: B-변환

그래프 G=(V, E, r)에서 $U=\{u_1,...u_k\}$가 서브 트리 $T \subseteq G$의 노드들이고 그 루트 노드가 $u_r \in U$이라고 하자. *B-변환*은 U의 노드들을 병합하여 하나의 노드 u_r로 변환한다. u_r를 제외한 모든 노드들과 T의 모든 간선들은 제거된다. u_r은 서브 트리의 루트에서 만들 수 있는 모든 문자열로 형성한 블룸필터의 비트열을 레이블로 갖는다. 변환된 u_r을 *B-노드*라 한다.

4. 변환 알고리즘

4.1 변환 이익 분석

4.1.1 변환 사이의 관계

서픽스 트리를 변환하는 알고리즘을 기술하기 전에 각 변환의 특성을 좀 더 자세히 살펴볼 필요가 있다. 먼저 변환의 적용 가능성을 생각해 보면 V-변환은 적용할 수 있는 노드의 형상이 제한적이고 H-변환과 B-변환은 잠재적으로 모든 노드에 적용 가능하다. 노드의 재변환 가능 여부를 생각하여 보면 V-변환된 노드는 카운트에 에러가 도입되었다는 점만 제외하고는 변환 전의 일반적인 노드와 동일하므로 V-변환된 노드는 다시 모든 변환의 대상이 될 수 있다. H-변환된 노드는 원래 V-변환이 가능하였다고 하더라도 이미 경로가 병합되어 단일 자식 노드를 가지지 않으므로 더 이상 V-변환 가능하지 않다. B-변환도 형상을 변화하고 레이블을 바꾸므로 B-변환된 노드는 이후 추가적인 V-나 H-변환이 불가능하다. 하지만 블룸 필터에 삽입하는 문자열만 다시 처리하면 보다 원래의 서브 트리를 포함하는 서브 트리에 포함하여 변환할 수 있으므로 다시 B-변환하는 것은 가능하다.

4.1.2 변환에 따른 공간 이익 분석

하나의 노드에 여러 가지 변환을 적용가능하다면 임의의 노드에 어떤 변환을 어떻게 적용할 것인가 하는 문제가 생기게 된다. 변환의 목적이 서픽스 트리의 사이즈를 줄이는 것이므로 변환으로 절약되는 공간의 크기를 판단의 기준으로 삼았다. N은 전체 서픽스 트리의 노드의 개수이고 M_n, M_e, M_b은 각각 노드, 간선, 블룸 필터의 크기라고 하자. k개의 노드가 변환되어 하나로 되는 경우에 V-, H-, B-변환에 의해 절약되는 공간의 크기 S_V, S_{HI}, S_{HL}, S_B를 생각해 보자. H-변환은 모두 단말여부인가의 여부에 따라 그 계산이 달라지므로 HI-변환과 HL-변환의 경우를 나누어 생각하였다. k개의 노드가 V-변환되는 경우 처음과 마지막 노드 사이의 간선이 모두 없어지고 노드 하나가 남으므로 식 (2)가

성립한다. B-변환의 경우는 사이즈 k의 서브트리 내의 루트를 제외한 모든 노드와 간선들이 사라지고 대신 블룸필터가 생기므로 식 (3)이 된다. HL-변환의 경우에 노드들은 하나가 남고 모두 없어지지만 간선은 하나도 없어지지 않으므로 공간 이득은 식 (4)이다. HI-변환의 경우도 절약되는 노드와 간선의 수는 HL-변환의 경우와 같지만 추가적으로 저장되는 SID의 비용을 계산하여야 한다. n번째 소수의 크기는 O(n log n)[10]이고 병합되는 노드들의 정리 3.2에서 k개의 NID를 이용하여 식이 구성되므로 SID의 크기는 2k log(Nlog N)이다. 따라서 S_{HI}는 식 (5)와 같다. 단순히 k개의 노드가 병합되는 단순한 경우에는 $S_V > S_B$과 $S_{HL} > S_{HI}$이 성립함을 알 수 있다. 여기에 $(k-1)M_e > M_b$이 성립한다면 S_B은 S_{HL}보다 커지고 전체 절약되는 크기의 순서는 $S_V > S_B > S_{HI} > S_{HL}$이 된다.

$$S_V = (k-1)(M_n + M_e) \quad (2)$$

$$S_B = (k-1)(M_n + M_e) - M_b \quad (3)$$

$$S_{HL} = (k-1)M_n \quad (4)$$

$$S_{HI} = (k-1)M_n - 2k\log(N\log N) \quad (5)$$

실제 서픽스 트리에서의 분석은 이보다 훨씬 복잡하여 여러 가지를 고려하여야 한다. 앞 절에서 기술한 바와 같이 한 번 변환된 노드가 다시 변환될 수도 있고 주어진 에러 범위 내에서 한 변환이라도 여러 가지로 적용할 수도 있다. 논문에서는 얻어지는 공간 이득을 다음과 같이 근사하여 변환 알고리즘에 이용하였다.

$$S'_V = kM_n + (k-1)M_e \quad (6)$$

$$S'_{HL} = kM_n, S'_{HI} = kM_n - 2k\log(N\log N) \quad (7)$$

V-변환에서 결과로 생긴 노드는 같은 레이블을 갖는 노드가 있다면 다시 H-변환될 수 있으므로 식 (6)으로 계산할 수 있고, H-변환의 경우도 같은 레이블을 갖는 노드들이 실제로 많으면 식 (7)로 근사할 수 있다.

4.2 변환 알고리즘

4.2.1 기본 그리디 알고리즘

알고리즘 1은 위의 공간 이익 계산을 기준으로 한 그리디 변환 알고리즘으로 서픽스 트리의 노드를 순회하며 각 노드에 대하여 가장 공간 이익이 큰 변환을 한다. B-변환이 서브트리에 대하여 적용되므로 노드들은 아래에서 위(Bottom-up)방식으로 순회하며 변환을 한다. 각 노드에서의 변환은 함수 transferNode에서 이루어지는데 해당 노드가 V-나 B-변환이 가능하다면 부모 노드에서 더 큰 변환이 가능한지를 체크하여 볼 필요가 있으므로 변환의 판단을 미루고 리턴한다. 이는 어느 조상 노드에서 V-나 B-변환이 불가능할 때까지 계속 전파되고 그 시점에서 판단이 미루어 졌던 각 자손 노드들에 대한 공간 이득을 계산하여 이득이 가장 큰 변환을 수행한다. 변환을 수행할 때, 입력받은 카운트 차이가 에러 범위 ε안에 드는 노드들을 변환한다. 변환의 결과로 생긴 노드들도 이후 변환에 사용될 수 있으므로 변환으로 생성된 노드들의 카운트의 범위(최소 값과 최대 값)을 저장하여야 한다. V-변환된 노드는 바로 H-변환 가능 여부를 체크하고 H-변환된 노드는 다시 변환될 수 있으므로 SID는 모든 트리의 노드가 순회된 다음에 계산하여 고정하여야 하고 중간 과정에는 병합된 노드들의 NID를 저장한다. 이 과정에서 한 노드는 한 번 이상 저장되지 않으므로 변환에 필요한 공간은 O(n)이다.

주어진 서픽스 트리의 루트 노드에 대하여 transferNode가 수행되고 나면 모든 변환 가능한 모든 노드가 변환되어 원래의 서픽스 트리는 서픽스 그래프로 변환된 상태이다. 그러나 아직 HI-노드들과 B-노드들은 최종 형태가 아니어서 각각 SID와 블룸필터가 계산되지 않은 상태이다. 변환 후에 있는 내부 노드들에 NID를 할당하고 이를 이용하여 HI-노드들의 SID를 계산하고 블룸필터를 생성하면 변환은 종료된다. 변환의 과정에서 각 노드들은 한번씩 trasnferNode 함수에 의해 방문되고 변환이 미뤄진 경우 실제 변환 시 다시 한 번 방문되어서 최대 두 번씩 방문되므로 변환에는 O(n)의 수행시간이 소요된다.

Algorithm 1 그리디 변환 알고리즘

```
algorithm transformST (τ : suffix tree, ε: error bound)
1: begin
2:   transferNode(the root node of τ , ε)
3:   for all internal nodes in τ
4:     assign NID
5:   for all created HI-Nodes
6:     calculate SID
7:   for all created B-Nodes
8:     make a bloom filter
9: end

transferNode (n: node, ε: error bound)
1: begin
2:   if n is a leaf node
3:     mark n as V- & B-transformable and return
4:   for all children c_i of n
5:     transferNode(c_i)
6:   if n has only one child and it is V-transformable
7:     if n is a single child
8:       mark n as V-transformable and return
9:     else
10:      transform descendants of n according to space gain S
11: else
12:   if subtree rooted at n is B-transformable
13:     mark n as B-transformable and return
14:   else
15:     transform descendants of children of n according to
        space gain S
16: end
```

4.2.2 에러 보장 알고리즘

그리디 알고리즘에서 초기의 에러 바운드는 변환에서 기준의 역할을 하지만 에러를 보장하지는 못한다.

H(k) 변환은 연속되어 k개의 노드가 H변환 되더라도 경로를 보전하지만 k+1개의 노드가 연속되어 변환되면 경로를 보장할 수 없다. 따라서 비록 노드의 카운트는 에러 한계에 들더라도 경로 자체가 유효한지 알 수 없으므로 에러를 보장할 수 없다. 에러 한도를 보장하는 변환 알고리즘은 그리디 알고리즘을 약간 변형하여 구현이 가능하다. 기본적으로 같은 알고리즘을 사용하되 변환에 아래와 같은 제약을 두어 에러 한도를 보장할 수 있다. 경로의 모호성이 발생하지 않도록 변환하면 되는데, 이는 H(k)-변환을 사용할 때 k+1개의 노드가 연속해서 변환하지 않도록 함으로써 가능하다. 즉 변환을 수행하다 자손 노드들이 자기에 연달아 k개가 HI-변환되었으면 변환을 하지 않는 것이다. 이렇게 하여 절대, 상대 에러에 상관없이 에러를 보장할 수 있지만 이는 변환을 제약하여 결과적으로 얻는 공간 이득을 감소시키는 단점이 있다. 실험 결과 그리디 알고리즘으로 원래 서픽스 그래프의 약 10% 수준으로 공간을 줄이는 경우에 같은 에러 한도를 보장하면 약 40% 수준으로 공간을 줄이는 효과가 감소하였다.

4.2.3 공간 보장 알고리즘

그리디 알고리즘은 주어진 공간 제약에 맞게 변환이 가능한 알고리즘으로 변형이 가능하다. 알고리즘 1의 9, 14 행은 기술한 바와 같이 공간이득을 계산하여 가장 큰 변환을 수행하는데 V-변환을 한 후에는 다시 H-변환 가능 여부를 체크하고 B-변환은 서브 트리의 모든 노드가 변환되지 않았거나 V-변환 되었을 경우에만 수행한다. 적용할 변환을 판단할 때 다음 두 가지 변화를 주어서 공간 보장이 가능하도록 한다.

V-변환 후 바로 가능한 H-변환을 수행하지 않고 전체 트리에서 순회가 끝난 후 일괄적으로 변환을 수행한다.

B-변환을 체크할 때 서브트리에 이미 변환이 수행된 노드가 있다고 하더라도 서브트리 내의 노드가 모두 에러 기준 안에 있고 그 개수가 이미 적용된 공간 이득 이상을 볼 수 있는 개수 이상이면 B-변환을 수행한다.

정리 4.1: 위의 방법으로 수정된 그리디 알고리즘에 생성하는 서픽스 그래프가 차지하는 공간은 주어진 에러 기준에 대하여 단조 감소한다.

증명: 에러 기준이 ε에서 $\varepsilon' > \varepsilon$으로 변경되었고 ε에서의 변환이 T, ε'에서의 변환이 T'이라고 하자. 각 노드에 대하여 변환 T에서 적용되는 변환은 T'에서 적용되는 꼭 같지는 않다. T와 T'에서의 변환을 분석하기 위하여 공간 이득 S을 보면 식 (2)~(7)에서 $S'_V > S'_B$, S'_H이므로 T에서 V-변환되었던 노드는 변환되었던 노드들의 경로가 ε이 증가하면서 더 길어지므로 T'에서도 V-변환된다. 비슷하게 T에서 B-변환되었던 노드들은 구조적으로 V-변환될 수 없었던 노드들이고 H-변환하는 공간 이득보다 B-변환하는 공간이득이 더 커서 B-변환된 것이므로 T'에서도 B-변환되게 된다. H-변환되었던 노드들은 상황에 따라 T'에서는 V-, H-, B- 변환이 모두 적용될 가능성이 있다. 그리고 변경된 알고리즘에 의하여 V-변환된 노드도 이후 B-변환될 수 있게 하므로 결과적으로 변환의 가짓수(T에서 적용되는 변환 → T'에서 적용되는 변환)는 (V→V, B), (H→H, V, B), (B→B)이다. 여기에서 적용되는 변환의 종류가 변하지 않는 경우는 에러 기준이 커져서 더 많은 노드가 하나의 노드로 변환되므로 증가된 에러 기준 ε'에서 차지하는 공간은 증가되지 않는다. 따라서 적용되는 변환이 바뀌는 경우에만 공간이 증가하지 않음을 보이면 된다. 변환이 바뀌는 경우 중 (H→V)로 바뀌는 경우는 식 (6)과 (7)에 의하여 공간이 감소하므로 T'에서 B-변환되는 노드들에 대하여서만 공간이 감소됨을 보이면 된다. 이 경우를 일반화 시켜서 어느 서브트리 내의 k개의 노드가 새로 B-변환의 대상으로 고려되는 경우를 생각해보자. 이 중 T에서 k_1개의 노드가 V-변환되어 l개의 V-노드가 생겼었고, k_2개의 노드가 HI-변환, k_3개의 노드가 HL-변환되었었고, k_4개의 노드가 B-변환되어 m개의 B-노드가 있었다고 가정하자. 이 경우 각 변환에 의한 공간 이득은 식 (2)~(7)에 의해 다음과 같다.

$$
\begin{aligned}
S_1 &= k_1 M_n + (k_1 - l) M_e \\
S_2 &= k_2 M_n \\
S_3 &= k_3 M_n - 2k_3 \log(N \log N) \\
S_4 &= (k_3 - m) M_n - m M_b + (k_3 - m) M_e \\
S_T &= S_1 + S_2 + S_3 + S_4 \\
S_{T'} &= (k-1)(M_n + M_e) - M_B, k = k_1 + k_2 + k_3 + k_4
\end{aligned}
$$

여기에서 공간 보장 알고리즘의 개수에 대한 조건으로

$$
k_2 + \frac{2\log(N \log N)}{M_e} k3 > \frac{(1-m)M_b - mM_n}{M_e} - m - l
$$

이 성립할 경우에 해당 서브트리를 B-변환 한다면 $S_{T'} > S_T$이 성립하므로 재변환으로 공간 이득은 증가한다. 따라서 에러 기준을 증가하였을 때 공간 이득은 증가하지 않는다. □

수정된 그리디 알고리즘은 에러 기준에 대하여 결과의 사이즈가 단조 감소하므로 이를 이용하여 공간 보장 알고리즘을 만드는 것은 어렵지 않다. 수정된 변환 알고리즘은 이미 변환된 서브트리에 대하여도 다시 한 번 가능한 B-변환을 수행한다. 이는 직관적으로 에러 기준이 증가함에 따라 서픽스 트리를 아래에서 위로 간단하게 만든다. 극단적으로 변환이 되었을 때 변환은 전체

서픽스 트리를 단 하나의 B-노드로 변환하여 결과 사이즈가 가장 작다. 에러를 전혀 허용하지 않을 경우에는 오직 같은 카운트를 갖는 노드들만이 병합되고 B-변환이나 V-변환되는 노드는 존재하지 않아 사이즈가 크게 줄지는 않는다. 공간 보장 알고리즘은 이 양극단 사이에서 목표 사이즈를 만족시키는 에러 기준을 이진 검색을 통하여 찾는다. 서픽스 트리의 루트의 카운트가 r일 때 공간 보장 알고리즘은 O(rNlog r)의 시간 비용을 갖는다.

4.2.4 혼합 알고리즘

앞에서 기술한 알고리즘은 모두 원래의 서픽스 트리, FST(Full Suffix Tree)에 적용되었으나 변환 알고리즘이 꼭 FST에 적용되어야 하는 것은 아니고 PST (Pruned Suffix Tree)에도 적용이 가능하다. 이 경우에는 포지티브 질의문도 부분적으로만 트리에 존재하므로 추측 알고리즘이 필요하다.

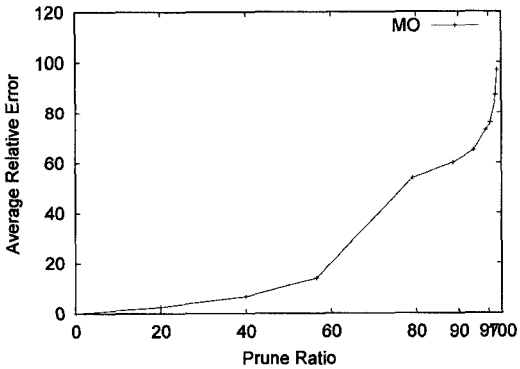


그림 6 가지치기 비율에 따른 추측의 정확도

그림 6은 노드를 잘라내는 비율에 따른 MO 알고리즘의 추측의 에러율 변화를 나타낸 그래프이다. 40% 정도까지는 노드들을 잘라내도 에러가 작고, 이후 서서히 증가하다가 90% 이상 노드를 잘라내면 에러가 급격히 증가함을 관찰할 수 있다. 이러한 경향을 이용하여 40~50%까지 노드를 잘라낸 PST에 변환 알고리즘을 적용하여 더 작게 만들고 여기에 추측 알고리즘을 적용하는 것이 혼합 알고리즘(Hybrid)이다. 혼합 알고리즘은 먼저 노드를 잘라내어 PST를 만들고 거기에 변환 알고리즘을 적용한다. 혼합 알고리즘의 결과는 부분문자열 정보를 축약하였으므로 PST와 마찬가지로 추측 알고리즘을 사용하여 선택도를 추측하여야 하고 이 추측 알고리즘에는 부분문자열 정보를 가지고 추측할 수 있는 어떤 알고리즘이라도 이용이 가능하다. FST를 변환한 경우는 기본적으로 추측알고리즘이 필요치 않고 선택도를 추측할 때 결과 서픽스 그래프를 검색하여 나온 카운트를 그대로 이용함을 상기하라.

5. 실험 결과

이 장에서는 제안하는 알고리즘의 성능에 대한 실험 결과를 기존의 여러 알고리즘과 비교하여 제시한다. 모든 실험은 Windows XP를 설치한 512MB의 메모리를 가진 2.80 GHz 펜티엄 4 PC에서 수행하였다.

5.1 실험 설정

데이타 집합. 알고리즘들을 실험하기 위하여 실생활 데이타와 생성한 데이타 양쪽을 모두 이용하였다. 실험에 사용된 실생활 데이타는 DBLP[11] 데이타베이스의 저자 이름 칼럼(authors)과 논문 제목 칼럼(titles)을 이용하였다. 그리고 두 가지 종류의 데이타 집합을 생성하였는데 모두 실생활 데이타인 이솝 우화, 천일야화, 셰익스피어의 희곡을 기반으로 생성하였다. 첫 번째 종류인 synthetic1은 사전으로부터 임의의 두 단어를 선택한 후 두 단어를 연결하여 생성하였다. 두 번째 종류인 synthetic2은 소스 텍스트에서 숫자나 문자를 임의로 선택하여 연결하여 생성하였다. 두 경우 모두 모집단이 되는 텍스트의 단어와 문장 분포 특성을 따랐다. 데이타 집합의 특성은 다음 테이블과 같다.

데이타	크기 (튜플)	튜플당 평균	
		토큰 개수	문자 개수
Author names	748,197	2.36	15.84
Paper titles	342,291	8.15	64.03
Synthetic1	100,000	2.56	20.16
Synthetic2	100,000	2.56	20.16

질의어의 선택. 일반적으로 사용되는 방법에 따라 질의어로는 '포지티브 질의어'와 '네거티브 질의어' 모두를 고려하였다. 각 데이타 집합에 대하여 포지티브 질의어는 질의 빈도에 관한 실험을 제외하고는 임의의 튜플에서 임의의 단어를 선택하여 구성하였다. 질의 빈도의 영향에 관한 실험에서는 단어들을 그 빈도에 따라 빈도 높음(high), 중간(middle), 낮음(low)의 세 가지 클래스로 나누어 분류하였다. '높음'의 분류에 속하는 단어들은 빈도가 상위 20%안에 드는 단어들 중 임의로 선택하였고, '중간'의 부류는 20~40%에 속하는 단어들, '낮음'의 분류는 그 외의 단어들 중에서 선택하였다. 네거티브 질의로는 데이타 집합에서 나타나지는 않으나 사전에서 추출한 단어들과 데이타 집합에서 임의의 단어를 선택한 후 문자들의 순서를 뒤집어 생성한 두 가지를 사용하였다. 첫 번째 집단은Markov 성질을 가지고 있고 두 번째 집단은 그렇지 않다고 볼 수 있다.

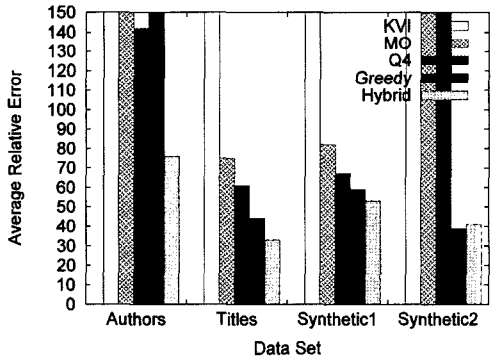
평가 기준. 포지티브 질의어에 대하여 사용한 기준은 *평균 상대 에러*인데, 이는 전체적인 추측의 정확도를 '|추측한 카운트 - 실제 카운트| /실제 카운트'로 측정하

는 방법이다. 그러나 이는 작은 선택도에 영향을 크게 받으므로 이 영향을 줄이기 위하여 널리 사용되는 방법[4,12,13]인 보정 상수(sanity constant)를 사용하였다. 이는 만약 실제 카운트가 보정 상수보다 작으면 보정 상수로 나누는 방법이다. 보정 상수로는 데이타 빈도 분포에서 1/4 위치에 존재하는 데이타의 카운트 값을 사용하였다. 또 다른 단점은 음의 상대 에러가 제한되어 선택도를 작게 추측하는 경향이 유리하다는 것이다. 따라서 상대 에러의 *에러 분포*를 또 다른 기준으로 삼아 추측의 전반적인 경향을 파악할 수 있도록 하였다. 전체 에러 범위를 [-100%, -90%), ..., [90%, 110%), [110%, ∞)로 나누어 각 버킷 안의 상대 에러 분포 개수를 표시하였다. 네거티브 질의인 경우에는 추측한 카운트와 실제 카운트의 차이의 절대값인 *평균 절대 에러*를 사용하였다.

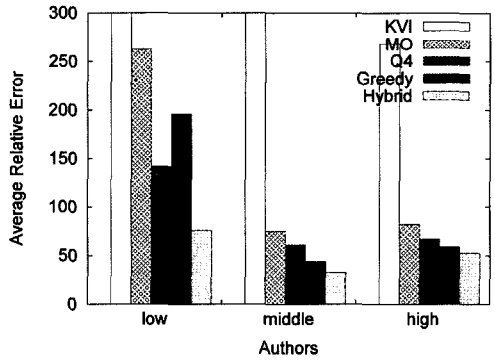
비교 알고리즘. 제안한 여러 알고리즘 중 기본 그리디 알고리즘(Greedy)과 혼합 알고리즘(Hybrid)을 다른 최신의 알고리즘들과 비교하였다. 혼합 알고리즘의 경우에는 MO를 추측 알고리즘으로 사용하였다. 비교의 대상이 된 알고리즘은 KVI와 MO알고리즘이다. Krishnan 등[1]은 서픽스 트리에서의 여러 추측 알고리즘들을 제안하였는데 그 중 알고리즘 $I_{1'}$을 KVI로 사용하였다. Greedy를 제외하고는 모든 실험에는 같은 크기의 공간을 할당하였는데 실험한 자료 구조에 따라 원래 서픽스 트리의 10 %의 공간을 할당하였다. Greedy는 10 %이하로 사이즈를 줄이면 에러가 급격히 증가하여서 15 %로 하였으므로 다른 알고리즘과 직접적인 비교는 할 수 없고 경향만을 파악할 수 있다.

5.2 실험 결과

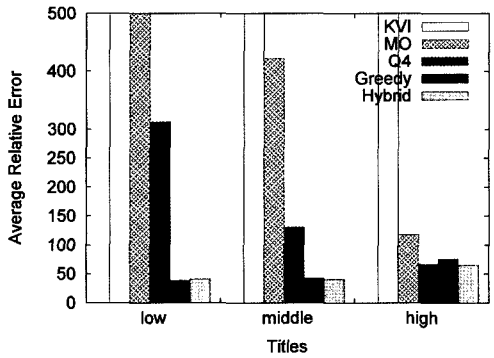
포지티브 질의어에 대한 추측의 정확도. 그림 7(a)는 다양한 데이타 집합과 질의어에 대한 평균 상대 에러를 그린 그래프이다. Greedy와 Hybrid는 모든 데이타군에서 다른 알고리즘들보다 정확한 성능을 보였다. 그러나 Hybrid의 정확도는 titles 데이타군과 synthetic2 데이타군에서 상대적으로 저하되었다. 이 Hybrid가 추측 알고리즘으로 MO를 사용한데 기인하는 것으로 이 데이타들에 대하여 MO가 다른 경우들에 비해서 낮은 성능을 보임을 마찬가지로 확인할 수 있다. 이를 좀 더 분석하여 보면 테이블5.1에서 titles 데이타가 authors 데이타에 비하여 더 많은 토큰들과 문자들을 가지고 있



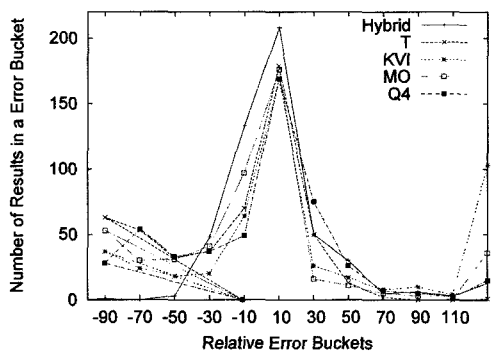
(a) 데이타 집합에 따른 평균상대에러



(b) 질의 빈도에 따른 평균상대에러(Authors)



(c) 질의 빈도에 따른 평균상대에러(Titles)



(d) 에러 분포

그림 7 상대 에러

음을 알 수 있다. synthetic2 데이타도 synthetic1에 비하여 좀 더 규칙성이 적다는 특성을 가지고 있다. 이러한 사실들은 MO나 MO를 보조루틴으로 사용하는 Hybrid는 데이타의 임의성이 커지면 성능이 저하됨을 암시한다. 만약 MO보다 더 정확한 추측 알고리즘을 사용하면 성능이 더 향상될 것이다.

그림 7(b),(c)는 질의어의 여러 빈도에 따른 평균 상대 에러를 비교한다. 마찬가지로 Greedy, Hybrid가 우수한 성능을 보여주었는데, Hybrid는 높은 빈도의 질의어에 대하여는 가장 정확한 추측을 하였으나 낮은 빈도의 질의어에 대하여는 상대적으로 성능이 저하되었다. 이는 역시 MO가 질의어의 빈도수가 낮아짐에 따라 에러가 증가하는 관찰과 일치한다. 높은 빈도의 질의어들은 PST에도 존재할 가능성이 높아서 이러한 경향을 보이는 것으로도 추측할 수 있다. 이에 비하여 Greedy는 원래의 FST를 바로 축약하므로 이러한 영향을 상대적으로 적게 받았다.

그림 7(d)는 각 알고리즘의 평균 상대 에러의 분포를 보여준다. 이전에 기술했듯이 KVI와 MO는 선택도를 낮게 추측하는 경향이 있는 문제점을 보여 많은 에러가 [-100%, -70%) 버킷에 존재하고 있음을 관찰할 수 있다. 이에 반해 Hybrid와 Greedy는 대부분의 에러가 [-30%, 30%)의 버킷에 존재하였고 매우 적은 수의 에러들만이 범위의 양쪽 끝에 존재하였다. 이는 뚜렷하게 Hybrid와 GREEDY는 선택도를 낮거나 높게 편향되어 추측하는 경향이 없음을 나타낸다.

네거티브 질의어에 대한 추측의 정확도. 네거티브 질의어를 실험하기 위해서는 authors와 titles 데이타 집합을 이용하였다. 질의어는 각 데이타 집합을 위하여 사전에는 있으나 해당 데이타에는 존재하지 않는 단어를 임의로 500개씩 추출하여 사용하였다. 다음 테이블은 네거티브 질의어에 관한 실험 결과를 보여준다.

추측알고리즘	저자명	논문 제목
KVI	7.4	6.6
MO	68	65
Hybrid	3.4	3.3

Hybrid는 네거티브 질의어에 대하여도 매우 정확하게 추측을 하였다. 그리고 약간 특이하게 KVI가 Hybrid와 거의 비슷한 성능을 보였다. 하지만 이는 KVI가 정확하게 추측을 하여서가 아니라 매우 심하게 선택도를 낮게 추측하는 경향이 있어서 나타난 결과이다. Hybrid는 그런 경향을 보이지 않으면서도 KVI에 비해서도 두 배 이상 정확하게 MO에 비해서는 스무배 이상 정확하게 추측을 하였다. 이는 블룸 필터가 매우 효율적으로 위양성을 걸러내는 역할을 하였기 때문으로 분석할 수 있다.

6. 관련 연구

부분문자열 선택도 추측. Krishnan 등[1]은 최초로 부분문자열의 선택도를 추측하는 문제를 제안했고 KVI를 포함하여 가지치기된 서픽스 트리를 사용하는 다양한 알고리즘들을 제안하였다. Wang 등[2]은 이 문제를 두 문자-숫자 칼럼이 서로 상관 관계를 가진 경우로 확장하였고, 이는 이후 [14]에서 개선되었다. Jagadish 등[3]은 추측 알고리즘의 정확도에 초점을 맞추어 MO, MOC, MOLC를 제안하였다.

Chen 등[4]은 불리언 질의의 선택도를 추측하는 좀 더 일반적인 문제를 고려하였다. 그들의 프레임워크는 [3]와 비슷하나 그들은 노드에 해당하는 문자열을 포함하는 문자열들(본 논문에서는 튜플)을 축약하여 표현하기 위하여 해시를 사용하는 집합 해싱 기법을 사용하였다.

XML 경로 선택도 추측. XML 경로 선택도 추측의 문제는 부분문자열 선택도 추측 문제와 비슷한 특성을 지녔다. 사실 부분문자열 선택도 추측 문제는 XML 경로 선택도 추측의 문제에서 단순 경로(simple path)의 선택도를 추측하는 부분 문제라고 볼 수 있다. XML 경로 선택도 추측은 이에 더하여 가지 경로(twig path), 조상-후손 경로(ancestor-descendant path)등 더 많은 문제를 풀어야 한다. 하지만 반드시 부분문자열 문제가 XML 경로 문제보다 쉽지는 않다. 부분문자열 문제는 훨씬 처리해야 할 실행할 데이타의 크기가 크기 때문이다. 예를 들어 DBLP 데이타로 부분문자열을 나타내는 서픽스 트리는 수십 메가바이트를 차지하지만 경로를 나타내는 서픽스 트리는 몇 십 키로 바이트만을 차지한다. XML 경로 선택도 문제와 부분문자열 선택도 문제의 유사성을 이용하여 Chen 등[15]은 그들의 부분문자열 선택도 추측에 관한 이전 연구를 이 문제에 적용하여 풀기도 하였다. 그들은 집합 해싱 기법을 가지 경로를 처리하는 데 사용하였다. Lim 등[12]은 노드를 병합하기 위하여 본 논문의 변환과 비슷한 방법을 사용하였다. 그러나 그들의 연구에서 변환된 노드는 항상 변환되면서 구조적 정보를 읽는다. 그리고 손실된 구조 정보를 Markov 히스토그램을 이용하여 일부 보전한다.

7. 결 론

우리는 부분문자열의 선택도를 추측하기 위하여 변환에 기반을 둔 방법을 제안하였다. 제안하는 알고리즘은 서픽스 트리의 노드들을 항상 잘라내어 없애버리기 보다는 일정한 에러를 허용하면서 서픽스 트리를 서픽스 그래프로 변환한다. 그리고 기본 변환 알고리즘을 이용하여 에러 한도를 보장하는 알고리즘과 공간 한도를 보장하는 알고리즘도 제안하였다. 이 중 에러 한도를 보장

하는 알고리즘은 부분문자열 연구에서 최초의 것이다. 또 제안하는 알고리즘은 PST에도 적용이 가능하여 기존의 추측 알고리즘과 혼합하여 사용할 수도 있다. 실험 결과는 제안하는 알고리즘이 기존의 알고리즘 보다 훨씬 적은 평균 에러를 가지고 고른 에러 분포의 특성을 가지고 있음을 보여준다.

참 고 문 헌

[1] P. Krishnan, Jeffrey Scott Vitter, and Bala Iyer. Estimating Alphanumeric Selectivity in the Presence of Wildcards. In Proceedings of the ACM SIGMOD, 1996.

[2] Min Wang, Jeffrey Scott Vitter, and Bala Iyer. Selectivity Estimation in the Presence of Alphanumeric Correlations. In IEEE International Conference on Data Engineering, 1997.

[3] H. V. Jagadish, Ol Kapitskaia, Raymond T. Ng, and Divesh Srivastava. One dimensional and multidimensional substring selectivity estimation. VLDB journal, 9:214-230, 2000.

[4] Zhiyuan Chen, Flip Korn, Nick Koudas, and S. Muthukrishnan. Selectivity Estimation For Boolean Queries.In Proceedings of ACM Symposium on Principles of Database Systems, 2000.

[5] E. M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. Journal of the ACM, 15:514-534, 1976.

[6] Naresh Neelapala, Romil Mittal, and Jayant R. Haritsa. SPINE: Putting Backbone into String Indexing. In IEEE International Conference on Data Engineering, 2004.

[7] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422-426, 1970.

[8] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Transaction on Networks, 8(3):281-293, 2000.

[9] Wei Wang, Haifeng Jiang, Hongjun Lu, and Jeffrey Xu Yu. Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. In Proceedings of the Conference on VLDB, 2004.

[10] Kenneth H. Rosen. Elementary Number Theory. Addison-Wesley Longman, Inc., 1988.

[11] M. Ley. Dblp. http://www.fnformatick.uni-tier.de/ley/db.

[12] L. Lim, Min Wang, S. Padmanabhan, Jeffery Scott Vitter, and R. Parr. XPathLearner: An on-line self-tuning Markov histogram for XML path selectivity estimation. In Proceedings of the Conference on VLDB, 2002.

[13] Surajit Chaudhuri, Venkatesh Ganti, and Luis Gravano. Selectivity Estimation for String Predicates: Overcoming the Underestimation Problem. In IEEE International Conference on Data Engineering, 2004.

[14] H. V. Jagadish, Olga Kapitskaia, Raymond T. Ng, and Divesh Srivastava. Muti-Dimensional Substring Selectivity Estimation. In Proceedings of the Conference on Very Large Data Bases, 1999.

[15] Zhiyuan Chen, H. V. Jagadish, Flip Korn, Nick Koudas, S. Muthukrishnan, Raymond T. Ng, and Divesh Srivastava. Counting Twig Matches in a Tree. In IEEE International Conference on Data Engineering, 2001.

이 홍 래

1992년 3월~1997년 2월 서울대학교 원자핵공학과. 2002년 3월~2004년 2월 서울대학교 컴퓨터공학과 대학원 석사과정

심 규 석

1986년 서울대학교 전기공학과 졸업(학사). 1988년 University of Maryland, College Park(석사). 1993년 University of Maryland, College Park(박사). 1993년~1994년 Federal Reserve Board, Research Staff. 1994년~1996년 IBM Almaden Research Center, Research Staff. 1996년~2000년 Bell Lab, MTS. 1999년~2002년 KAIST 전산학과 조교수. 2002년~현재 서울대학교 전기컴퓨터공학부 부교수. 현재 VLDB저널과 IEEE TKDE저널의 Editorial Board, ACM SIGMOD, VLDB, ICDE, ACM SIGKDD, ICDM, EDBT를 비롯 다수의 국제 학술대회의 Program Committee Member임. 관심분야는 데이타마이닝, 데이타베이스, XML, Stream Data

김 형 주

1982년 서울대학교 전산학(학사). 1985년 미국 텍사스 대학교 전산학(석사). 1988년 미국 텍사스 대학교 전산학(박사) 1998년 5월~1988년 9월 미국 텍사스 대학교 Post Doc. 1988년 9월~1990년 12월 미국 조지아 공과대학 조교수. 1991년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 데이타베이스, XML, 시맨틱웹, 온톨로지 등