pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2013.2.4.267

Efficient Similarity Joins by Adaptive Prefix Filtering

Jong Soo Park†

ABSTRACT

As an important operation with many applications such as data cleaning and duplicate detection, the similarity join is a challenging issue, which finds all pairs of records whose similarities are above a given threshold in a dataset. We propose a new algorithm that uses the prefix filtering principle as strong constraints on generation of candidate pairs for fast similarity joins. The candidate pair is generated only when the current prefix token of a probing record shares one prefix token of an indexing record within the constrained prefix tokens by the principle. This generation method needs not to compute an upper bound of the overlap between two records, which results in reduction of execution time. Experimental results show that our algorithm significantly outperforms the previous prefix filtering-based algorithms on real datasets.

Keywords : Similarity Join, Prefix Filtering, Algorithm, Performance Evaluation

맞춤 접두 필터링을 이용한 효율적인 유사도 조인

박 종 수†

요 약

데이터 정제나 복사 탐지와 같은 많은 응용들을 가진 중요한 연산인 유사도 조인은 도전적인 주제로 데이터집합에서 주어진 한계치 이상의 유사도를 가지는 모든 쌍의 레코드들을 찾는 것이다. 우리는 빠른 유사도 조인을 위해 후보 쌍들의 생성 시에 접두 필터링 원리를 강한 제약 조건으로 사용하는 새 알고리즘을 제안한다. 그 원리에 의해 한정된 접두 토큰들내에서 탐색 레코드의 현재 접두 토큰이 인덱싱 레코드의 접두 토큰을 공유할 때에만 후보 쌍이 생성된다. 이 생성 방법은 두 레코드들 사이에 공통부분의 상한 값을 계산할 필요가 없어서 실행시간을 감소시킨다. 실제 데이터 집합에 적용된 실험 결과는 제안된 알고리즘이 이전의 접두 필터링 방법의 알고리즘들에 비해 상당히 우수함을 보여준다.

키워드 : 유사도 조인, 접두 필터링, 알고리즘, 성능 평가

1. 서 론

인터넷의 사용과 모바일 컴퓨터의 발전으로 대용량의 데이터가 급속히 증가하고 다양한 종류의 데이터를 관리해야 하는 상황이다. 대용량 데이터베이스에서 문장이나 단어들이 주어진 한계치 이상의 유사성을 가진 문서나 웹 페이지들을 찾아내는 유사도 조인(similarity join) 문제는 데이터 정제나 복사 탐지 등의 많은 응용 분야를 가지는 중요한 연산이다[1, 2, 3].

유사도는 두 개체들을 구성하는 토큰들이 얼마나 많이 유사한 정도를 나타내는 [0, 1] 사이의 값을 계산해내는 유사도 함수(similarity function)로 측정된다[2]. 유사도 조인은 데이터집합에서 주어진 한계치 이상의 유사도를 갖는 모든 쌍의 레코드들을 찾아내는 연산이다. 유사도를 측정하는 함수들로는 Jaccard similarity, Cosine similarity, Overlap similarity 등이 있다[2, 4, 5]. 본 논문에서는 주로 Jaccard similarity에 초점을 맞춘다.

대부분의 인덱스-기반 유사도 조인 알고리즘들은 두 단계인 후보 생성 단계와 검증 단계로 이루어진다[5-7]. 조인 후보를 작게 생성하는 접근 방식과 검증을 빠르게 하는 방법 등이 제안되고 있다. 본 논문에서는 조인 후보를 조건 제한으로 간단히 생성하는 방법과 검증을 빠르게 하는 방법을 적용한 새로운 유사도 조인 알고리즘을 제안한다. 많은 알고리즘의 기본 개념에 해당되는 prefix filtering principle[2, 7]을 유사도 조인 알고리즘에 맞춤 방식으로 적용하여 빠른 실행 속도를 가진 방법론을 모색한다.

2절에서 유사도 함수와 prefix filtering의 원리를 서술하고 대표적인 유사도 조인 알고리즘의 주요 특성을 설명한

※ 이 논문은 2012년도 성신여자대학교 학술연구조성비 지원에 의하여 연구되었음.

† 종신회원 : 성신여자대학교 IT학부 교수

논문접수 : 2012년 9월 7일
수 정 일 : 1차 2012년 11월 9일
심사완료 : 2012년 12월 1일

* Corresponding Author : Jong Soo Park(jpark@sungshin.ac.kr)

다. 3절에서 제안하는 맞춤 prefix filtering의 내용을 설명하고 알고리즘을 기술한다. 4절에서 실험결과를 테이블과 그림으로 보여주고, 5절에서 결론을 맺는다.

2. 유사도와 접두 필터링 방법

유사도 함수(similarity function)의 선택은 응용 영역에 따라 아주 큰 영향을 받게 되는 데 본 논문에서는 널리 사용되고 있는 쟈카드 유사도(Jaccard similarity)와 공통부분 유사도(Overlap similarity)에 대해서만 토론한다. 레코드 x는 |x|개의 토큰들로 구성된다. 두 레코드 x와 y를 고려하면 두 유사도는 다음과 같이 정의된다[5, 7].

$$\text{Jaccard similarity: } J(x,\ y) = \frac{|x \cap y|}{|x \cup y|}.$$

$$\text{Overlap similarity: } O(x,\ y) = |x \cap y|.$$

[0, 1] 사이의 값으로 주어지는 쟈카드 유사도의 한계치를 t라 하면 쟈카드 유사도와 공통부분 유사도는 다음과 같은 관계식을 가진다[5, 7].

$$J(x,\ y) \geq t \ \Leftrightarrow \ O(x,\ y) \geq \alpha = \frac{t}{1+t} \cdot (|x|+|y|), \quad (1)$$

앞에서 설명한 기호를 사용하여 prefix filtering을 적용할 때 가장 기본적인 원리를 보조정리 1에서 설명한다[2, 7].

보조정리 1: 접두 필터링 원리(Prefix Filtering Principle) 토큰 전체 집합 U의 순서를 이루는 O와 레코드들의 집합이 주어지면, 토큰들 각각은 O의 순서에 따라 정렬이 된다. 레코드 x의 p-prefix를 x에서 앞에서 첫 p 개의 토큰들로 이루진다고 하자. 만약 $O(x,\ y) \geq \alpha$이면, 그러면 x의 $(|x|-\alpha+1)$-prefix와 y의 $(|y|-\alpha+1)$-prefix는 적어도 한 개의 토큰을 공유해야 한다.

유사도 조인 알고리즘들 중에서 대표적인 PPJoin 알고리즘[5, 7]은 All-Pairs 알고리즘[4]에 보조정리 1의 Prefix filtering과 비교하는 두 레코드의 토큰들의 상대적인 위치에 제한을 하는 Positional filtering을 결합하여 만든 알고리즘이다. 이 알고리즘은 너무 많은 조인 후보 쌍들을 만들어지게 하여 조인 검증 시간이 많이 소요되는데, 조인 후보 쌍들을 감소시키는 suffix filtering을 적용한 PPJoin+ 알고리즘도 제안하였다[7].

MPJoin 알고리즘[6]은 PPJoin 알고리즘을 개선한 유사도 조인 알고리즘으로 인덱싱 레코드의 토큰이 조인 후보 쌍이 될 가능성이 없으면 inverted index에서 그 항목을 삭제하여 성능을 개선하는 특징을 가진다. 그러나 MPJoin 알고리즘[7]의 중간 단계에서 inverted index에서 가져온 인덱싱 토큰 (x_c, j)를 삭제하기 전에 x_c의 overlap score를 $-\infty$로 설정하는 단계(line 12, $M(x_c).os \leftarrow -\infty$)로 인하여 틀린 결과가 나옴을 실험으로 확인하였다. 이를 해결하는 한 방법으로 이 단계를 그 다음 두 단계들(lines 13-14) 다음에 오게 하고 토큰 x_c의 overlap score의 값을 0으로 설정하면 정확한 결과를 얻을 수 있다. 여기서 토큰 (x, j)는 레코드 x에서 j 번째의 토큰을 표시한다.

3. 맞춤 접두 필터링 알고리즘

보조정리 1을 적용하여 기존의 유사도 조인 알고리즘을 개선할 수 있는 부분은 비교하는 탐색 레코드 x와 인덱싱 레코드 y 사이의 prefix 범위를 미리 맞춤 방식으로 찾아내어 이 범위 내에서 유사도 조인 후보 쌍들을 빠르게 결정하는 것이다. Equation (1)과 보조정리 1로부터 탐색 레코드 x와 인덱싱 레코드 y가 접두 필터링에 사용되는 토큰들의 개수를 각각 prefix_x와 prefix_y로 아래 Equations (3)과 (4)로 나타내고, α는 Equation (1)에서 최소 공통부분 유사도로 Equation (2)에 다시 표시한다. 탐색 레코드 x의 prefix로 올 수 있는 최대 크기 max_probe_prefix는 Equation (3)에서 α 값이 최소가 되는 $|y| = t \cdot |x|$일 때 Equation (5)로 나타낸다. 여기서 t는 쟈카드 유사도 한계치이다. 이후의 처리 과정에서 x의 토큰들이 인덱싱 레코드의 토큰들로 사용될 때 inverted index에 저장되어야 될 최대 크기 max_index_prefix는 Equation (6)으로 계산된다[5-7].

$$\alpha = \left\lceil \frac{t}{t+1} \cdot (|x|+|y|) \right\rceil, \quad (2)$$

$$\text{prefix\_x} = |x| - \alpha + 1, \quad (3)$$

$$\text{prefix\_y} = |y| - \alpha + 1, \quad (4)$$

$$\text{max\_probe\_prefix} = |x| - \lceil t \cdot |x| \rceil + 1, \quad (5)$$

$$\text{max\_index\_prefix} = |x| - \left\lceil \frac{2t}{t+1} \cdot |x| \right\rceil + 1. \quad (6)$$

PPJoin[7]에서는 탐색 레코드의 탐색 토큰 (x, i)의 inverted index에서 가져온 인덱싱 토큰 (y, j)에서 $|y| \leq t \cdot x$이면 삭제하고 x의 현재 위치 i와 y의 위치 j를 고려하여 위치 조건이 맞으면 overlap 값을 계산하여 유사도 조인 후보로 결정한다. MPJoin[6]에서 토큰을 삭제하는 방법은 먼저 탐색 레코드 x의 prefix_size를 Equation (6)에 따라 초기값으로 설정하고 이 x의 토큰이 inverted index에 저장되어 다음 처리에서 인덱싱 토큰으로 될 때 그 시점에서 Equation (4)와 같이 계산되어 prefix_size가 동적으로 갱신된다. 이후 그 인덱싱 토큰 (y, j)가 조인 후보 여부를 결정하는 시점에 j가 y의 prefix_size 보다 크면 토큰 (y, j)는 삭제된다.

본 논문에서 제안하는 APJoin 알고리즘은 다음과 같은

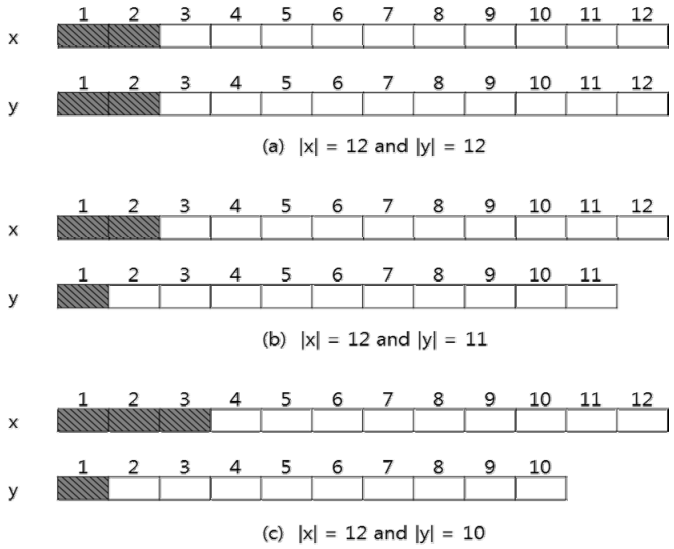

Fig. 1. Prefix size and |y| for t = 0.8 and |x| = 12

특성을 가지고 있다: 첫째, PPJoin과 MPJoin에서 사용한 prefix filtering과 positional filtering을 적용한다. 둘째, MPJoin에서 사용한 동적으로 갱신되는 인덱싱 레코드의 prefix_size를 적용하는 방법을 탐색 레코드와 인덱싱 레코드의 prefix 범위 설정에 이용하였다. 즉, 한 레코드가 탐색 레코드로 처음 고려될 때 이 레코드의 prefix의 범위를 Equation (3)으로부터 설정하고, 이 레코드와 prefix 범위 안에 들어올 수 있는 인덱싱 레코드의 prefix의 범위를 Equation (4)로부터 맞춤 방식으로 초기에 설정한다. 셋째, 탐색 레코드와 인덱싱 레코드의 prefix 범위가 설정되어 있기 때문에 비교되는 탐색 토큰과 인덱싱 토큰의 두 레코드가 설정된 prefix 범위 안에 있으면 유사도 조인 후보 쌍으로 결정된다. 유사도 조인 후보 쌍을 결정할 때, PPJoin과 MPJoin에서는 현재 공통부분 토큰들의 개수, 남아있는 토큰들의 개수, 그리고 α 값을 계산하여 결정하는데, APJoin에서는 prefix 범위 이내인지 아닌 지만을 검사한다. APJoin 알고리즘에서 탐색 레코드 x의 i번째 토큰 (x, i)와 이 토큰의 inverted index에서 가져온 인덱싱 레코드 y의 j번째 토큰 (y, j)를 비교하여 두 레코드를 유사도 조인 후보 쌍으로 결정하는 과정은 다음과 같다:

> 만약 y의 크기가 $t \cdot |x|$ 보다 작으면, 토큰 (y, j)를 삭제한다; 그렇지 않고 만약 j가 y의 prefix 범위를 벗어나면, 토큰 (y, j)를 삭제한다; 그렇지 않고 만약 i가 x의 prefix 범위를 벗어나면, 다음 토큰으로 이동한다; 그렇지 않으면, x와 y가 유사도 조인 후보 쌍이 된다.

탐색 레코드와 인덱싱 레코드의 prefix 범위는 Equations (3)과 (4)에서 계산하고 배열에 저장한다. Fig. 1과 Table 1은 $t = 0.8$이고 탐색 레코드의 크기 $|x| = 12$일 때, 고려될 수 있는 인덱싱 레코드의 크기는 $\lceil t \cdot |x| \rceil \le |y| \le |x|$가 되어 |y| = 10, 11, 12가 된다. Table 1의 왼쪽 세 열과 Fig. 1에서 이런 경우의 레코드들에 대하여 설명하고 있다. Fig.

Table 1. Prefix size with varying |x| and |y| for t = 0.8

	|x| = 12			|y| = 12			
|x|	**12**	**12**	**12**	12	13	14	15
|y|	12	11	10	**12**	**12**	**12**	**12**
α	11	11	10	11	12	12	12
prefix_x	2	2	3	2	2	3	4
prefix_y	2	1	1	2	1	1	1

1에서 빗금 친 칸들로 탐색 레코드와 인덱싱 레코드의 prefix의 범위를 설명하고, Fig. 2의 알고리즘에서 해당 경우의 prefix의 값으로 prefix_x[1:3] = {2, 2, 3}과 prefix_y[1:3] = {2, 1, 1}로 계산된다. 두 레코드의 비교 토큰들의 위치가 주어진 prefix 범위를 벗어나면 유사도 조인 후보에서 제외된다. Table 1의 $|y| = 12$인 마지막 네 열에서는 크기가 12인 탐색 레코드 x가 다음 처리 과정에서 인덱싱 레코드 y로 사용되는 경우에 해당되는 변수들의 값을 설명하고 있다. prefix_y의 값들 중에서 최대값은 2가 되어 |x| = 12일 때 max_index_prefix = 2이므로 서로 같음을 보여주고 있다. 이것은 크기가 12인 탐색 레코드 x의 토큰들 중에서 앞에서부터 Equation (6)의 max_index_prefix 만큼만 inverted index에 저장하면 충분하다는 것을 의미한다.

Fig. 2에서 알고리즘 APJoin의 입력은 레코드들의 집합 R과 주어진 쟈카드 유사도 한계치 t이고, 출력으로 한계치 t 이상의 값을 갖는 모든 레코드 쌍들을 찾아낸다. 각 레코드는 토큰들로 구성되고, 레코드들은 토큰들의 개수에 따라 비내림 차순으로 정렬되어 있다. 단계 5 - 10에서 Equations (2) - (6)에 해당되는 변수들을 계산한다. 단계 8에서 탐색 레코드 x가 주어지면 인덱싱 레코드 y의 크기가 결정되어 다른 식들에 사용된다. 단계 14 - 21은 탐색 토큰 (x, i)와 인덱싱 토큰 (y, j)로 앞에서 설명한 조건 검사만으로 유사도 조인 후보 쌍을 결정한다. 조건 검사에 맞지

```
Algorithm: APJoin(R, t)
1  S ← ∅;
2  I_w ← ∅ (1 ≤ w ≤ |U|);     // initialize inverted index
3  for each x ∈ R do
4    A ← empty map from record id to int;
5    max_probe_prefix ← |x| − ⌈t · |x|⌉ + 1;          // Eq. (5)
6    max_index_prefix ← |x| − ⌈2|x| · t/(t+1)⌉ + 1; // Eq. (6)
7    for i = 1 to max_probe_prefix do
8      α = ⌈(|x|+|x|−i+1) · t/(t+1)⌉;      // |y| = |x|−i+1
9      prefix_x[i] = |x|−α+1;                    // Eq. (3)
10     prefix_y[i] = (|x|−i+1)−α+1;          // Eq. (4)
11   for i = 1 to max_probe_prefix do
12     w ← x[i];                          // 레코드 x의 토큰 (x, i)
13     for each (y, j) ∈ I_w do            // 레코드 y의 토큰 (y, j)
14       if |y| < t · |x|                  // 현재 노드 이후 삭제
15         (y, j)와 연결된 노드들 삭제;
16       else if j > prefix_y[|x|−|y|+1]
17         (y, j) 삭제;                    // 현재 노드 삭제
18       else if i > prefix_x[|x| − |y| + 1]
19         continue;                       // 다음 노드로 이동
20       else
21         A[y] = A[y] + 1;                // candidate generation
22   for i = 1 to max_index_prefix do
23     w ← x[i];
24     I_w = I_w ∪ {(x, i)};       // 토큰 (x, i)를 inverted index에
25   VerifyZip(x, A, α);                   // verification phase
26 return S
```

Fig. 2. Algorithm APJoin

않는 현재 토큰이 유사도 조인 후보가 될 가능성이 전혀 없으면 inverted index에서 삭제하여 이후 비교 검사되는 인덱싱 레코드의 토큰들의 개수를 줄인다. 단계 18의 조건은 Fig. 1의 (a)와 (b)의 탐색 레코드 x의 3번째 토큰과 같이 현재 인덱싱 레코드의 토큰과 조인 후보가 될 가능성이 없으면 현재 토큰은 처리하지 않고 다음 토큰으로 이동한다. x의 3번째 토큰은 Fig. 1의 (c)에서 사용될 것이다. 단계 21에서 탐색 토큰 (x, i)와 인덱싱 토큰 (y, j)가 두 레코드 x와 y의 prefix 범위 안에 들어오기 때문에 x와 y가 유사도 조인 후보 쌍으로 생성된다. 단계 22 – 24에서 앞으로 사용할 토큰들을 inverted index에 저장한다. 단계 25의 함수 VerifyZip은 지퍼 합병 방식으로 공통부분을 계산하여 유사도 조인 쌍을 결정하는 함수다. 이 함수에서 두 레코드의 토큰들을 비교하면서 현재 공통부분 토큰들의 개수, 남아있는 토큰들의 개수, 그리고 α 값을 비교하여 계속 비교 검증할 것인 지 또는 중단할 것인 지를 결정하여 빠르게 유사도 조인 후보 쌍을 검증한다.

4. 실험 결과 및 분석

본 논문에서 제안된 AP_Join 알고리즘과 최근의 유사도 조인 알고리즘들 중에서 비교 대상이 되는 PPJoin, PPJoin+, MPJoin 알고리즘을 C++ 언어로 구현하여 성능을 비교 분석하였다. 실험에 사용된 컴퓨터의 운영체제는 MS Windows 7 64bit이고, 프로그램 개발 환경으로 MS Visual Studio 2008을 사용하였다. 사용된 컴퓨터 하드웨어의 주요 사양으로 CPU가 Intel i920 2.67GHz이고 메인 메모리는 16GB DDR3 RAM이다.

실험 데이터는 DBLP 웹 사이트에 있는 참고문헌 레코드들로 구성된 데이터와 Enron 회사의 email들로 구성된 데이터를 사용하였고[4-7], 각각 DBLP와 ENRON 데이터라고 한다. DBLP 데이터는 988,567개의 레코드들로 이루어지고, 각 레코드는 출판물의 저자 이름과 제목에 포함된 단어들을 정수 토큰들로 변환되어 전체 토큰들의 개수는 약 71만개고 레코드들의 평균 토큰 개수는 15.2개이다. ENRON 데이터는 245,481개의 이메일들로 구성되고, 각 메일은 공백이나 특수문자로 분리된 단어들을 추출하여 정수 토큰들로 변환되었다. ENRON 데이터에서 전체 토큰들의 개수는 약 236만개이고 이메일들의 평균 토큰 개수는 285.5개이다.

알고리즘의 성능 평가를 위해 동일한 시간 복잡도의 범주를 갖는 네 알고리즘들을 프로그래밍하여 DBLP 데이터를 입력으로 한 실험결과를 Table 2에서 보여주고 있다. Table 2의 |Join|은 유사도 조인 결과 쌍들의 개수이고, |Cand|는 유사도 조인 후보 쌍들의 개수이고, Time은 실행시간을 나

타낸다. Table 2의 |Comp|는 탐색 토큰과 인덱싱 토큰을 검사하여 유사도 조인 후보 쌍을 결정하는 단계들의 회수를 나타내는 값이다. MPJoin의 |Comp|값이 PPJoin에 비해서 작은 것은 인덱싱 토큰의 위치 j가 prefix 범위 보다 크면 삭제되기 때문이고, APJoin의 |Comp|가 더 작은 것은 탐색 레코드의 토큰 위치와 인덱싱 레코드의 토큰 위치가 설정된 prefix 범위 이내에서만 유사도 조인 후보쌍을 결정하기 때문이다. |Comp|가 작아지면 실행시간도 작아지게 된다. 실행시간 비교에서 본 논문에서 제안된 APJoin은 실행속도 면에서 PPJoin에 비해 68% 그리고 MPJoin에 비해 32%의 성능 개선을 보여주고 있다. Fig. 3과 Fig. 4는 쟈카드 유사도 한계치 t가 변화할 때 주어진 입력 데이터에 따라 각 알고리즘의 실행 시간을 보여준다. DBLP 데이터에서는 APJoin 알고리즘의 실행시간이 MPJoin에 비해 32% 정도 일정하게 더 빨리 수행됨을 보여주고, ENRON 데이터에서 APJoin 알고리즘의 상대적인 실행시간은 유사도 한계치 t가 0.95일 때 4%에서 t가 0.75일 때 33%로 점차 개선됨을 보여준다. Fig. 4에서 ENRON 데이터의 레코드들이 DBLP 데이터에 비해서 토큰들의 평균 개수가 많음으로 인해 실행시간이 많이 소요되고 알고리즘들의 상대적인 실행 시간 특성이 Fig. 3과 다름을 보여준다.

APJoin 알고리즘에서 개선해야 될 부분은 Table 2의 |Cand|로 표시된 유사도 조인 후보 쌍들의 개수를 줄이는 것과 ENRON 데이터와 같이 평균 토큰 개수가 큰 레코드들을 처리하는 시간을 줄이는 것이다. 특히, APJoin 알고리즘에서 PPJoin+ 알고리즘과 같이 후보 수를 줄이는 방법을 모색하면 더 좋은 성능을 갖는 알고리즘이 될 것이다.

Table 2. Experimental results for t = 0.85 on DBLP

	|Comp|	|Cand|	|Join|	Time(sec)
PPJoin	4,132,651	1,356,337	3,683	2.106
PPJoin+	4,132,651	11,905	3,683	1.810
MPJoin	2,200,083	1,356,628	3,683	0.978
APJoin	1,378,845	1,357,015	3,683	0.668

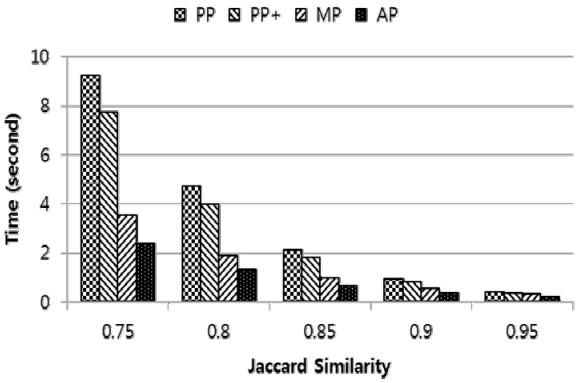

Fig. 3. Execution times on DBLP

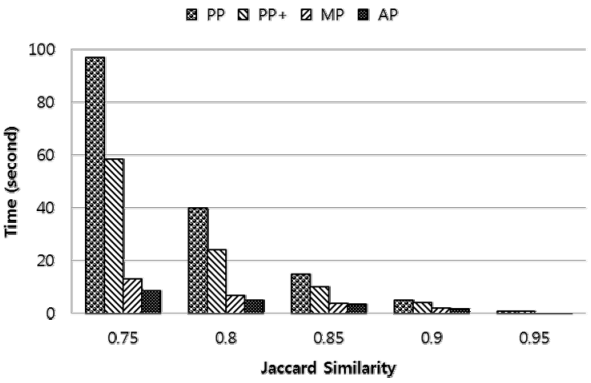

Fig. 4. Execution times on ENRON

5. 결 론

본 논문에서 각 레코드가 토큰들로 이루어진 대용량의 레코드들 중에 서로 유사한 레코드들을 찾아내는 효과적인 유사도 조인(similarity join) 알고리즘을 제안한다. 이 알고리즘은 접두 필터링(prefix filtering) 원리에 따라 탐색 레코드와 인덱싱 레코드의 접두 토큰들의 개수를 설정하여 이 범위 내에 들어오는 토큰이 일치하면 유사도 조인 후보로 만들어 검증한다. 실제 데이터를 사용한 실험을 통하여 제안된 알고리즘이 기존의 접두 필터링에 기반한 알고리즘들에 비해서 더 빠르게 결과를 찾아내는 것을 보여주었다. 제안된 알고리즘은 대용량 데이터 분석 등에 활용될 수 있다.

참 고 문 헌

[1] A.Z. Broder, “On the resemblance and containment of documents,” In *Proceedings of the SEQS Conference*. pp. 21-29, 1997.

[2] S. Chaudhuri, V. Ganti, and R. Kaushik. “A primitive operator for similarity joins in data cleaning.” In *ICDE*, pp.5-16, 2006.

[3] M.R. Henzinger, “Finding near-duplicate web pages: A large-scale evaluation of algorithms,” In *Proceedings of ACM SIGIR*. pp.284-291, 2006.

[4] R.J. Bayardo, Y. Ma, and R. Srikant, “Scaling up all pairs similarity search,” In *WWW'07*, pp.131-140, 2007.

[5] C. Xiao, W. Wang, X. Lin, and J.X. Yu, “Efficient Similarity Joins for Near Duplicate Detection.” In *WWW'08*, pp.131-140, 2008.

[6] L.A. Ribeiro and T. Härder, “Generalizing prefix filtering to improve set similarity joins,” *Information Systems* 36, pp.62-78, 2011.

[7] C. Xiao, W. Wang, X. Lin, J.X. Yu, and G. Wang, “Efficient Similarity Joins for Near-Duplicate Detection,” *ACM TODS*, Vol.36, No.3, Article 15, August, 2011.

박 종 수

e-mail : jpark@sungshin.ac.kr

1981년 부산대학교 전기기계공학과(학사)

1983년, 1990년 한국과학기술원 전기 및 전자공학과(석사, 박사)

1983년~1986년 국방부 군무설계기좌

1990년~현 재 성신여자대학교 IT학부 교수

관심분야 : Data mining, Database, Transportation geography