시계열 데이터베이스에서 타임 워핑 하의 서브시퀀스 매칭 : 관찰, 최적화, 성능 결과

김 만 순[†] · 김 상 욱[††]

요 약

본 논문에서는 시계열 데이터베이스에서 타임 워핑 하의 서브시퀀스 매칭을 효과적으로 처리하는 방안에 관하여 논의한다. 타임 워핑은 시퀀스의 길이가 서로 다른 경우에도 유사한 패턴을 갖는 시퀀스들을 찾을 수 있도록 해 준다. 먼저, 사전 실험을 통하여 기존의 기본적인 처리 방식인 Naive-Scan의 성능 병목이 CPU 처리 과정에 있음을 지적하고, Naive-Scan의 CPU 처리 과정을 최적화하는 새로운 기법을 제안한다. 제안된 기법은 질의 시퀀스와 서브시퀀스들간의 타임 워핑 거리들을 계산하는 과정에서 발생하는 중복 작업들을 사전에 제거함으로써 CPU 처리 성능을 극대화한다. 제안된 기법이 착오 기각을 발생시키지 않음과 Naive-Scan을 처리하기 위한 최적의 기법임을 이론적으로 증명한다. 또한, 제안된 기법을 기존의 타임 워핑 하의 서브시퀀스 매칭 기법인 LB-Scan과 ST-Filter의 후처리 단계에 적용하는 방안에 관하여 논의한다. 다양한 실험을 통한 성능 평가에 의하여 제안된 최적화 기법이 가져오는 성능 개선 효과를 정량적으로 검증한다. 실험 결과에 의하면, 기존의 타임 워핑 하의 서브시퀀스 매칭을 위한 모든 기법들이 제안된 최적화 기법에 의하여 성능이 개선되는 것으로 나타났다. 특히, Naive-Scan은 최적화 기법의 적용 전에는 가장 떨어지는 성능을 보였으나, 최적화 기법의 적용 후에는 모든 경우에서 ST-Filter나 LB-Scan을 사용한 경우보다 더 좋은 성능을 보였다. 이것은 성능 병목인 CPU 처리 과정을 최적화함으로써 기존 기법들인 Naive-Scan, LB-Scan, ST-Filter 간의 처리 성능 상의 순위 역전 현상이 발생하였음을 보이는 매우 중요한 결과이다.

Subsequence Matching Under Time Warping in Time-Series Databases : Observation, Optimization, and Performance Results

Man-Soon Kim[†] · Sang-Wook Kim[††]

ABSTRACT

This paper discusses an effective processing of subsequence matching under time warping in time-series databases. Time warping is a transformation that enables finding of sequences with similar patterns even when they are of different lengths. Through a preliminary experiment, we first point out that the performance bottleneck of Naive-Scan, a basic method for processing of subsequence matching under time warping, is on the CPU processing step. Then, we propose a novel method that optimizes the CPU processing step of Naive-Scan. The proposed method maximizes the CPU performance by eliminating all the redundant calculations occurring in computing the time warping distance between the query sequence and data subsequences. We formally prove the proposed method does not incur false dismissals and also is the optimal one for processing Naive-Scan. Also, we discuss the ways to apply the proposed method to the post-processing step of LB-Scan and ST-Filter, the previous methods for processing of subsequence matching under time warping. Then, we quantitatively verify the performance improvement effects obtained by the proposed method via extensive experiments. The result shows that the performance of all the three previous methods improves by employing the proposed method. Especially, Naive-Scan, which is known to show the worst performance, performs much better than LB-Scan as well as ST-Filter in all cases when it employs the proposed method for CPU processing. This result is so meaningful in that the performance inversion among Naive-Scan, LB-Scan, and ST-Filter has occurred by optimizing the CPU processing step, which is their performance bottleneck.

키워드 : 시계열 데이터베이스(Time-Series Databases), 서브시퀀스 매칭(Subsequence Matching), 타임 워핑(Time Warping)

1. 서 론

시계열 데이터베이스(time-series database)란 객체의 변화되는 값들의 연속으로 구성된 데이터 시퀀스(data sequence)들의 집합이다[1]. 대표적인 예로는 주가 데이터, 환율 데이터, 기온 데이터, 제품 판매량 데이터, 기업 성장률 데이터 등이 있다[2-4]. 시퀀스 매칭(sequence matching)이란 주어진 질의 시퀀스(query sequence)와 변화의 패턴이 유사한 시퀀스들을 시계열 데이터베이스로부터 찾아내는 데

※ 이 논문은 2003년도 한국학술진흥재단의 선도과학자 연구비 지원(KRF-2003-041-D00486)과 강원대학교 IT 연구 센터를 통한 연구비 지원을 받았습니다.
† 정 회 원 : 강원대학교 컴퓨터정보통신공학과
†† 종신회원 : 한양대학교 정보통신학부 교수
논문접수 : 2004년 1월 12일, 심사완료 : 2004년 9월 11일

이터 마이닝(data mining) 및 데이터 웨어하우징(data warehousing) 분야의 중요한 연산이다[1, 3-6].

시퀀스 매칭에 관한 기존의 연구에서는 길이 n의 시퀀스를 n 차원 공간상의 한 점으로 간주한다. 또한, 길이 n인 동일한 서로 다른 두 시퀀스 $X(=[x_1, x_2, \cdots, x_n])$와 $Y(=[y_1, y_2, \cdots, y_n])$간의 유사한 정도를 측정하는 척도로서 아래의 식과 같이 정의되는 거리 함수 $L_p(X, Y)$를 널리 사용한다. L_1은 맨하탄 거리(Manhattan distance), L_2는 유클리드 거리(Euclidean distance), L_∞은 대응되는 각 요소 값 쌍의 거리 중 최대 거리를 의미한다[7]. 응용에서 주어진 허용치 ε보다 작은 $L_p(X, Y)$를 갖는 임의의 두 시퀀스 X, Y를 유사하다고 간주한다[1, 4, 6, 8-11].

$$L_p(X, Y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

L_p거리 함수만을 이용한 시퀀스 매칭을 통해서는 사용자가 원하는 시퀀스들을 검색하지 못하는 경우가 빈번하게 발생한다. 따라서 응용 분야에 적합한 유사 모델(similarity model)을 적절하게 정의할 수 있도록 변환(transform)을 지원하기도 한다. 초기의 연구인 참고 문헌 [1, 4] 등에서는 변환을 지원하지 않았으나, 이후에는 스케일링(scaling)[3, 9], 시프팅(shifting)[3, 9], 정규화(normalization)[10, 12, 13], 이동 평균(moving average)[6, 14], 타임 워핑(time warping)[15-19] 등의 다양한 변환을 지원하는 방법들이 제안되었다.

이들 중 타임 워핑은 시퀀스내의 각 요소 값을 임의의 수만큼 반복시키는 것을 허용하는 변환이다[16]. 타임 워핑 후의 두 시퀀스들 간의 거리를 타임 워핑 거리(time warping distance)라 한다. 두 시퀀스 S와 Q간의 타임 워핑 거리(time warping distance) D_{tw}는 다음과 같이 재귀적으로 정의된다[16-18].

[정의 1]

(1) $D_{tw}(\langle\rangle, \langle\rangle) = 0$,

(2) $D_{tw}(S, \langle\rangle) = D_{tw}(\langle\rangle, Q) = \infty$

(3) $D_{tw}(S, Q) = (|Lp(First(S), First(Q))|^p + |\min(D_{tw}(S, Rest(Q)), D_{tw}(Rest(S), Q), D_{tw}(Rest(S), Rest(Q)))|^p)^{1/p}$

여기서, First(S)는 각각 S의 첫 번째 요소 s_1을 의미하며, Rest(S)는 s_1을 제외한 S의 나머지 요소들로 구성되는 시퀀스를 의미한다. $\langle\rangle$은 요소가 존재하지 않는 널 시퀀스(null sequence)를 의미한다. min은 세 개의 인자들 중 가장 작은 값을 가지는 것을 취하는 함수이다. L_p는 응용에서 적합한 것을 선택하여 사용할 수 있다.

□

본 논문에서는 현재 가장 널리 사용되는 맨해튼 거리(Manhattan distance) L_1을 기반으로 하는 타임 워핑 거리에 연구의 초점을 맞추고자 한다. 두 시퀀스들을 대상으로 하는 타임 워핑은 변환 후의 두 시퀀스들 간의 타임 워핑 거리를 최소화하는 방향으로 진행된다. 예를 들어, 두 시퀀스 S = <20, 21, 21, 20, 20, 23, 23, 23> 와 Q = <20, 20, 21, 20, 23>를 타임 워핑에 의하여 동일한 시퀀스 <20, 20, 21, 21, 20, 20, 23, 23, 23>으로 변환될 수 있으며, 이 결과 $D_{tw}(S, Q)$는 0이 된다.

전술한 바와 같이, L_p거리는 두 시퀀스의 길이가 동일한 경우에만 적용할 수 있다. 반면, 타임 워핑 거리는 데이터베이스내의 시퀀스들의 길이가 서로 달라서 L_p거리 함수를 이용하여 유사 정도를 직접 측정할 수 없는 경우에 매우 유용하다[18][1]. 현재, 타임 워핑은 음성 인식 분야에서 널리 사용되고 있으며[20], 심전도 데이터, 주가 데이터, 기온 데이터, 기업 성장률 데이터 등에도 유사한 방식으로 적용할 수 있다.

최근, 타임 워핑 하의 시퀀스 매칭에 관한 다양한 연구가 수행되어 왔다[15-19]. 타임 워핑 하의 시퀀스 매칭의 처리를 위한 기존의 기법들은 크게 여과 단계(filtering step)와 후처리 단계(post processing step)로 구성된다. 여과 단계는 주어진 질의 시퀀스와 전혀 유사하지 않는 시퀀스들을 미리 제거함으로써 최종 결과에 포함될 가능성이 매우 높은 시퀀스들로만 구성되는 후보 집합(candidate set)을 구성하는 단계이다. 질의 시퀀스와 실제로 유사한 시퀀스를 필터링 단계에서 후보 집합 내에 포함시키지 못하는 현상을 착오 기각(false dismissal)이라 한다. 반면, 질의 시퀀스와 유사하지 않은 일부의 시퀀스를 필터링 단계에서 후보 집합 내에 포함시키는 현상을 착오 채택(false alarm)이라 한다. 후처리 단계는 후보 집합에 속하는 각 시퀀스를 디스크로부터 액세스하여 이것이 질의 시퀀스와 실제로 유사한가의 여부를 판단함으로써 착오 채택을 제거하는 단계이다.

참고 문헌 [15]에서는 여과 단계 없이 모든 시퀀스들 각각에 대하여 질의 시퀀스와의 타임 워핑 거리를 동적 프로그래밍(dynamic programming)을 이용하여 계산함으로써 타임 워핑 하의 시퀀스 매칭을 처리하는 기법을 제안하였다. 본 논문에서는 참고 문헌 [18]의 명칭을 따라 이 기법을 Naive-Scan이라 부른다. 여과 단계 없이 모든 시퀀스들을 후보로 고려하는 Naive-Scan은 그 처리 성능이 매우 떨어지므로, 참고 문헌 [16]과 [17]에서는 여과 단계를 채택하는

1) 다음과 같은 경우, 서로 다른 길이를 가지는 시퀀스들 간의 유사 정도의 측정이 요구된다[18]. 첫째, 두 시퀀스들을 위한 요소 값의 측정 주기가 다른 경우이다. 예를 들어, 한 시퀀스는 매 분마다 요소 값을 측정하고, 다른 시퀀스는 매 시간마다 요소 값을 측정할 수 있다. 둘째, 측정 주기는 같지만, 측정 시작 시점이 다른 경우이다. 예를 들어, 한 시퀀스는 측정이 1년 전부터 시작되었지만, 다른 시퀀스는 새로 데이터베이스에 추가되어 오늘부터 측정이 시작될 수 있다. 이와 같이 비교하고자 하는 시퀀스들의 길이가 서로 다른 경우, 타임 워핑은 시퀀스들의 개별적인 요소 값의 차이보다는 시간의 변화에 따르는 시퀀스들의 전체적인 경향이 얼마나 유사한가를 파악하는데 유용하게 사용되는 변환이다.

기법들을 제안하였다. 본 논문에서는 참고 문헌 [18]의 명칭을 따라 이 기법들을 각각 LB-Scan과 ST-Filter라 부른다. LB-Scan은 별도의 자료 구조 없이 모든 시퀀스들을 대상으로 여과 단계를 신속하게 수행하고, 이 결과 반환되는 후보 시퀀스들만을 대상으로 후처리 단계를 수행한다. 반면, ST-Filter는 접미어 트리(suffix tree)라는 별도의 자료 구조를 이용하여 여과 단계를 수행하고, 이 결과 반환되는 후보 시퀀스들만을 대상으로 후처리 단계를 수행한다. LB-Scan과 ST-Filter 모두 후처리 단계에서는 Naive-Scan에서와 같이 동적 프로그래밍을 이용하여 타임 워핑 거리를 계산한다[2].

시퀀스 매칭은 다음과 같이 전체 매칭(whole matching)과 서브시퀀스 매칭(subsequence matching)으로 구분된다[4].

- 전체 매칭 : 시퀀스 $S_1, S_2, \cdots, S_N$을 포함하는 데이터베이스 D로부터 질의 시퀀스 Q와 유사한 시퀀스 S_i를 검색한다.
- 서브시퀀스 매칭 : 시퀀스 $S_1, S_2, \cdots, S_N$을 포함하는 데이터베이스 D로부터 질의 시퀀스 Q와 유사한 서브시퀀스 $S_i[j:k]$를 포함하는 시퀀스 S_i와 j, k 값을 검색한다. 여기서 j와 k는 시퀀스 S_i내에서 질의 시퀀스와 유사한 서브시퀀스가 시작하는 위치와 끝나는 위치를 의미한다.

서브시퀀스 매칭은 전체 매칭을 일반화한 것이므로[4, 9, 16, 21, 22] 실제 응용 분야에서 널리 사용된다. 따라서 본 논문에서는 보다 실용적인 타임 워핑 하의 효과적인 **서브시퀀스 매칭 문제**를 다루고자 한다. 기존에 제안된 LB-Scan 및 ST-Filter를 이용하여 실제 상황에서 사용 가능한 정도의 서브시퀀스 매칭 성능을 얻을 수 있으나, 실제 데이터베이스 환경에서 이러한 기법들을 이용한 서브시퀀스 매칭은 아직도 수초에서 수십 초의 매우 긴 처리 시간을 요구한다[17]. 따라서 보다 빠른 처리를 위한 추가의 성능 개선 방안이 요구된다.

Naive-Scan의 기본 전략은 각 시퀀스를 디스크로부터 액세스 한 후, 그 시퀀스 내의 각 서브시퀀스에 대하여 동적 프로그래밍을 이용하여 질의 시퀀스와의 타임 워핑 거리를 계산하는 것이다. 이러한 Naive-Scan의 기본 전략은 기존의 모든 타임 워핑 하의 서브시퀀스 매칭 기법들의 후처리 단계에서 공통적으로 사용된다. 따라서 Naive-Scan의 처리 과정의 최적화는 기존의 여과 단계를 포함하는 기법들의 수행 시간을 단축시키는 효과를 가지므로 타임 워핑 하의 서브시퀀스 매칭 연구에서 매우 중요한 의미를 가진다. 본 논문에서는 이 점을 착안하여 Naive-Scan을 최적화하기 위한 매우 효과적인 기법을 제안한다. 본 논문의 주요 공헌을 간략히 요약하면 다음과 같다.

- Naive-Scan의 전체 처리 시간의 대부분을 CPU 처리 시간이 차지함을 보이고, CPU 처리 과정의 최적화가 Naive-Scan의 전체 성능을 개선할 수 있는 매우 중요한 이슈임을 지적한다.
- Naive-Scan의 CPU 처리 과정을 최적화하기 위한 새로운 기법을 제안한다. 제안된 기법은 질의 시퀀스와 서브시퀀스들간의 타임 워핑 거리들을 계산하는 과정에서 발생하는 많은 중복 작업 및 불필요한 작업을 사전에 제거함으로써 CPU 처리 성능을 극대화할 수 있다.
- 제안된 기법이 착오 기각을 발생시키지 않음과 Naive-Scan을 처리하기 위한 최적의 기법임을 이론적으로 증명한다.
- 제안된 기법을 기존의 각 타임 워핑 하의 서브시퀀스 매칭 기법의 후처리 단계에 적용하는 방안을 제시한다.
- 다양한 실험들을 통한 성능 평가에 의하여 제안된 기법이 기존의 기법들에 미치는 성능 개선 효과를 정량적으로 검증한다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련 연구로서 타임 워핑 하의 시퀀스 매칭을 위한 기존의 기법들을 소개하고, 장단점을 논의한다. 제3장에서는 사전 실험을 통하여 Naive-Scan의 전체 처리 시간 중 CPU 처리 시간이 차지하는 비중이 대부분임을 보인다. 제4장에서는 기존의 후처리 단계 처리 방식에서 발생하는 많은 CPU 중복 계산의 문제점을 지적하고, 이러한 문제점을 해결하는 새로운 기법을 제안한다. 제5장에서는 제안된 기법의 우수성을 다양한 실험들을 기반으로 하는 성능 평가를 통하여 검증한다. 끝으로, 제6장에서는 본 논문을 요약하고, 결론을 내린다.

2. 관련 연구

본 장에서는 관련 연구로서 타임 워핑 하의 시퀀스 매칭을 착오 기각 없이 수행하는 기존의 연구들로서 Naive-Scan, LB-Scan, ST-Filter를 소개한다. 각 기법에 관하여 (1) 전체 시퀀스 매칭 방안, (2) 서브시퀀스 매칭 방안[3], (3) 주요 특징에 관하여 논의한다.

2) 이러한 세 가지 기법들 이외에도 참고 문헌 [16]에서 제안한 FastMap 기반 기법, 참고 문헌 [18, 19]에서 제안한 인덱스 기반 기법(index-based approach) 등이 있다. FastMap 기반 기법은 다차원 인덱스(multidimensional index)를 이용한 여과 단계를 통하여 빠른 처리 성능을 제공하나, 착오 기각을 유발시키는 문제점이 있다. 또한, 인덱스 기반 기법은 착오 기각을 유발시키지 않으면서도 다차원 인덱스를 이용한 여과 단계를 수행하는 좋은 기법이나, 타임 워핑 거리 계산을 위하여 L1을 기본 거리 함수로 사용하는 위의 세 가지 기법들과는 달리, L_∞을 기본 거리 함수로 사용한다[18, 19]. 즉, FastMap 기반 기법과 인덱스 기반 기법은 위의 세 가지 기법들과는 적용 응용 분야 측면에서 차이가 있다. 본 논문에서는 이러한 이유로 인하여 이 두 가지 기법들을 이후의 논의의 대상에서 제외한다.

3) ST-Filter는 원래 서브시퀀스 매칭을 대상으로 제안된 기법인 반면, Naive-Scan과 LB-Scan은 전체 매칭을 대상으로 제안된 기법이다. 본 장에서는 Naive-Scan과 LB-Scan을 기본 아이디어를 이용하여 서브시퀀스 매칭을 수행하는 일반적인 방안을 소개한다.

2.1 Naive-Scan[15]

전체 매칭 방안 : 디스크로부터 각 데이터 시퀀스를 액세스한 후, 이 데이터 시퀀스 S와 질의 시퀀스 Q의 타임 워핑 거리 $D_{tw}(S, Q)$를 계산함으로써 전체 매칭을 수행한다. $D_{tw}(S, Q)$를 효과적으로 계산하기 위한 방법으로서 동적 프로그래밍(dynamic programming)을 사용한다[15]. 계산된 $D_{tw}(S, Q)$값이 허용치 ε이하인 경우, 해당 시퀀스 S가 질의 시퀀스 Q와 유사하다고 간주한다.

동적 프로그래밍을 사용하여 S와 Q간의 타임 워핑 거리를 계산할 때, 거리 축적 테이블(cumulative distance table) T의 각 요소 T(i, j)는 다음과 같은 재귀 관계(recurrence relation)에 의하여 구성된다[15]. 동적 프로그래밍 알고리즘은 아래의 재귀 관계를 이용하여 거리 축적 테이블 T를 아래에서 위로 채워나간다.

$$T(0, 0) = 0$$
$$T(0, j) = T(i, 0) = \infty$$
$$T(i, j) = |Q[i] - S[j]| + \min(T(i-1, j), T(i, j-1), T(i-1, j-1))$$

다음의 (그림 1)은 기본 거리 함수로서 L_1이 사용되는 경우, 거리 축적 테이블을 이용한 두 시퀀스 S와 Q의 타임 워핑 거리 계산 예를 보인다. 계산 결과, $D_{tw}(S, Q)$는 12가 된다.

6	16	11	12
6	13	9	10
7	10	7	8
6	6	4	5
5	3	2	3
4	1	1	2
S / Q	3	4	3

(그림 1) L_1을 이용한 S = ⟨4, 5, 6, 7, 6, 6⟩과 Q = ⟨3, 4, 3⟩의 타임 워핑 거리 계산의 예

서브시퀀스 매칭 방안 : 각 데이터 시퀀스 S를 디스크로부터 액세스한 후, S에 속하는 각 서브시퀀스 S[i : j]에 대하여 질의 시퀀스 Q와의 타임 워핑 거리 $D_{tw}(S[i : j], Q)$를 동적 프로그래밍을 이용하여 계산함으로써 서브시퀀스 매칭을 수행한다.

특징 : 여과 단계를 거치지 않으므로 후처리 단계의 수행 시간이 지나치게 크다[18]. 즉, 모든 데이터 시퀀스들을 디스크로부터 액세스해야 한다는 부담이 있다. 또한, (서브)시퀀스 S와 Q의 D_{tw}를 계산할 때의 CPU 수행 시간은 $O(|S| * |Q|)$이므로 매우 크다. 여기서, |S|와 |Q|는 각각 시퀀스 S와 Q의 크기를 의미한다. 이 결과, 많은 시퀀스들로 구성되는 대형 데이터베이스 환경에서는 검색 성능이 떨어진다.

2.2 LB-Scan[16]

전체 매칭 방안 : 타임 워핑 거리 D_{tw}의 반환 값보다 항상 작은 값을 반환하는 하한 함수(lower-bound function) D_{lb}를 이용하여 여과 단계를 수행한다. 즉, 여과 단계에서는 디스크로부터 각 데이터 시퀀스를 액세스한 후, 이 데이터 시퀀스 S와 질의 시퀀스 Q에 대하여 $D_{lb}(S, Q)$를 적용한다. 여과 단계에서 D_{lb}의 반환 값이 허용치 ε이하인 데이터 시퀀스 S′에 대해서는 질의 시퀀스 Q와의 타임 워핑 거리 $D_{tw}(S', Q)$를 계산하는 후처리 단계를 수행한다. $D_{tw}(S', Q)$의 계산을 위하여 Naive-Scan과 동일한 방식으로 동적 프로그래밍을 사용한다.

서브시퀀스 매칭 방안 : 여과 단계에서 디스크로부터 각 데이터 시퀀스 S를 액세스한 후, S에 속하는 각 서브시퀀스 S[i : j]와 질의 시퀀스 Q에 대하여 $D_{lb}(S[i : j], Q)$를 적용한다. 여기서에서 D_{lb}의 반환 값이 허용치 ε이하인 서브시퀀스 S[i : j]′에 대해서는 동적 프로그래밍을 이용하여 질의 시퀀스 Q와의 타임 워핑 거리 $D_{tw}(S[i : j]', Q)$를 계산하는 후처리 단계를 수행한다.

특징 : 별도의 자료 구조를 채택하지 않으므로 여과 단계에서 모든 데이터 시퀀스들이 디스크로부터 액세스된다. 따라서 디스크 액세스 시간은 Naive-Scan과 동일하다. 여과 단계에서는 모든 (서브)시퀀스 S와 질의 시퀀스 Q 간의 D_{lb}가 계산된다. 각 $D_{lb}(S, Q)$를 계산할 때의 CPU 수행 시간은 $O(|S|+|Q|)$로서 $O(|S|*|Q|)$인 $D_{tw}(S, Q)$와 비교하여 CPU 수행 시간이 매우 작다[7]. 여과 단계를 통하여 최종 결과에 포함될 가능성이 없는 (서브)시퀀스들을 사전에 제외시킬 수 있으므로 후처리 단계의 수행 시간을 크게 줄일 수 있다[18]. 따라서 여과 단계에서 제외되는 (서브)시퀀스들이 많은 경우, 성능 개선 효과는 매우 크다.

2.3 ST-Filter[17]

전체 매칭 방안 : 여과 단계를 위하여 데이터베이스 내의 각 시퀀스의 요소 값들을 심볼로 변환시키고, 이들을 접미어 트리(suffix tree)[24] 내에 저장시킨다. 여과 단계에서는 접미어 트리 검색을 이용하여 질의 시퀀스 Q와의 타임 워핑 거리 D_{tw}가 허용치 ε이하일 가능성이 있는 후보 시퀀스 S′들을 걸러낸다. 후처리 단계에서는 이러한 각 S′을 대상으로 동적 프로그래밍을 사용하여 $D_{tw}(S', Q)$를 계산한다.

서브시퀀스 매칭 방안 : 여과 단계를 위하여 각 데이터 시퀀스내 각 접미어의 요소 값들을 심볼로 변환시키고, 이들을 접미어 트리 내에 저장시킨다. 여과 단계에서는 접미어 트리 검색을 통하여 질의 시퀀스 Q와의 타임 워핑 거리 D_{tw}가 허용치 ε이하일 가능성이 있는 후보 서브시퀀스 S[i : j]′들을 걸러낸다. 후처리 단계에서는 이러한 각 S[i : j]′을 대상으로 $D_{tw}(S[i : j]', Q)$를 계산한다.

특징 : 접미어 트리 검색을 사용하므로 LB-Scan과는 달리 전체 데이터 시퀀스들이 아닌 접미어 트리의 일부만을 디스크로부터 액세스함으로써 여과 단계를 수행할 수 있다. 그러나 이 접미어 트리의 크기는 데이터 시퀀스들이 저장된 파일보다 큰 것이 일반적이다. 또한, 좋은 시퀀스 매칭 성능을 제공하기 위한 최적의 도메인 분류(categorization)[17]가 쉽지 않으며[18], 동일한 데이터베이스의 경우에도 질의 시퀀스마다 서브시퀀스 처리 성능에 큰 차이가 있다.

3. 사전 실험 및 결과 분석

Naive-Scan의 기본 전략은 각 시퀀스를 디스크로부터 액세스 한 후, 그 시퀀스 내의 각 서브시퀀스에 대하여 동적 프로그래밍을 이용하여 질의 시퀀스와의 타임 워핑 거리를 계산하는 것이다. 이러한 Naive-Scan의 기본 전략은 기존의 모든 타임 워핑 하의 서브시퀀스 매칭 기법들의 후처리 단계에서 공통적으로 사용된다. 본 장에서는 Naive-Scan을 대상으로 하는 사전 실험 결과를 제시함으로써 본 논문에서 최적화 하고자 하는 대상을 제시한다.

본 논문에서는 Naive-Scan을 이용한 서브시퀀스 매칭의 처리 과정에서 요구되는 시간적 요소를 Time(Disk)와 Time(CPU)로 구분한다. Time(Disk)는 해당 시퀀스를 디스크로부터 액세스하는 데 소요되는 시간을 의미하며, Time(CPU)는 동적 프로그래밍을 이용하여 질의 시퀀스와 각 서브시퀀스의 타임 워핑 거리를 계산하는데 소요되는 시간을 의미한다. 이와 같이, 전체 수행 시간을 두 가지 요소로 구분하는 이유는 최적화 할 대상을 정확하게 파악하고, 이를 해결하기 위한 전략을 올바르게 수립하기 위해서이다. 본 연구에서는 먼저 전체 Naive-Scan의 수행 시간 중 이 두 가지 요소가 차지하는 비중을 파악하기 위하여 다음과 같은 사전 실험을 수행하였다.

실험 환경으로 사용된 하드웨어 플랫폼은 1.28 GB 크기의 주기억장치와 1.7GHz Pentium Ⅳ CPU를 탑재한 PC이며, 소프트웨어 플랫폼은 Linux Kernel Version 2.4.18이다. 실험에서 사용된 시계열 데이터는 길이가 512인 620개의 한국의 실제 주식 시퀀스들이다. 사용된 질의 시퀀스의 길이는 300이며, 허용치 ε은 2개에서 7개까지의 최종 질의 결과가 나오도록 설정하였다. <표 1>은 이러한 사전 실험 결과를 나타낸 것이다.

〈표 1〉 Naive-Scan을 이용한 타임 워핑 하의 서브시퀀스 매칭의 수행 시간 분석

	Total-Time	Time(CPU)	Time(DISK)
소요 시간 (msec)	62,515	62,292	223

위 <표 1>에서 나타난 결과를 보면 전체 수행 시간 62,515msec 중 Time(CPU)와 Time(DISK)는 각각 62,292msec과 223msec로 나타났다. 즉, 전체 Naive-Scan의 수행 시간 중에서 CPU 처리 시간의 비중은 99% 이상인 것이다. 이것은 CPU 처리 과정을 최적화함으로써 전체 Naive-Scan의 수행 시간을 크게 개선할 수 있음을 의미하는 것이다. 또한, 이러한 Naive-Scan에서 사용하는 CPU 처리 과정은 기존의 모든 타임 워핑 하의 서브시퀀스 매칭 기법들의 후처리 단계에서 공통적으로 채택되고 있다. 따라서 Naive-Scan의 CPU 처리 과정의 최적화는 기존의 여과 단계를 포함하는 기법들의 수행 시간을 단축시키는 효과를 가져올 수 있으므로 타임 워핑 하의 서브시퀀스 매칭 연구에서 매우 중요한 의미를 가진다.

4. 성능 최적화 방안

본 장에서는 타임 워핑 하의 서브시퀀스 매칭의 CPU 처리 과정에 대한 최적화 방안에 관하여 논의한다. 먼저 제 4.1절에서는 제안하는 기법이 기반을 두고 있는 이론적 배경을 제시하고, 제 4.2절에서는 성능 최적화를 위한 새로운 기법을 제안한다. 제 4.3절에서는 제안된 기법의 견고성 및 효율성에 대하여 논의하고, 제 4.4절에서는 제안된 기법을 여과 단계를 포함하는 기존의 여러 기법에 적용하는 방안을 제시한다.

4.1 이론적 배경

Naive-Scan CPU 처리 과정의 최적화를 위하여 본 논문에서 제안하는 기법은 다음과 같은 이론적 배경을 기반으로 하고 있다.

[정리 1]

두 시퀀스 S와 Q에 대하여, S내의 임의의 접두어(prefix) S[1 : j]와 Q와의 타임 워핑 거리 D_{tw}(S[1 : j], Q)는 D_{tw}(S, Q)를 계산하기 위한 과정의 부산물(by-product)로서 함께 계산된다.

[증명]

동적 프로그래밍을 이용하여 S와 Q간의 거리 축적 테이블((그림 1) 참조)을 구성할 때, 마지막 요소인 T(|S|,|Q|)를 계산하기 전에 T(j,|Q|)가 먼저 계산됨을 보임으로써 위의 정리를 증명한다.

n (= |S| - j)에 대한 수학적 귀납법을 이용하여 증명한다.

case 1 : n = 1(j = |S| - 1)

T(|S|,|Q|)=|Q[|Q|]-S[|S|]|+min(T(|S|-1,|Q|), T(|S|-1,|Q|-1), T(|S|,|Q|-1))

즉, T(|S|,|Q|)를 계산하는 행의 직전 행에서 T(|S|-1,|Q|)가 계산된다. 따라서 n = 1일 때 성립한다.

case 2 : n = k(j = |S| - k)

T(|S|,|Q|)를 계산하는 행의 이전의 행에서 T(|S|-k, |Q|)가

z계산된다고 가정하자.

case 3 : n = k + 1(j = |S| - (k+1))

case 2에 의하여 T(|S|, |Q|)를 계산하는 행의 이전의 행에서 T(|S|-k, |Q|)가 계산된다고 가정하였다. 또한, case 1에 의하여 T(|S|-k, |Q|)를 계산하는 행의 직전의 행에서 T (|S|-(k+1), |Q|)가 계산됨을 보였다. 따라서 T(|S|, |Q|)를 계산하는 행의 이전의 행에서 T(|S|-(k+1), |Q|)가 계산된다. 그러므로 n = k + 1일 때 성립한다.

그러므로 수학적 귀납법에 의하여 위의 [정리 1]은 성립한다. □

[정리 2][4]

$\forall j\ (1 \le j \le |Q|)\ T(i, j) > \varepsilon \Rightarrow \forall k\ (k > i)\ T(k, |Q|) > \varepsilon$

[증명]

먼저, 동적 프로그래밍을 이용하여 S와 Q간의 거리 축적 테이블을 구성할 때, i 번째 행의 모든 요소 T(i, j)가 ε보다 큰 값을 갖는 경우, i+1 번째 모든 요소 T(i+1, j)가 ε보다 큰 값을 가짐을 보인다.

j에 대하여 수학적 귀납법을 이용하여 증명한다.

case 1 : j = 1

$T(i+1, 1) = |Q[1]-S[i+1]| + \min(T(i, 1), T(i, 0), T(i+1, 0))$
$= |Q[1]-S[i+1]| + \min(T(i, 1), \infty, \infty) > \varepsilon$

따라서 j = 1일 때 성립한다.

case 2 : j = k

T(i+1, k) > ε가 성립한다고 하자.

case 3 : j = k+1

i 번째 행의 모든 요소는 ε보다 큰 값을 가지므로 T(i, k+1)와 T(i, k)는 모두 ε보다 큰 값을 갖는다. 또한, case 2에 의하여 T(i+1, k)는 ε보다 큰 값을 갖는다. 따라서

$T(i+1, k+1) = |Q[k+1]-S[i+1]| + \min(T(i, k+1), T(i, k), T(i+1, k)) > \varepsilon$

따라서 j = k+1일 때 성립한다.

이와 같이, 수학적 귀납법에 의하여 i 번째 행의 모든 요소 T(i, j)가 ε보다 큰 값을 갖는 경우, i+1 번째 행의 모든 요소 T(i+1, j)들은 ε보다 큰 값을 갖는다. 또한, 이렇게 i+1 번째 행의 모든 요소 T(i, j)가 ε보다 큰 값을 가지므로, i+2 번째 행의 모든 요소 T(i+2, j)들도 ε보다 큰 값을 갖는다. 동일한 방식의 적용을 통하여 i 번째 행의 모든 요소 T(i, j)가 ε보다 큰 값을 갖는 경우, k (> i) 번째 행의 모든 요소 T(k, j)들은 ε보다 큰 값을 가지게 되므로 위의 정리는 성립된다.

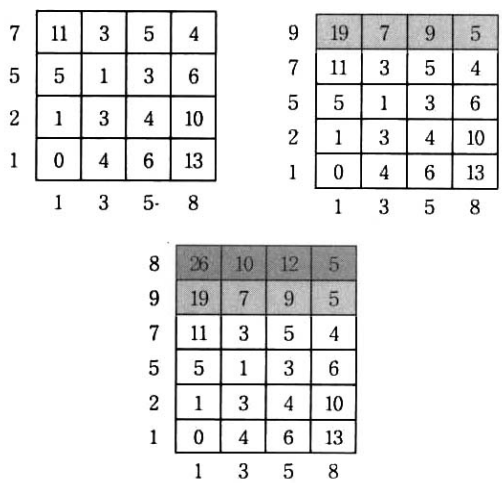
4) 참고 문헌 [17]에서도 이와 유사한 정리를 제시하고, 이러한 특성을 이용한 처리 방안을 ST-Filter에 적용한 바 있다. 그러나 Naive-Scan에는 이러한 방식을 적용하지 않았다[23].

4.2 제안하는 기법

본 절에서는 제 4.1절의 이론적인 배경을 기반으로 Naive-Scan을 이용한 타임 워핑 하의 서브시퀀스 매칭에서 CPU 처리 과정을 최적화하는 기법을 제안한다.

4.2.1 기본 전략

기존의 Naive-Scan의 방식에서는 시퀀스 S의 각 서브시퀀스 S[i : j] (1≤i≤|S|, i≤j≤|S|)에 대하여 각각 서로 다른 거리 축적 테이블을 구성한다. 시퀀스 S = <1, 2, 5, 7, 9, 8>와 질의 시퀀스 Q = <1, 3, 5, 8>를 고려해 보자. (그림 2)은 질의 시퀀스 Q와 시퀀스 S의 서브시퀀스 S[1 : 4], S[1 : 5], S[1 : 6]의 각 거리 축적 테이블을 구성하는 과정을 나타낸 것이다. 그림에 나타난 바와 같이 세 거리 축적 테이블들 간에는 많은 중복이 존재한다. 서브시퀀스 S[1 : 4]에 대한 거리 축적 테이블은 서브시퀀스 S[1 : 5] 및 S[1 : 6]을 위한 거리 축적 테이블의 1~4행에 완전히 포함되며, 서브시퀀스 S[1 : 5]를 위한 거리 축적 테이블 역시 서브시퀀스 S[1 : 6]을 위한 거리 축적 테이블에 완전히 포함된다. 이와 같이, 기존의 Naive-Scan의 방식대로 모든 가능한 서브시퀀스들에 대하여 거리 축적 테이블을 개별적으로 구성하는 경우, 매우 많은 계산 과정의 중복이 발생하게 된다.



(그림 2) 동일한 접미어의 접두어인 세 서브시퀀스들에 대한 거리 축적 테이블

본 연구에서는 이러한 계산 과정의 중복을 제거하기 위한 방법으로서 시퀀스 S의 모든 서브시퀀스가 아닌 모든 접미어 S[i : |S|]에 대해서만 거리 축적 테이블을 구성하는 방식을 제안한다. 시퀀스 S내의 접미어가 아닌 서브시퀀스 S[i : j] (1≤i<|S|, i≤j<|S|)는 S의 접미어(suffix)인 S[i : |S|]의 접두어(prefix)이다. 정리 1에 의하여 접미어가 아닌 서브시퀀스 S[i : j]와 질의 시퀀스 Q와의 타임 워핑 거리 D_{tw}(S[i : j], Q)는 D_{tw}(S[i : |S|], Q)의 계산하기 위한 과정의 부산

물로서 함께 계산된다. 따라서 시퀀스 S의 각 접미어 S[i : |S|]에 대하여 하나의 거리 축적 테이블을 구성함으로써 시퀀스 S에 포함되는 (|S|-i)개의 접미어가 아닌 서브시퀀스들에 대한 질의 시퀀스 Q와의 타임 워핑 거리 $D_{tw}(S[i:j], Q)$를 모두 얻을 수 있으며, 이 결과 기존 방식에서 발생하던 (|S|-i)회의 테이블 구성 시간을 모두 제거할 수 있다. 본 논문에서는 이러한 전략을 ***테이블 공유 전략(table-sharing strategy)***이라 정의한다.

(그림 2)의 예에 테이블 공유 전략을 사용하게 되면, S의 접미어의 하나인 S[1 : 6]에 대한 거리 축적 테이블만을 구성함으로써 이의 접두어에 해당되는 서브시퀀스 S[1 : 1], S[1 : 2], S[1 : 3], S[1 : 4], S[1 : 5], S[1 : 6]의 질의 시퀀스와의 타임 워핑 거리를 모두 구하게 된다.

본 연구에서는 CPU 수행 시간의 최적화를 위하여 정리 2를 이용한 또 하나의 전략을 제안한다. 테이블 공유 전략에 의한 시퀀스 S의 접미어 S[i : |S|]에 대한 거리 축적 테이블 구성하는 것은 S[i : |S|]의 접두어인 (|S|-i+1)개의 서브시퀀스들 S[i : j]에 대한 질의 시퀀스 Q와의 타임 워핑 거리들을 j가 증가하는 순서로 구하는 과정을 의미한다. 정리 2는 이 과정에서 접미어 S[i : |S|]에 대한 거리 축적 테이블을 j가 |S|로 될 때까지 완전히 구성할 필요가 없음을 알려주는 것이다. 즉, 테이블 구성 과정에서 j 열에 존재하는 모든 값이 질의에서 주어진 ε보다 크다면 더 이상의 진행은 무의미하다는 것이다. 따라서 더 이상의 테이블 구성의 진행을 중단하고, 길이가 더 긴 서브시퀀스들을 최종 결과에서 안전하게 제외시킬 수 있다. 본 논문에서는 이 전략을 ***테이블 커팅 전략(table-cutting strategy)***이라 정의한다.

(그림 2)에서 가장 오른쪽 그림을 S의 접미어 S[1 : 6]에 대한 거리 축적 테이블만을 구성하는 과정이라 가정하자. 질의에서 주어진 ε이 2라 하면, S[1 : 6]의 접두어에 해당되는 네 번째 서브시퀀스 S[1 : 4]와 대응되는 행 내의 모든 값이 2보다 크므로 테이블 구성은 중단된다. 따라서 이후의 행과 대응되는 서브시퀀스들에 대한 타임 워핑 거리 계산 과정을 제거시킬 수 있다.

4.2.2 알고리즘

(알고리즘 1)은 테이블 공유 전략과 테이블 커팅 전략을 적용한 질의 시퀀스 Q와 데이터 시퀀스 S간의 서브시퀀스 매칭 알고리즘을 나타낸 것이다. 원래의 Naive-Scan에서의 서브시퀀스 매칭에서와는 달리 S의 각 접미어의 접두어에 해당하는 다수의 서브시퀀스들에 대해서는 거리 축적 테이블 공유한다(라인 1). 또한, 하나의 접미어에 대한 거리 축적 테이블의 구성 과정에서 하나의 열에 해당되는 요소 값들이 모두 ε보다 크게 되는 경우 더 이상의 테이블 구성 진행을 중단하고, 다음 접미어에 대한 테이블 구성을 시작한다(라인 5).

```
1 FOR each suffix S[i : |S|] (1≤i≤|S|) of data sequence S
2     FOR each j (1≤j≤|S|-i)
3         fill the j-th row of the cumulative distance table,
          T(j, k) (1≤k≤|Q|), by dynamic programming
4         IF T(j,|Q|) is smaller than ε, THEN add S[i : j]
          into the answer-set
5         ELSE IF every T(j, k) (1≤k≤|Q|) is larger than
          ε, THEN get out of the inner loop
```

(알고리즘 1) 최적화된 서브시퀀스 매칭 알고리즘

4.2.3 논의 사항

먼저, [정리 3]을 통하여 제안된 기법의 견고성(robustness)을 제시한다.

[정리 3]

제안된 기법을 이용하여 타임 워핑 하의 서브시퀀스 매칭을 수행하는 경우 착오 기각이 발생하지 않는다.

[증명]

시퀀스 S내의 임의의 서브시퀀스 S[i : j]는 S의 접미어 S[i : |S|]의 한 접두어와 일대일 대응된다. 제안된 기법에서는 S의 모든 접미어들에 대하여 거리 축적 테이블을 구성하므로 [정리 1]에 의하여 그 과정에서 모든 서브시퀀스들의 질의 시퀀스와의 타임 워핑 거리를 빠짐 없이 계산하게 된다. (알고리즘 1)의 라인 5에 의하여, 일부 서브시퀀스들을 최종 결과 대상에서 미리 제외시키지만, [정리 2]에 의하여 이러한 서브시퀀스들은 최종 결과에 들어갈 수 없는 것이다. 요약하면, 제안된 기법은 모든 서브시퀀스들을 대상으로 질의 시퀀스와의 타임 워핑 거리를 계산하는 기존의 Naive-Scan 방식과 항상 동일한 최종 결과를 구하게 된다. 따라서 제안된 기법에서 착오 기각은 발생하지 않는다.

다음은 제안된 기법의 효율성을 규명한다. 기존의 Naive-Scan 방식에서는 모든 서브시퀀스들을 위한 별도의 거리 축적 테이블을 모두 구성한다. 시퀀스 S내에는 모두 $O(|S|^2)$개의 서브시퀀스들이 존재한다. 각 서브시퀀스 S[i : j]와 질의 시퀀스 Q간의 거리 축적 테이블을 구하는 시간은 O(|Q|*|S[i : j]|)이다. 따라서 질의 시퀀스 Q와 시퀀스 S에 대한 서브시퀀스 매칭을 수행하기 위한 CPU 처리 시간은 $O(|Q|*|S|^3)$이다.

반면, 제안된 기법에서는 테이블 공유 전략에 의하여 시퀀스 S내의 접미어들에 대해서만 거리 축적 테이블을 구성하게 된다. 시퀀스 S내에는 O(|S|)개의 접미어들이 존재하므로, 제안된 기법으로 질의 시퀀스 Q와 시퀀스 S에 대한 서브시퀀스 매칭을 수행하기 위한 CPU 처리 시간은 $O(|Q|*|S|^2)$가 된다. 따라서 기존의 타임 워핑 하의 서브시퀀스 매칭의 수행 시간을 크게 줄일 수 있다. 뿐만 아니라, 테이블 커팅 전략에 의하여 더 이상의 진행이 무의미한 경우에는 거리 축적 테이블을 구성을 도중에 중단하게 되므로 수행 시간은 더욱 줄어든다.

4.3 여과 단계를 포함하는 기법들과의 결합 방안

여과 단계를 포함하는 기존의 LB-Scan과 ST-Filter에서도 후처리 단계에서 Naive-Scan에서의 서브시퀀스 매칭 방식을 그대로 사용해 왔다. 본 절에서는 제안된 기법을 여과 단계를 포함하는 기존의 기법들과의 결합하는 방안에 관하여 논의한다. 또한, 제 5장에서는 제안된 기법을 기존 기법에 결합함으로써 얻어지는 성능 개선 효과에 대한 실험 결과를 제시한다.

4.3.1 LB-Scan과의 결합

LB-Scan에서는 각 시퀀스 별로 여과 단계가 수행되며, 여과 단계에서 하한 함수를 통하여 후보로 지명된 서브시퀀스들을 대상으로 해당 시퀀스를 위한 추가의 디스크 액세스 없이 바로 후처리 단계에 들어가게 된다. 후처리 단계에서는 각 후보 서브시퀀스와 질의 시퀀스간의 거리 축적 테이블을 구성함으로써 그 후보 서브시퀀스를 최종 결과 집합에 포함시킬지의 여부를 결정한다. 그러나 이러한 기존의 방식은 거리 축적 테이블의 공유가 가능한 다수의 후보 서브시퀀스들에 대한 거리 축적 테이블을 개별적으로 구성하도록 하므로 CPU 처리의 낭비가 발생한다. 이것은 동일한 접미어의 접두어인 다수의 서브시퀀스들이 후보로 선택되는 경우에 발생한다.

이러한 문제를 해결하기 위하여 본 연구에서는 여과 단계에서 시퀀스 S의 각 접미어의 접두어에 해당되는 서브시퀀스들을 대상으로 여과 단계를 수행한다. 또한, 여기서 후보로 선정된 접두어들 중 길이가 가장 긴 접두어를 찾는다. 후처리 단계에서는 이 접두어와 질의 시퀀스간의 거리 축적 테이블을 구성함으로써 후보 서브시퀀스들의 최종 결과 집합내의 채택 여부를 결정한다. 이러한 최적화 기법을 통하여 동일한 접미어의 접두어인 서브시퀀스들에 대하여 거리 축적 테이블을 중복하여 구성하는 기존 방식의 문제점을 해결할 수 있다. 따라서 후처리 단계의 수행 시간이 크게 줄어든다.

4.3.2 ST-Filter와의 결합

ST-Filter의 여과 단계에서는 전체 시퀀스들을 대상으로 구축된 접미어 트리에 대하여 범위 질의가 수행되며, 여과 단계에서 후보로 지명된 서브시퀀스들이 <seqID, startOffset, endOffset> 형태로 반환된다. 여기서, seqID는 해당 후보 서브시퀀스가 포함되는 시퀀스의 위치를 나타내며, startOffset과 endOffset은 그 시퀀스내서 해당 후보 서브시퀀스가 존재하는 시작 위치 및 끝 위치를 나타낸다. 후처리 단계에서는 후보 서브시퀀스에 관한 이러한 정보를 이용하여 해당 시퀀스를 디스크로부터 액세스하고, 후보 서브시퀀스와 질의 시퀀스간의 거리 축적 테이블을 구성함으로써 그 후보 서브시퀀스를 최종 결과 집합에 포함시킬지의 여부를 결정한다.

접미어 트리 내에 저장된 서브시퀀스들은 동일한 시퀀스로부터 추출되었는지의 여부에 관계없이 독립적으로 관리된다. 또한, 기존의 ST-Filter의 후처리 과정에서는 ***접미어 트리 검색을 통하여 반환되는 순서***로 각 후보 서브시퀀스를 질의 시퀀스와 비교한다[17]. 이러한 처리 방식은 다음과 같은 두 가지 측면에서 성능상의 문제들을 야기 시킨다.

- **디스크 액세스 오버헤드** : 이것은 동일한 데이터 시퀀스를 디스크로부터 반복적으로 액세스함으로써 발생하는 성능상의 문제이다. 이 문제는 접미어 트리 검색의 결과, 같은 데이터 시퀀스에 속하는 서로 다른 서브시퀀스들이 후보로 선택되는 경우에 발생한다. 즉, 같은 시퀀스에 속하는 후보 서브시퀀스들이라 할 지라도 접미어 트리 검색의 결과로 반환되는 시점은 서로 다르므로 각각의 처리를 위하여 같은 데이터 시퀀스를 디스크로부터 여러 번 액세스해야 하는 것이다.
- **CPU 처리 오버헤드** : 이것은 거리 축적 테이블의 공유가 가능한 다수의 후보 서브시퀀스들에 대한 거리 축적 테이블을 개별적으로 구성함으로써 발생하는 성능상의 문제이다. 이것은 같은 시퀀스 내에 속하는 동일한 접미어의 접두어인 다수의 서브시퀀스들이 접미어 트리 검색의 결과 후보로 선택되는 경우에 발생한다. 즉, 이들이 후보로 선택되는 시점이 서로 다르므로 각각의 처리를 위하여 매번 거리 축적 테이블을 구성해야 하는 것이다.

위의 디스크 액세스 및 CPU 처리의 오버헤드가 발생하는 근본적인 원인은 ① 서브시퀀스들이 동일한 시퀀스로부터 추출되었는지의 여부에 관계없이 접미어 트리 내에 독립적으로 저장되며, ② 후처리 단계에서 접미어 트리 검색의 결과로 반환되는 순서 그대로 후보 서브시퀀스를 질의 시퀀스와 비교한다는 데에 있다. 본 연구에서는 이러한 문제를 해결하기 위하여 다음과 같은 최적화 기법을 제안한다. 제안하는 기법에서는 접미어 트리 검색의 결과로 반환되는 후보 서브시퀀스들의 처리 순서를 조정함으로써 동일한 시퀀스 내에 존재하는 동일한 접미어의 접두어인 후보 서브시퀀스들을 연속적으로 처리한다.

이를 위한 구체적인 방법은 다음과 같다. ① 각 후보 서브시퀀스의 식별자 <seqID, startOffset, endOffset>를 접미어 트리 검색 단계에 의하여 반환되는 순서대로 주기억장치 내에 저장한다. ② <seqID, startOffset, endOffset>를 정렬 키(sort key)로 사용하여 앞에서 저장한 모든 후보 서브시퀀스 식별자들을 정렬한다. ③ 단계 ②에 의하여 정렬된 순서로 각 시퀀스 S를 차례로 디스크로부터 액세스하고, 시퀀스 S 내 동일한 접미어의 접두어인 후보 서브시퀀스들에 대하여 하나의 거리 축적 테이블을 구성함으로써 이 후보 시퀀스들과 질의 시퀀스와의 타임 워핑 거리를 계산한다.

이러한 최적화 기법으로 인하여 후보 서브시퀀스를 포함하는 시퀀스는 디스크로부터 단 한번만 액세스되며, 동일한 접미어의 접두어인 서브시퀀스들에 대하여 거리 축적 테이블을 중복하여 구성하는 기존의 문제점을 모두 해결할 수 있다.

5. 성능 평가

본 장에서는 실험에 의한 성능 분석을 통하여 제안된 기법을 채택함으로써 나타나는 성능 개선 효과를 정량적으로 규명한다. 먼저, 제 5.1절에서는 성능 평가를 위한 실험 환경을 설명한다. 제 5.2절에서는 성능 분석 결과를 제시하고 분석한다.

5.1 실험 환경

본 연구에서는 성능 분석을 위하여 실제 데이터베이스 K_Stock_Data와 합성 데이터 Syn_Data를 사용하였다. K_Stock_Data는 한국의 실제 주식 데이터로서 길이가 300인 620개의 데이터 시퀀스들로 구성된다. 합성 데이터 Syn_Data은 다음과 같은 랜덤 워크(random walk) 형태로 생성된 시퀀스 $S = \langle s_1, s_2, \cdots, s_n \rangle$들의 집합이다.

$$s_i = s_{i-1} + z_i$$

여기서 z_i는 구간 $[-0.1, 0.1]$ 사이에서 균일한 분포를 취하는 랜덤 변수이며, 시퀀스의 첫 요소 값 s_1은 구간 $[1, 10]$ 사이의 임의의 값을 취하도록 하였다. 대형의 시계열 데이터베이스 환경에 대한 실험을 위하여 각각 1,000개, 2,000개, 3,000개, 4,000개의 길이가 200인 데이터 시퀀스들로 구성된 네 가지 Syn_Data들과 길이가 각각 200, 300, 400, 500인 1,000개의 데이터 시퀀스들로 구성된 네 가지 Syn_Data들을 대상으로 실험하였다.

질의 시퀀스 Q는 데이터베이스로부터 임의로 선택한 한 시퀀스로부터 길이가 Len(Q)인 임의의 위치의 서브시퀀스를 선택하여 "그대로" 사용하는 방법으로 생성하였다. 질의 구성 시에는 참고 문헌 [21]에 나타난 바와 같이, 질의 선택률(query selectivity)을 아래의 식과 같이 정의하고, 각 질의에 대하여 원하는 선택률을 만족하도록 허용치 ε을 조정하였다. 또한, 성능 지수로는 서브시퀀스 매칭의 수행 시간을 사용하였으며, 동일한 환경에서 서로 다른 질의 시퀀스 50개에 대한 수행 시간을 측정하여 그 평균값을 구하였다.

$$\text{선택률} = \frac{\text{데이터베이스 내에서 질의 시퀀스 Q와 } \varepsilon \text{ - 매치하는 모든 서브시퀀스들의 수}}{\text{데이터베이스 내에서 길이가 Len(Q)인 가능한 모든 데이터 서브시퀀스들의 수}}$$

실험을 위한 하드웨어 플랫폼은 1.7GHz Pentium IV와 1,280 MB의 주기억장치가 장착된 PC를 사용하였으며, 소프트웨어 플랫폼은 운영 체제 Linux Kernel Version 2.4.18 및 컴파일러 Glibc 2.2.4를 사용하였다. 실험 중 다른 시스템 및 사용자 프로세스들과의 상호 간섭을 방지하기 위하여 운영 체제를 단일 사용자 모드로 설정하여 모든 사용자 프로세스들을 제거한 상황에서 실험하였다. 또한, 버퍼링 효과를 피하고, 실제 디스크 액세스를 보장하도록 하기 위하여 버퍼를 이용하지 않는 I/O 시스템 콜(system call)을 사용하였다. ST_Filter를 위한 도메인 분류(domain categorization) 방법으로서 최대 엔트로피 기법(maximum entropy method)을 이용하여 도메인이 50개의 구간을 갖도록 하였다.

다음의 <표 2>는 각 실험에서 사용된 인자 값들을 요약한 것이다.

〈표 2〉 실험에서 사용된 인자 값들

	exp1	exp2	exp3	exp4	exp5
data set	K_Stock_ Data	K_Stock_ Data	K_Stock_ Data	Syn_ Data	Syn_ Data
seqNum	620	620	620	1000~4000	1000
seqLen	300	300	300	200	200~500
queryLen	110	110	80~260	60	60~240
maxWRatio	5	3~12	5	5	5
selectivity	7.1×10^{-8}~ 4.97×10^{-7}	2.13×10^{-7}	2.13×10^{-7}	1.47×10^{-7}	1.47×10^{-7}

5.2 실험 결과 및 분석

실험 1의 목적은 선택률의 변화에 따르는 제안된 최적화 기법의 성능 개선 효과를 규명하기 위한 것이다. 질의 선택률을 변화시키면서 제안된 최적화 기법과 기존 기법을 각각 후처리 단계에서 채택하는 Naive-Scan, LB-Scan, ST- Filter의 성능을 비교하였다. 사용된 질의 선택률은 7.10×10^{-8}(최종 결과 : 2개), 2.13×10^{-7}(최종 결과 : 6개), 3.55×10^{-7}(최종 결과 : 10개), 4.97×10^{-7}(최종 결과 : 14개)이다. 사용된 질의 시퀀스의 길이는 110이다.

<표 3>는 실험 1의 결과를 나타낸 것이다. F-Time과 PP-Time은 각각 여과 단계, 후처리 단계에서 소요되는 시간을 나타내며, Total은 전체 서브시퀀스 매칭의 수행 시간을 나타낸다. Original과 Optimized는 각각 Naive-Scan, LB-Scan, ST-Filter의 후처리 단계의 처리 방식으로서 기존의 기법과 제안된 최적화 기법을 의미한다. 또한, '# of candidates'는 LB-Scan과 ST-Filter의 여과 단계의 결과로 나타나는 후보 서브시퀀스들의 수를 나타낸다.

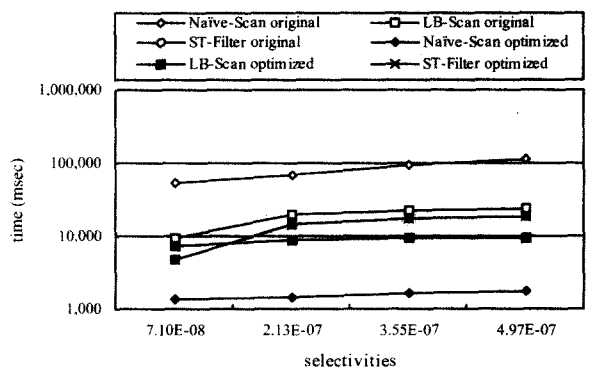
먼저, Naive-Scan의 결과를 분석한다. 제안된 최적화 기법은 기존의 방식과 비교하여 선택률에 따라 약 40배에서 62배까지의 매우 큰 성능 개선 효과를 가지는 것으로 나타났다. 이것은 제안한 테이블 공유 전략 및 테이블 커팅 전략이 매우 효과적임을 보여주는 것이다. LB-Scan 역시 제안된 기법을 사용함으로써 후처리 단계에서의 성능 향상 효과가 선택률에 따라 55배에서 117배까지 향상되는 것으로 나타났다. 이러한 후처리 단계에서의 성능 개선으로 인하여 전체 처리 시간 측면의 성능 개선 효과도 1.24에서 2.50배 사이로 나타났다.

〈표 3〉 선택률 변화에 따르는 제안된 기법의 성능 개선 효과

Selectivity		7.10×10^{-8}			2.13×10^{-7}			3.55×10^{-7}			4.97×10^{-7}		
Methods to be compared		Naive-Scan	LB- Scan	ST_Filter	Naive-Scan	LB- Scan	ST_Filter	Naive-Scan	LB- Scan	ST_Filter	Naive-Scan	LB- Scan	ST_Filter
F-Time	Original	0	7,418	4,680	0	8,447	14,523	0	8,940	17,719	0	9,196	19,121
	Optimized	**0**	**7,421**	**4,703**	**0**	**8,464**	**14,501**	**0**	**8,979**	**17,705**	**0**	**9,236**	**19,141**
PP-Time	Original	55,327	1,872	4	69,991	11,397	22	92,599	14,054	22	110,016	14,537	22
	Optimized	**1,379**	**16**	**2**	**1,483**	**98**	**4**	**1,648**	**189**	**6**	**1,773**	**263**	**7**
Total	Original	55,327	9,290	4,684	69,991	19,844	14,545	92,599	22,994	17,741	110,016	23,733	19,143
	Optimized	**1,379**	**7,437**	**4,705**	**1,483**	**8,562**	**14,505**	**1,648**	**9,168**	**17,711**	**1,773**	**9,499**	**19,148**
# of candidates		-	195,909	531	-	1,244,275	2,564	-	1,552,827	2,567	-	1,604,689	2,756

ST-Filter의 경우, 제안된 기법을 사용함으로써 얻어지는 성능 향상 효과가 거의 없는 것으로 나타났다. 이는 ST-Filter의 여과 단계를 통하여 나타난 후보 서브시퀀스들의 수가 매우 적으며, 뿐만 아니라 이들 대부분이 하나의 시퀀스에 포함되기 때문이다. 후처리 단계에서 소요된 수행 시간 측면에서는 성능 개선 효과가 있으나, 전체 소요 시간 측면에서 보면 거의 변화가 없는 수준이다. 즉, ST-Filter는 여과 단계에서의 수행 시간이 전체 수행 시간의 대부분을 차지한다.[5]

(그림 3)은 <표 2>의 전체 수행 시간을 쉽게 비교할 수 있도록 그래프 형태로 표현한 것이다. 가로축은 선택률의 크기를 나타내며, 세로축은 타임 워핑 하의 서브시퀀스 매칭의 전체 처리 시간을 나타낸다. 처리 시간이 로그 스케일로 표현되었음에 유의해야 한다. ST-Filter의 경우, 최적화의 성능 개선 효과가 미미하므로, 최적화 전과 후의 그래프가 거의 동일한 형태로 나타났다. LB-Scan의 경우, 최적화에 의하여 상당한 성능 개선의 효과를 보였다. Naive-Scan의 경우, 최적화로 인한 성능 개선 효과가 가장 두드러지게 나타났다. 특히, Naive-Scan이 후처리 단계의 최적화 전에는 가장 떨어지는 성능을 보였으나, 최적화 후에는 선택률의 변화에 관계없이 모든 경우에서 ST-Filter나 LB-Scan을 사용한 경우보다 훨씬 더 좋은 성능을 보이는 것으로 나타났다. 이것은 제안된 최적화 기법에서 사용한 테이블 공유 전략 및 테이블 커팅 전략의 성능 개선 효과가 Naive-Scan에서 가장 두드러지게 나타났음을 의미하는 것이다. 이러한 결과는 성능 병목인 CPU 처리 과정을 최적화함으로써 기존 기법들인 Naive-Scan, LB-Scan, ST-Filter 간의 처리 성능 상의 순위 역전 현상이 발생하였음을 보이는 것이다. 이러한 결과는 기존에 전혀 예상하지 못한 매우 놀라운 현상이다.



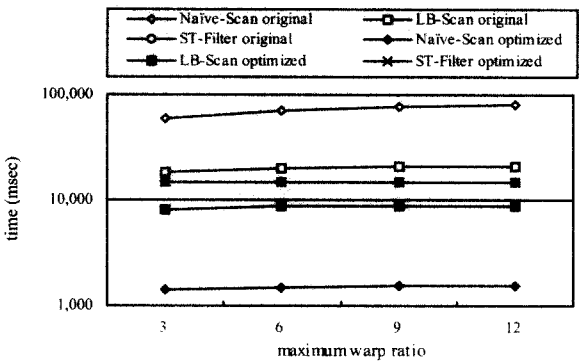
(그림 3) 선택률 변화에 따른 성능 결과

실험 2에서는 최대 WarpRatio[15]를 변화시키면서 제안된 최적화 기법과 기존 기법을 각각 후처리 단계에서 채택하는 Naive-Scan, LB-Scan, ST-Filter에 대한 실험을 수행하였다. 사용된 최대 WarpRatio는 3, 6, 9, 12이다. 실험 2의 목적은 최대 WarpRatio의 변화에 따라 제안된 최적화 기법의 성능 개선 효과가 어떻게 나타나는가를 규명하기 위한 것이다. 사용된 질의 시퀀스의 길이는 110이며, 선택률은 2.13×10^{-7}을 사용하였다.

(그림 4)는 실험 결과를 나타낸 것이다. 가로축은 최대 WarpRatio의 크기를 나타내며, 세로축은 타임 워핑 하의 서브시퀀스 매칭의 전체 수행 시간을 로그 스케일로 나타낸 것이다. 최대 WarpRatio가 증가함에 따라 모든 경우에서 전체 수행 시간은 점차 증가하는 것으로 나타났다. 이것은 최대 WarpRatio가 클수록 서브시퀀스 내의 각 요소 값이 반복될 수 있는 회수가 다양해지므로, CPU 처리 시간이 증가되기 때문이다. 실험 1의 결과와 마찬가지로, 제안

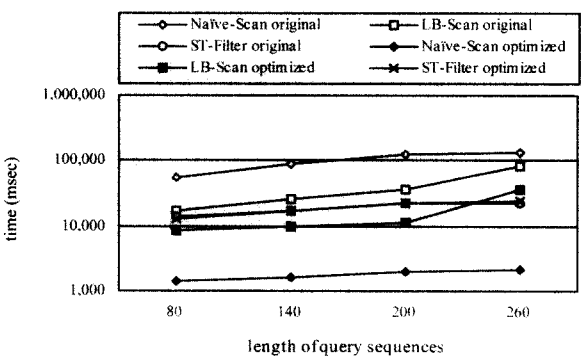
5) 이것은 여과 단계에서 사용하는 대형의 접미어 트리의 순회(traverse) 시간이 매우 크기 때문이다. 서브시퀀스 매칭에서 사용되는 접미어 트리의 크기는 일반적으로 실제 시계열 데이터베이스의 크기보다 훨씬 더 크다. 참고로, 본 실험에서 사용된 데이터베이스는 크기는 약 2.3MB이며, 이에 대한 접미어 트리의 크기는 약 48MB로서 데이터베이스 크기의 20배가 넘는 저장 공간을 차지한다.

된 최적화 기법은 ST-Filter를 제외한 LB-Scan 및 Naive-Scan에서 큰 성능 개선 효과를 보였다. 특히, 최적화로 인한 Naive-Scan의 성능 향상 효과가 42배에서 52배로 가장 큰 것으로 나타났으며, 이 결과 전체 수행 시간 측면에서 다른 모든 기법들과 비교하여 월등한 성능을 보였다. 이 최적화된 Naive-Scan은 두 번째로 좋은 성능을 보인 최적화된 LB-Scan에 비해 5배에서 6배까지 더 좋은 성능을 보였다.



(그림 4) 최대 WarpRatio 변화에 따른 성능 결과

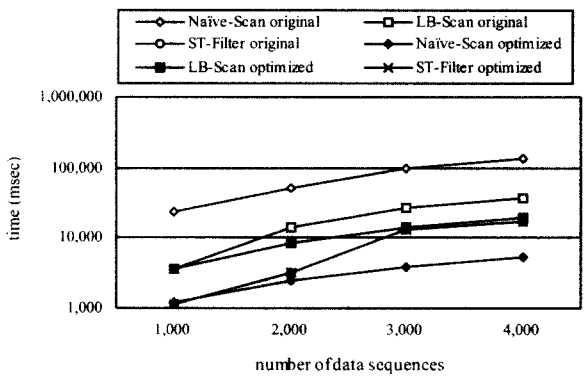
실험 3에서는 다양한 길이의 질의 시퀀스에 대하여 제안된 최적화 기법과 기존 기법을 각각 후처리 단계에서 채택하는 Naive-Scan, LB-Scan, ST-Filter에 대한 실험을 수행하였다. 사용된 질의 시퀀스의 길이는 80, 140, 200, 260이다. 실험 3의 목적은 질의 시퀀스 길이의 변화에 따르는 제안된 최적화 기법의 성능 개선 효과를 규명하기 위한 것이다. 사용된 최대 WarpRatio는 5이며, 선택률은 2.13×10^{-7}을 사용하였다.



(그림 5) 질의 시퀀스 길이의 변화에 따른 성능 결과

(그림 5)는 실험 결과를 나타낸 것이다. 가로축은 사용된 질의 시퀀스 길이를 나타내며, 세로축은 타임 워핑 하의 서브시퀀스 매칭의 전체 수행 시간을 로그 스케일로 나타낸 것이다. 질의 시퀀스의 길이가 길어짐에 따라 모든 경우에서 전체 수행 시간은 크게 증가하는 것으로 나타났다. 이는 서브시퀀스 S와 질의 시퀀스 Q에 대한 거리 축적 테이블을 구성하는 시간이 $O(|S|\times|Q|)$로 나타나기 때문이다. 전체적인 경향은 이전의 다른 실험들에서와 마찬가지로 나타났으며, Naive-Scan에서의 제안된 최적화 기법의 성능 향상 효과가 가장 크고, 이 결과 최적화된 Naive-Scan이 가장 좋은 처리 성능을 보였다.

전술한 바와 같이 K_Stock_Data는 소규모의 데이터이므로, 이후의 실험에서는 다양한 시퀀스의 수 및 시퀀스 길이를 갖는 대규모의 Syn_Data를 대상으로 각 기법의 성능을 분석한다. 실험 4에서는 1,000개에서 4,000개까지의 다양한 수의 시퀀스들을 가지는 데이터베이스를 대상으로 각 기법들의 수행 시간을 비교한다. 사용된 평균 시퀀스의 길이와 선택률은 각각 200과 1.47×10^{-7}이다. (그림 6)은 실험 결과를 나타낸 것이다. 가로축은 사용된 데이터베이스 내의 시퀀스들의 수를 나타내며, 세로축은 타임 워핑 하의 서브시퀀스 매칭의 전체 수행 시간을 로그 스케일로 나타낸 것이다.



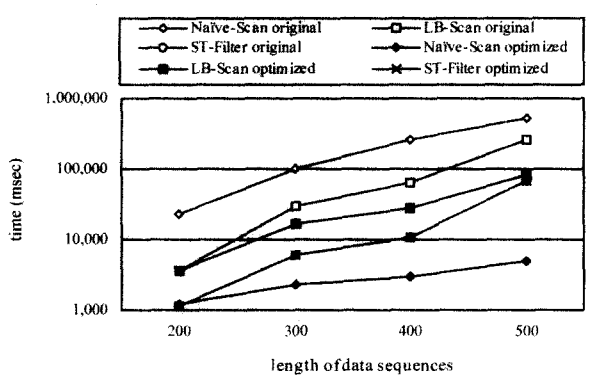
(그림 6) 시퀀스 개수의 변화에 따른 성능 결과

Naive_Scan, LB_Scan, ST_Filter 모두 시퀀스의 수가 증가함에 따라 수행 시간이 급격히 증가하는 것으로 나타났다. Naive_Scan과 LB_Scan의 경우에는 전체 시퀀스들을 모두 디스크로부터 액세스하기 때문이며, ST_Filter의 경우에는 크기가 시퀀스의 수에 비례하는 접미어 트리의 상당 부분을 순회해야 하기 때문이다. 특히, 최적화 전의 Naive-Scan은 수행 시간의 증가폭이 가장 크게 나타났으나, 최적화 이후에는 수행 시간이 19배에서 25배까지 크게 단축되는 모습을 보였다. 최적화된 Naive-Scan이 모든 경우에서 가장 좋은 성능을 보였으며, 두 번째로 좋은 성능을 보이는 기법에 비해 최대 약 3배 정도까지 좋은 수행 성능을 보였다. 또한, 이러한 성능 개선 효과는 데이터베이스 내의 시퀀스 수가 증가할수록 점차 커지는 것으로 나타났다.

끝으로, 실험 5에서는 200에서 500까지의 다양한 길이의 시퀀스들을 가지는 데이터들을 대상으로 각 기법의 수행

시간을 비교한다. 사용된 데이터베이스 내의 시퀀스들의 수와 선택률은 각각 1,000과 1.47×10^{-7}이다. 또한, 사용된 질의 시퀀스의 길이는 60, 120, 180, 240이다. (그림 7)은 실험 결과를 나타낸 것이다. 가로축은 사용된 데이터베이스 내의 시퀀스들의 길이를 나타내며, 세로축은 타임 워핑 하의 서브시퀀스 매칭의 전체 수행 시간을 로그 스케일로 나타낸 것이다.

실험 4에서와 같이, 최적화 전의 Naive_Scan, LB_Scan, ST_Filter는 시퀀스의 길이가 증가함에 따라 검색 성능이 급격히 저하되는 것으로 나타났다. 이들 중, 최적화 전에는 모든 경우에서 ST-Filter가 가장 나은 성능을 보였다. 최적화 후에는 Naive-Scan이 시퀀스의 길이가 증가함에 따라 최소 19배에서 최대 107배까지 가장 큰 성능 개선 효과를 보였다. 반면, LB-Scan은 최적화 후 시퀀스의 길이가 증가함에 따라 최적화 전과 비교하여 검색 시간이 크게 감소하며, 최대 3배까지의 성능 개선을 보였다. 끝으로, ST-Filter의 경우에는 전술한 바와 같이 인덱스 검색 단계 후에 나타나는 후보 서브시퀀스들의 수가 매우 작으므로 성능 개선 효과가 미미한 것으로 나타났다. 전체적으로 후처리 단계에서 최적화 기법을 채택한 Naive-Scan이 가장 좋은 성능을 나타냈으며, 두 번째로 좋은 성능을 보이는 기법에 비해 최대 약 3배 정도까지 좋은 수행 성능을 보였다.



(그림 7) 시퀀스 길이의 변화에 따른 성능 결과

6. 결 론

본 논문에서는 시계열 데이터베이스에서 타임 워핑 하의 서브시퀀스 매칭을 효과적으로 처리하는 방안에 대하여 논의하였다. 본 논문의 주요 공헌은 다음과 같이 요약될 수 있다.

- 기존의 가장 기본적인 처리 방식인 Naive-Scan의 성능 병목이 CPU 처리에 있음을 보이고, CPU 처리 과정의 최적화가 Naive-Scan의 전체 성능을 개선할 수 있는 매우 중요한 이슈임을 지적하였다.
- Naive-Scan의 CPU 처리 과정을 최적화하는 새로운 기법을 제안하였다. 제안된 기법은 질의 시퀀스와 서브시퀀스들간의 타임 워핑 거리들을 계산하는 과정에서 발생하는 많은 중복 작업을 사전에 제거함으로써 CPU 처리 성능을 극대화할 수 있다.
- 제안된 기법이 착오 기각을 발생시키지 않음과 Naive-Scan을 처리하기 위한 최적의 기법임을 이론적으로 증명하였다.
- 제안된 기법을 기존의 각 타임 워핑 하의 서브시퀀스 매칭 기법인 LB-Scan과 ST-Filter의 후처리 단계에 적용하는 방안을 제시하였다.
- 다양한 실험을 통한 성능 평가에 의하여 제안된 후처리 단계의 최적화 기법이 가져오는 성능 개선 효과를 정량적으로 검증하였다.

실험 결과에 의하면, 기존의 타임 워핑 하의 서브시퀀스 매칭을 위한 모든 기법들이 제안된 최적화 기법에 의하여 성능이 개선되는 것으로 나타났다. 특히, 가장 기본적인 기법인 Naive-Scan은 제안된 최적화 기법의 적용 전에는 가장 떨어지는 성능을 보였으나, 최적화 기법의 적용 후에는 모든 경우에서 ST-Filter나 LB-Scan을 사용한 경우보다 더 좋은 성능을 보였다. 이러한 결과는 기존에 전혀 예상하지 못한 매우 놀라운 것이며, 성능 병목인 CPU 처리 과정을 최적화함으로써 기존 기법들인 Naive-Scan, LB-Scan, ST-Filter 간의 처리 성능 상의 순위 역전 현상이 발생하였음을 보이는 것이다.

최적화된 Naive-Scan의 성능의 우수성은 (1) 시퀀스 길이가 길수록, (2) 저장된 시퀀스 개수가 많을수록, (3) 선택률이 작을수록 더 두드러지는 것으로 나타났다. 이러한 경향은 실제 대형 데이터베이스 환경의 특성을 고려할 때 매우 바람직한 것이다.

참 고 문 헌

[1] R. Agrawal, C. Faloutsos and A. Swami, "Efficient Similarity Search in Sequence Databases," In *Proc. Int'l. Conf. on Foundations of Data Organization and Algorithms*, FODO, pp.69-84, Oct., 1993.

[2] C. Chatfield, *The Analysis of Time-Series : An Introduction*, Third Edition, Chapman and Hall, 1984.

[3] R. Agrawal et al., "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases," In *Proc. Int'l. Conf. on Very Large Data Bases*, VLDB, pp.490-501, Sept., 1995.

[4] C. Faloutsos, M. Ranganathan and Y. Manolopoulos, "Fast Subsequence Matching in Time-series Databases," In *Proc. Int'l. Conf. on Management of Data*, ACM SIGMOD, pp.419-429, May, 1994.

[5] M. S., Chen, J., Han and P. S., Yu, "Data Mining : An Overview from Database Perspective," *IEEE Trans. on Knowledge and Data Engineering*, Vol.8, No.6, pp.866-883, 1996.

[6] D. Rafiei and A. Mendelzon, "Similarity-Based Queries for Time-Series Data," In *Proc. Int'l. Conf. on Management of Data*, ACM SIGMOD, pp.13-24, 1997.

[7] B. K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary L_p Norms," In *Proc. Int'l. Conf. on Very Large Data Bases*, VLDB, pp.385-394, 2000.

[8] K. P. Chan and A. W. C. Fu, "Efficient Time Series Matching by Wavelets," In *Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp.126-133, 1999.

[9] K. K. W. Chu and M. H. Wong, "Fast Time-Series Searching with Scaling and Shifting," In *Proc. Int'l. Symp. on Principles of Database Systems*, ACM PODS, pp.237-248, May, 1999.

[10] D. Q. Goldin and P. C. Kanellakis, "On Similarity Queries for Time-Series Data : Constraint Specification and Implementation," In *Proc. Int'l. Conf. on Principles and Practice of Constraint Programming*, CP, pp.137-153, Sept., 1995.

[11] D. Rafiei, "On Similarity-Based Queries for Time Series Data," In *Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp.410-417, 1999.

[12] G. Das, D. Gunopulos and H. Mannila, "Finding Similar Time Series," In *Proc. European Symp. on Principles of Data Mining and Knowledge Discovery*, PKDD, pp. 88-100, 1997.

[13] W. K. Loh, S. W. Kim and K. Y. Whang, "Index Interpolation : An Approach for Subsequence Matching Supporting Normalization Transform in Time-Series Databases," In *Proc. ACM Int'l. Conf. on Information and Knowledge Management*, ACM CIKM, pp.480-487, 2000.

[14] W. K. Loh, S. W. Kim and K. Y. Whang, "Index Interpolation : A Subsequence Matching Algorithm Supporting Moving Average Transform of Arbitrary Order in Time-Series Databases," *IEICE Trans. on Information and Systems*, Vol.E84-D, No.1, pp.76-86, 2001.

[15] D. J. Berndt and J. Clifford, "Finding Patterns in Time Series : A Dynamic Programming Approach," *Advances in Knowledge Discovery and Data Mining*, pp.229-248, 1996.

[16] B. K. Yi, H. V. Jagadish and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences Under Time Warping," In *Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp.201-208, 1998.

[17] S. H. Park et al., "Efficient Searches for Similar Subsequences of Difference Lengths in Sequence Databases," In *Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp.23-32, 2000.

[18] S. W. Kim, S. H. Park and W. W. Chu, "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases," In *Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp.607-614, 2001.

[19] S. H. Park, S. W. Kim, J. S. Cho and S. Padmanabhan, "Prefix-Querying : An Approach for Effective Subsequence Matching Under Time Warping in Sequence Databases," In *Proc. ACM Int'l. Conf. on Information and Knowledge Management*, ACM CIKM, pp.255-262, 2001.

[20] L. Rabiner and H. H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, 1993.

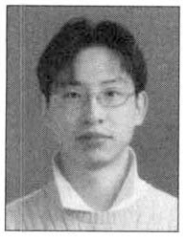
[21] Y. S. Moon, K. Y. Whang and W. K. Loh, "Duality-Based Subsequence Matching in Time-Series Databases," In Proc. *Int'l Conf. on Data Engineering*, IEEE ICDE, pp.263-272, 2001.

[22] Y. S. Moon, K. Y. Whang and W. S. Han, "GeneralMatch : A Subsequence Matching Method in Time-Series Databases Based on Generalized Windows," In Proc. *Int'l. Conf. on Management of Data*, ACM SIGMOD, 2002.

[23] S. H. Park, *private communication*, 2003.

[24] G. A. Stephen, *String Searching Algorithms*, World Scientific Publishing, 1994.

김 만 순



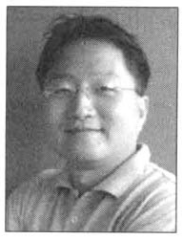
e-mail : beanjam@empal.com

1998년 강원대학교 삼림경영학과(학사)

1998년~1999년 중부임업시험장 삼림조사 연구소 연구 보조

2000년~2003년 강원대학교 컴퓨터정보통신공학과(석사)

김 상 욱



e-mail : wook@hanyang.ac.kr

1989년 서울대학교 컴퓨터공학과(학사)

1991년 한국과학기술원 전산학과(석사)

1994년 한국과학기술원 전산학과(박사)

1991년 미국 Stanford University, Computer Science Department 방문 연구원

1994년~1995년 KAIST 정보전자연구소 전문 연구원

1999년~2000년 미국 IBM T.J. Watson Research Center Post-Doc.

1995년~2000년 강원대학교 컴퓨터정보통신공학부 부교수

2003년~현재 한양대학교 정보통신대학 정보통신학부 부교수

관심분야 : 데이타베이스 시스템, 저장 시스템, 트랜잭션 관리, 데이타 마이닝, 멀티미디어 정보 검색, 공간 데이터베이스/GIS, 주기억장치 데이타베이스