시계열 데이타베이스에서 서브시퀀스 매칭의 성능 병목: 관찰, 해결 방안, 성능 평가

(The Performance Bottleneck of Subsequence Matching in Time-Series Databases: Observation, Solution, and Performance Evaluation)

김 상 욱 †

(Sang-Wook Kim)

요 약 서브시퀀스 매칭은 주어진 질의 시퀀스와 변화의 추세가 유사한 서브시퀀스들을 시계열 데이타베이스로부터 검색하는 연산이다. 본 논문에서는 서브시퀀스 매칭 처리의 성능 병목을 파악하고, 이를 해결함으로써 전체 서브시퀀스 매칭의 성능을 크게 개선하는 방안에 관하여 논의한다. 먼저, 사전 실험을 통하여 전체 서브시퀀스 매칭의 처리 시간 중 인덱스 검색 단계와 후처리 단계에서 디스크 액세스 시간 및 CPU 처리 시간이 차지하는 비중을 분석한다. 이를 바탕으로 후처리 단계가 서브시퀀스 매칭의 성능 병목이며, 후처리 단계의 최적화가 기존의 서브시퀀스 매칭 기법들이 간과한 매우 중요한 이슈임을 지적한다. 이러한 서브시퀀스 매칭의 성능 병목을 해결하기 위하여 후처리 단계를 최적으로 처리할 수 있는 간단하면서도 매우 효과적인 기법을 제안한다. 제안된 기법은 후처리 단계에서 후보 서브시퀀스들이 질의 시퀀스와 실제로 유사한가를 판단하는 순서를 조정함으로써 기존의 후처리 단계의 처리에서 발생하는 많은 디스크 액세스의 중복과 CPU 처리의 중복을 완전히 제거할 수 있다. 제안된 기법이 착오 기각을 발생시키지 않음과 후처리 단계를 처리하기 위한 최적의 기법임을 이론적으로 증명한다. 또한, 실제 데이타와 생성 데이타를 이용한 다양한 실험들을 통하여 제안된 기법의 성능 개선 효과를 정량적으로 검증한다. 실험 결과에 의하면, 제안된 기법은 기존 기법의 후처리 단계 수행 시간을 실제 주식 데이타를 이용한 실험의 경우 3.91 배에서 9.42배까지, 대규모의 생성 데이타를 이용한 실험의 경우 4.97 배에서 5.61배까지 개선시키는 것으로 나타났다. 또한, 제안된 기법을 채택함으로써 전체 서브시퀀스 매칭 처리 시간의 90%에 이르던 후처리 단계의 비중을 70%이하로 내릴 수 있었다. 이것은 제안된 기법이 서브시퀀스 매칭의 성능 병목을 성공적으로 해결하였음을 보여주는 것이다. 이 결과, 제안된 기법은 전체 서브시퀀스 매칭의 성능을 실제 주식 데이타를 사용한 실험의 경우 3.05 배에서 5.60 배까지, 대규모의 생성 데이타를 이용한 실험의 경우 3.68 배에서 4.21 배까지 개선시킬 수 있었다.

키워드 : 시계열 데이타베이스, 서브시퀀스 매칭, 성능 분석, 인덱스 검색, 후처리

Abstract Subsequence matching is an operation that finds subsequences whose changing patterns are similar to a given query sequence from time-series databases. This paper points out the performance bottleneck in subsequence matching, and then proposes an effective method that improves the performance of entire subsequence matching significantly by resolving the performance bottleneck. First, we analyze the disk access and CPU processing times required during the index searching and post processing steps through preliminary experiments. Based on their results, we show that the post processing step is the main performance bottleneck in subsequence matching, and then claim that its optimization is a crucial issue overlooked in previous approaches. In order to resolve the performance bottleneck, we propose a simple but quite effective method that processes the post processing step in the optimal way. By rearranging the order of candidate subsequences to be compared with a query

· 본 연구는 한국과학재단 목적기초연구사업(과제번호: R05-2002-01085-0)과 2003년도 한양대학교 교내연구비의 지원으로 수행되었음

† 종신회원 : 한양대학교 정보통신대학 정보통신학부 교수

wook@hanyang.ac.kr

논문접수 : 2002년 12월 7일

심사완료 : 2003년 3월 29일

sequence, our method completely eliminates the redundancy of disk accesses and CPU processing occurred in the post processing step. We formally prove that our method is optimal and also does not incur any false dismissal. We show the effectiveness of our method by extensive experiments. The results show that our method achieves significant speed-up in the post processing step 3.91 to 9.42 times when using a data set of real-world stock sequences and 4.97 to 5.61 times when using data sets of a large volume of synthetic sequences. Also, the results show that our method reduces the weight of the post processing step in entire subsequence matching from about 90% to less than 70%. This implies that our method successfully resolves the performance bottleneck in subsequence matching. As a result, our method provides excellent performance in entire subsequence matching. The experimental results reveal that it is 3.05 to 5.60 times faster when using a data set of real-world stock sequences and 3.68 to 4.21 times faster when using data sets of a large volume of synthetic sequences compared with the previous one.

Key words : time-series databases, subsequence matching, performance analysis, index search, post-processing

1. 서 론

시계열 데이타베이스(time-series database)란 객체의 변화되는 값들의 연속으로 구성된 데이타 시퀀스(data sequence: 이후부터 간략히 시퀀스라 칭함)들의 집합이다[1]. 대표적인 예로는 주가 데이타, 환율 데이타, 기온 데이타, 제품 판매량 데이타, 기업 성장률 데이타 등이 있다[2][3][4]. 유사 시퀀스 매칭(similar sequence matching)이란 주어진 질의 시퀀스(query sequence)와 변화의 패턴이 유사한 시퀀스들을 시계열 데이타베이스로부터 찾아내는 연산이며[1][3][4][5], 데이타 마이닝(data mining) 및 데이타 웨어하우징(data warehousing) 분야에서 중요한 연산으로 사용된다[6][7].

유사 시퀀스 매칭에 관한 기존의 많은 연구에서는 길이 n의 시퀀스를 n 차원 공간상의 한 점으로 매핑 한다. 또한, 길이가 동일한 서로 다른 두 시퀀스 $X(=<x_0, x_1, ..., x_{n-1}>)$와 $Y(=<y_0, y_1, ..., y_{n-1}>)$간의 유사성을 측정하는 척도로서 아래와 같이 정의되는 유클리드 거리(Euclidean distance) D(X, Y)를 널리 사용한다[1][8][9][4][10][7][11][1]. D(X, Y)가 ε이하인 임의의 두 시퀀스 X, Y는 ε-매치(ε-match)한다고 한다[5].

$$D(X,Y) = \sqrt{\sum_{i=0}^{n-1} (x_i - y_i)^2}$$

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

- 전체 매칭: 데이타베이스 D 내에 존재하는 시퀀스 $S_1, S_2, ..., S_N$에 대하여 질의 시퀀스 Q와 허용치 ε이 주어질 때, D로부터 Q와 ε-매치하는 시퀀스 S_i를 검색한다. 이때, $S_1, S_2, ..., S_N$ 및 Q의 길이는 동일해야 한다.
- 서브시퀀스 매칭: 데이타베이스 D 내에 존재하는 시퀀스 $S_1, S_2, ..., S_N$에 대하여 질의 시퀀스 Q와 허용치 ε이 주어질 때, D로부터 Q와 ε-매치 하는 서브시퀀스 X를 포함하는 시퀀스 S_i와 이 서브시퀀스 X가 S_i내에서 시작하는 위치를 검색한다. 이때, $S_1, S_2, ..., S_N$ 및 Q는 서로 다른 길이를 갖는 것이 허용된다.

전체 매칭과는 달리, 서브시퀀스 매칭은 사용되는 데이타 및 질의 시퀀스의 길이에 대한 제약이 없으므로 실제 응용 분야에서 널리 사용된다. 따라서 본 논문은 이러한 서브시퀀스 매칭을 연구의 대상으로 한다.

참고 문헌 [4] 및 [5]에서는 서브시퀀스 매칭을 위한 처리 기법을 제안하였다. 참고 문헌 [5]의 명칭을 따라 본 논문에서는 [4]의 기법을 FRM, [5]의 기법을 Dual-Match라 부른다. 두 기법 모두 미리 지정된 고정된 길이 w를 갖는 윈도우(window) 개념을 이용한다. 서브시퀀스 매칭을 위하여 두 기법이 취하는 공통적인 아이디어는 다음과 같다.

먼저, 인덱스 구성을 위하여 각 시퀀스로부터 길이 w의 윈도우들을 추출하고, 각 윈도우를 이산 퓨리에 변환(discrete Fourier transform: DFT) 혹은 웨이블릿 변환(wavelet transform)을 이용하여 저차원 f 공간(f << w) 상의 윈도우 점(window point)으로 변환한다. 효과적인 서브시퀀스 매칭을 위하여 이러한 윈도우 점들을 다차원 인덱스(multidimensional index)의 하나인 R^*-트리[17]에 저장한다.

1) 유사성을 측정하기 위한 척도로서 유클리드 거리 이외에도 응용에 따라 맨하탄 거리(Manhattan distance)[12], 대응되는 요소 쌍의 최대 거리(maximum distance in any pair of elements)[3], 타임 워핑 거리(time warping distance)[13][14][15][16] 등이 사용될 수 있다.

허용치가 ε인 서브시퀀스 매칭의 처리를 위하여 길이 l인 질의 시퀀스로부터 길이 w의 윈도우들을 추출하고, 각 윈도우를 DFT 혹은 웨이블릿 변환(wavelet transform)을 이용하여 저차원 f 공간(f << w) 상의 윈도우 점으로 변환한다. 각 윈도우 점에 대하여 $\varepsilon/\sqrt{p}$ ($p = \lfloor l/w \rfloor$)를 허용치로 갖는 범위 질의를 R^*-트리 상에서 수행한다. 본 논문에서는 이 과정을 인덱스 검색 단계(index search step)라 부른다. 이러한 인덱스 검색 단계의 결과, 질의 시퀀스와 ε-매치 할 가능성이 높은 많은 후보 서브시퀀스(candidate subsequence)들이 반환된다. 그 다음, 착오 채택(false alarm)[1][4]을 해결하기 위하여 이 후보 서브시퀀스들을 포함하는 각 시퀀스를 디스크로부터 액세스하여 후보 서브시퀀스가 질의 시퀀스와 실제로 ε-매치 하는가의 여부를 판단한다. 본 논문에서는 이 과정을 후처리 단계(post-processing step)라 부른다.

FRM은 데이타 시퀀스로부터는 모든 가능한 위치에서 시작되는 슬라이딩 윈도우(sliding window)들을 추출하고, 질의 시퀀스로부터는 디스조인트 윈도우(disjoint window)들을 추출하는 방식을 사용한다[4]. 따라서 인덱싱의 대상이 되는 슬라이딩 윈도우 점들의 수가 매우 많아지며, 이 결과 R^*-트리를 위한 저장 공간의 오버헤드와 인덱스 검색 성능이 저하된다. FRM은 이 문제를 해결하기 위하여 각각의 윈도우 점 대신 다수의 윈도우 점들을 포함하는 최소 포함 사각형(minimum bounding rectangle: MBR)들을 R^*-트리 내에 저장한다. 그러나 FRM에서는 MBR 내의 죽은 공간(dead space)[17]으로 인하여 인덱스 검색 단계에서 발생하는 착오 채택(false alarm)[1]의 수가 크게 증가한다. 따라서 서브시퀀스 매칭의 성능이 저하된다는 문제점이 있다.

Dual-Match는 FRM과는 반대로 데이타 시퀀스로부터는 디스조인트 윈도우들을 추출하고, 질의 시퀀스로부터는 슬라이딩 윈도우들을 추출하는 방식을 사용한다[5]. 이와 같은 역할 교환으로 Dual Match는 FRM과 비교하여 R^* 트리 내에 저장할 윈도우 점들의 수를 약 1/w로 줄일 수 있다. 이 결과, MBR들을 저장하는 FRM과는 달리 Dual-Match에서는 윈도우 점 자체를 R^* 트리 내에 저장하는 것이 가능하다. 따라서 인덱스 검색 단계에서 MBR 내의 죽은 공간으로 인한 착오 채택이 제거되며, 이 결과 서브시퀀스 매칭의 처리 성능이 크게 향상된다.

그러나 데이타베이스 환경에서 Dual Match를 이용한 서브시퀀스 매칭은 아직도 수 초에서 수십 초의 매우 긴 처리 시간을 요구한다[5]. 따라서 서브시퀀스 매칭의 빠른 처리를 위한 추가의 성능 개선 방안이 필요하다. 이를 위하여 본 논문에서는 서브시퀀스 매칭 처리의 성능 병목(performance bottleneck)을 파악하고, 이를 신속하게 처리하기 위한 간단하면서도 매우 효과적인 기법을 제안한다. 본 논문의 주요 공헌을 간략히 요약하면 다음과 같다.

- 전체 서브시퀀스 매칭의 처리 과정에서 요구되는 시간을 IS-Time(CPU), IS-Time(DISK), PP-Time(CPU), PP-Time(DISK)의 네 가지 요소들로 분류하고, 사전 실험을 통하여 전체 처리 시간 중 각 요소가 차지하는 비중을 규명한다. 여기서, IS-Time(CPU), IS-Time(DISK), PP-Time(CPU), PP-Time(DISK)는 각각 인덱스 검색 단계의 CPU 처리 시간과 디스크 액세스 시간, 후처리 단계의 CPU 처리 시간과 디스크 액세스 시간을 의미한다.
- 후처리 단계가 서브시퀀스 매칭의 성능 병목이며, 후처리 단계의 최적화가 FRM 및 Dual-Match 등 기존 기법들이 간과한 매우 중요한 이슈임을 지적한다.
- 성능 병목을 해결하기 위하여 후처리 단계를 최적으로 처리할 수 있는 기법을 제안한다. 제안된 기법은 후보 서브시퀀스들의 질의 시퀀스와의 ε-매치 여부를 판단하는 순서를 조정함으로써 착오 기각(false dismissal)[1][4] 없이 후처리 단계의 처리 성능을 극대화할 수 있다.
- 제안된 기법이 착오 기각을 발생시키지 않음과 후처리 단계를 처리하기 위한 최적의 기법임을 이론적으로 증명한다.
- 다양한 실험들을 기반으로 하는 성능 평가를 통하여 제안된 기법이 얼마만큼의 성능 개선 효과를 갖는가를 정량적으로 검증한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구로서 서브시퀀스 매칭에 대한 기존의 기법들을 소개하고, 장단점을 논의한다. 제 3장에서는 사전 실험 결과를 해석하고, 후처리 단계가 전체 서브시퀀스 매칭의 성능 병목임을 보인다. 또한, 후처리 단계의 최적화가 FRM 및 Dual Match 등 기존 기법들이 간과한 매우 중요한 이슈임을 지적한다. 제 4장에서는 기존 기법들의 후처리 단계 처리 방식의 문제점을 지적하고, 이러한 문제점을 해결하는 새로운 기법을 제안한다. 제 5장에서는 제안된 기법의 우수성을 다양한 성능 평가를 통하여 검증한다. 끝으로, 제 6장에서는 본 논문을 요약하고, 결론을 내린다.

2. 관련 연구

본 장에서는 관련 연구로서 유클리드 거리 기반의 유사 모델을 사용하는 기존의 유사 시퀀스 매칭 기법들을 소개하고, 그 장단점에 관하여 논의한다[2].

2.1 전체 매칭

참고 문헌 [1]에서는 모든 데이타 시퀀스들과 질의 시퀀스의 길이가 동일하다는 전제 하에 전체 매칭 기법을 제안하였다. 먼저, 길이 l인 각 데이타 시퀀스를 DFT를 이용하여 저차원 f-공간(f << l) 상의 점으로 변환하고, 이들을 R^*-트리[Bec90]에 저장함으로써 인덱싱을 수행한다. 여기서, 저차원 변환은 R^*-트리 등의 다차원 인덱스(multidimensional index)가 가지는 고차원 문제(dimensionality curse)[1][18][19]를 해결하기 위하여 사용된다.

전체 매칭을 위하여 길이가 l인 질의 시퀀스를 위와 동일한 방법으로 저차원 f-공간상의 한 점으로 변환하고, 변환한 질의 점을 중심점, 허용치 ε을 범위로 사용하는 범위 질의를 구성한다. 사전에 구성된 R^*-트리에 대하여 이 범위 질의를 처리하는 인덱스 검색 단계(index search step)를 수행한다. 인덱스 검색 단계의 결과, 질의 점과 유클리드 거리가 ε 이하인 모든 데이타 점들과 대응되는 데이타 시퀀스들을 반환하는데, 이들을 후보 집합(candidate set)이라 한다.

질의 시퀀스와 ε-매치 하는 데이타 시퀀스들을 후보 집합에 포함시키지 못하는 현상을 착오 기각(false dismissal)이라 한다. 반면, 질의 시퀀스와 ε-매치 하지 않는 일부의 데이타 시퀀스들을 후보 집합에 포함시키는 현상을 착오 채택(false alarm)이라 한다. 참고 문헌 [1]에서는 Parseval 정리를 이용하여 이 기법이 착오 기각을 유발하지 않음을 증명하였다. 그러나 저차원 변환 시의 정보 손실로 인한 착오 채택이 발생할 수 있다. 착오 채택을 해결하기 위하여 범위 질의로 구한 후보 집합에 대하여 후처리 단계(post-processing step)를 수행한다. 후처리 단계에서는 디스크로부터 해당 데이타 시퀀스를 액세스하고, 이 시퀀스가 질의 시퀀스와 실제로 ε 매치 하는가의 여부를 조사함으로써 착오 채택을 제거한 최종 질의 결과를 생성한다.

2) 응용에 따라 유클리드 거리만을 이용한 시퀀스 매칭을 통해서는 사용자가 원하는 시퀀스들을 검색하지 못하는 경우가 발생할 수 있다. 따라서 응용 분야의 특성에 따라 유사한 정도를 유연하게 정의할 수 있도록 변환(transformation)이 사용된다. 이러한 변환을 함께 지원하는 유사 시퀀스 매칭 기법들이 다음의 참고 문헌들에 제안되어 있다. 정규화(normalization) 지원 기법: [3][10][7][20][9][21]. 이동 평균(moving average) 지원 기법: [7][11][22]. 타임 워핑(time warping) 지원 기법: [13][15][7][14][16][23].

2.2 FRM

참고 문헌 [4]에서는 임의의 길이를 갖는 질의 시퀀스와 데이타 시퀀스들을 대상으로 하는 서브시퀀스 매칭 방법 FRM을 제안하였다. FRM은 전체 매칭 기법의 기본 아이디어를 확장한 것으로서, R^*-트리 인덱싱을 위하여 고정된 길이 w를 갖는 윈도우(window) 개념을 이용한다.

먼저, 길이가 Len(S)인 각 데이타 시퀀스 S로부터 모든 가능한 위치에서 시작되는 길이 w의 슬라이딩 윈도우(sliding window)들을 추출하고, DFT를 이용하여 각 윈도우를 저차원 f-공간상(f << w)의 한 점으로 변환한다. 본 논문에서는 이 점을 데이타 윈도우 점(data window point)이라 정의한다. 시퀀스 S로부터 슬라이딩 윈도우들을 추출하므로 각 시퀀스 당 생성되는 데이타 윈도우 점들의 수는 (Len(S)-w+1)이 된다. 이 결과, 인덱싱의 대상이 되는 전체 데이타 윈도우 점들의 수가 매우 많아지므로, R^*-트리를 위한 저장 공간이 커지며, 이 결과 인덱스 검색 단계의 성능이 크게 저하된다[4]. FRM에서는 이러한 문제를 해결하기 위하여 다수의 데이타 윈도우 점들을 포함하는 최소 포함 사각형(minimum bounding rectangle: MBR)들을 구성한 후, 이 MBR들을 R^*-트리[17]에 저장하는 방법을 사용하였다.

서브시퀀스 매칭을 위해서는 길이 Len(Q)인 질의 시퀀스 Q로부터 길이 w인 ⌊Len(Q)/w⌋개의 디스조인트 윈도우(*disjoint window*)들을 추출하고, DFT를 이용하여 각 윈도우를 저차원 f-공간상의 점으로 변환한다. 본 논문에서는 이를 질의 윈도우 점(*query window point*)이라 정의한다. 각 질의 윈도우 점에 대하여, 그 질의 윈도우 점을 중심점, 허용치 $\varepsilon/\sqrt{p}$ ($p = \lfloor Len(Q)/w \rfloor$)를 범위로 사용하는 범위 질의를 구성한다. 사전에 만들어진 R^*-트리에 대하여 이 범위 질의를 처리하는 인덱스 검색 단계를 수행함으로써 후보 집합을 구한다. 후보 집합은 저차원 f-공간상에서 질의 윈도우 점과의 유클리드 거리가 $\varepsilon/\sqrt{p}$ 이하인 데이타 윈도우 점들을 포함한다. 이러한 점들과 대응되는 데이타 윈도우들을 후보 윈도우(candidate window)라 정의한다. 각 후보 윈도우는 최종 질의 결과에 포함될 가능성이 높은 후보 서브시퀀스(candidate subsequence)와 일대일 대응된다. 최종적으로, 착오 채택을 해결하기 위하여 각 후보 서브시퀀스를 디스크로부터 액세스하여 질의 시퀀스와 ε-매치 하는가를 판단하는 후처리 단계를 수행한다. 참고 문헌 [4]에서는 이러한 서브시퀀스 매칭 기법에서 착오 기각이 발생하지 않음을 증명하였다.

2.3 Dual-Match

FRM에서는 인덱싱을 위한 저장 공간의 오버헤드를 줄이기 위하여 개별적인 데이타 윈도우 점들 대신 다수의 윈도우 점들을 포함하는 MBR들을 R*-트리 내에 저장한다. 그러나 이러한 MBR 내부에는 죽은 공간(dead space)[17]이 존재하게 되므로, 이로 인하여 후보 서브시퀀스의 착오 채택이 발생되며, 이것은 처리 성능의 저하로 직결된다[5]. 참고 문헌 [5]에서는 이러한 문제점을 해결하기 위한 방법으로서 이원성 기반 서브시퀀스 매칭(duality-based subsequence matching : Dual Match)을 제안하였다.

Dual-Match에서는 길이가 Len(S)인 데이타 시퀀스 S로부터 슬라이딩 윈도우를 추출하고 길이 Len(Q)인 질의 시퀀스 Q로부터 디스조인트 윈도우를 추출하는 FRM과는 반대로 데이타 시퀀스로부터는 디스조인트 윈도우를 추출하고 질의 시퀀스로부터는 슬라이딩 윈도우를 추출하는 방식을 사용한다. 이와 같은 역할 교환을 통하여 Dual-Match는 각 시퀀스로부터 ⌊Len(S)/w⌋개(여기서 w는 윈도우의 길이)의 데이타 윈도우 점들만을 추출하게 되므로 R*-트리에 저장할 데이타 윈도우 점들의 수를 FRM의 약 1/w로 줄일 수 있다. 이 결과, R*-트리에 MBR을 저장하는 FRM과는 달리 Dual-Match에서는 윈도우 점 자체를 저장하는 것이 가능해진다. 따라서 MBR 내의 죽은 공간으로 인한 후보 서브시퀀스의 착오 채택이 발생되지 않으므로, 처리 성능이 크게 개선된다. 참고 문헌 [5]에서는 다양한 실험을 통하여 Dual-Match의 검색 성능이 FRM과 비교하여 크게 개선됨을 보였으며, Dual Match가 착오 기각 없이 서브시퀀스 매칭을 올바르게 수행함을 증명하였다.

3. 서브시퀀스 매칭의 성능 병목

본 장에서는 서브시퀀스 매칭의 성능 병목과 그 해결 방안에 관하여 논의한다. 먼저, 제 3.1절에서는 서브시퀀스 매칭의 처리 시간을 네 가지 요소로 구분하고, 전체 처리 시간 중 각 요소가 차지하는 비중을 사전 실험을 통하여 분석한다. 제 3.2절에서는 후처리 단계가 서브시퀀스 매칭의 성능 병목임을 지적하고, 후처리 단계의 최적화가 FRM 및 Dual Match 등 기존 기법들에서 간과한 매우 중요한 이슈임을 보인다.

3.1 사전 실험 결과 및 분석

서브시퀀스 매칭은 크게 인덱스 검색 단계와 후처리 단계로 구분된다. 본 논문에서는 전체 서브시퀀스 매칭의 처리 과정에서 요구되는 시간적 요소를 IS Time (CPU), IS-Time(DISK), PP-Time(CPU), PP-Time (DISK)의 네 가지로 세분화한다. 여기서, IS-Time (CPU)와 IS-Time(DISK)는 각각 인덱스 검색 단계에서 요구되는 CPU 처리 시간과 디스크 액세스 시간을 의미하며, PP-Time(CPU)와 PP-Time(DISK)는 후처리 단계에서 요구되는 CPU 처리 시간과 디스크 액세스 시간을 의미한다. 이와 같이, 전체 수행 시간을 네 가지 요소로 구분하는 이유는 서브시퀀스 매칭의 성능 병목을 보다 정확하게 파악하고, 이를 해결하기 위한 전략을 수립하기 위해서이다. 본 연구에서는 먼저 위에서 서브시퀀스 매칭의 전체 처리 시간 중 각 요소가 차지하는 비중을 분석하기 위하여 다음과 같은 사전 실험을 수행하였다.

실험 환경으로 사용된 하드웨어 플랫폼은 768 MB 크기의 주기억장치와 700 MHz Pentium III를 탑재한 PC이며, 소프트웨어 플랫폼은 MS Windows 2000이다. 데이타 페이지 및 R*-트리 인덱스 페이지의 크기로 4KB를 사용하였다. 참고 문헌 [5]의 성능 평가에서와 같이, 데이타 및 인덱스 파일을 액세스할 때, 버퍼링 효과를 제거한 Windows 2000의 I/O 시스템 콜(system call)을 사용함으로써 매 페이지 액세스 시 실제 디스크 액세스를 보장하였다.

사용된 실험 데이타는 길이가 1,024인 620개의 한국의 실제 주식 데이타이다. 저차원 변환을 위해서는 DFT를 사용하고, 각 시퀀스로부터 인덱싱을 위한 6개의 특성을 추출함으로써 6차원 R* 트리를 구성하였다[3]. 사용된 질의 시퀀스의 길이는 768이며, 허용치 ε은 15개의 최종 질의 결과가 나오도록 설정하였다. 윈도우의 길이로는 FRM의 경우 512, Dual Match의 경우 256으로 설정하였다. Dual Match의 윈도우 길이를 FRM의 1/2로 설정한 것은 참고 문헌 [5]에서 언급한 *최대 윈도우 크기(maximum window size) 이론*을 반영한 것이다. 또한, Dual Match와 FRM의 공정한 성능 비교를 위하여 참고 문헌 [Moo01]에서와 같이 두 기법을 위한 R 트리의 크기가 유사하도록 FRM에서 한 MBR에 속하는 평균 데이타 윈도우 점들의 수를 설정하였다.

그림 1과 그림 2는 각각 FRM 및 Dual Match에 대한 위의 사전 실험 결과를 나타낸 것이다. 가로축은 서브시퀀스 매칭의 처리에 소요되는 각 요소를 나타내며, 세로축은 각 요소에서 소요된 시간을 나타낸다.

---

3) 참고 문헌 [5]의 성능 실험에서와 같이, 일반성을 위하여 DFT 변환의 결과로 나타나는 첫 번째 DFT 계수의 허수 부분인 0 대신, 네 번째 DFT 계수의 실수 부분을 사용한다.

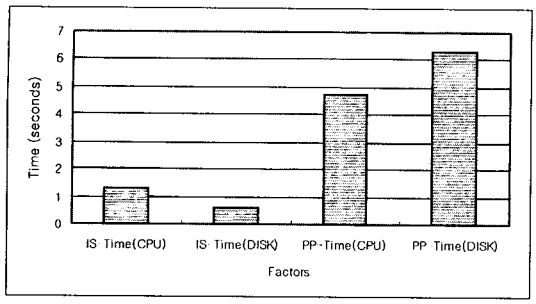

그림 1 FRM의 요소별 소요 시간

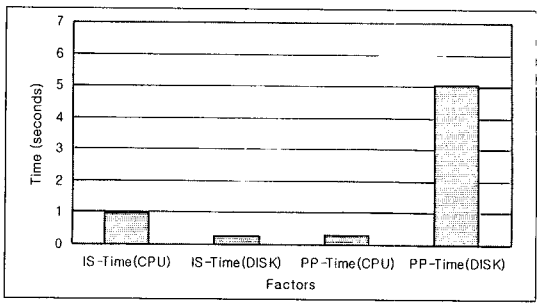

그림 2 Dual-Match의 요소별 소요 시간

먼저, 그림 1에 나타난 FRM의 결과를 보면, 전체 처리 시간 12.9초 중 IS-Time(CPU), IS-Time(DISK), PP-Time(CPU), PP-Time(DISK)에서 소요되는 시간은 각각 1.28초, 0.60초, 4.75초, 6.27초로 나타났다. 즉, 인덱스 검색 단계에서 소요된 시간(IS-Time(CPU)+IS-Time(DISK))은 전체 처리 시간의 15%이며, 후처리 단계에서 소요된 시간(PP-Time(CPU)+PP-Time(DISK))이 나머지 85%를 차지한다. 특히, PP-Time(DISK)가 전체 처리 시간의 50%에 가까운 비중을 보였다.

이러한 결과는 FRM이 R^*-트리의 크기를 줄이기 위하여 점이 아닌 MBR을 R^*-트리에 저장한다는 사실로 설명할 수 있다. 즉, 하나의 MBR은 다수의 데이타 윈도우 점들을 포함하므로 R^*-트리에 저장해야 할 대상의 수가 줄어든다. 따라서 R^*-트리의 크기가 작아지고, 이 결과, 인덱스 검색 시간이 작아진다. 반면, 각 MBR 내의 죽은 공간(dead space)으로 인한 착오 채택들이 많이 발생하므로[5], 이들을 확인하기 위한 후처리 단계의 시간이 크게 증가하는 것이다. 본 실험에서 수행한 서브시퀀스 매칭의 최종 질의 결과의 개수가 15인 반면 FRM의 인덱스 검색 단계에서 반환되는 후보 서브시퀀스의 개수는 556,021로 나타났다.

다음은 그림 2에 나타난 Dual-Match의 결과를 살펴보자. 전체 처리 시간은 FRM의 약 50%인 6.58초이며, 이 중 IS-Time(CPU), IS-Time(DISK), PP-Time(CPU), PP-Time(DISK)에서 소요되는 시간은 각각 0.97초, 0.26초, 0.29초, 5.05초로 나타났다. 인덱스 검색 단계에서 소요된 시간(IS-Time(CPU)+IS-Time(DISK))은 전체 처리 시간의 18%이며, 후처리 단계에서 소요된 시간(PP-Time(CPU)+PP Time(DISK))이 나머지 82%를 차지한다. PP Time(DISK)는 전체 처리 시간의 77%에 가까운 비중을 보였다.

각 시간 요소를 FRM과 비교하면, 인덱스 검색 단계에서 IS-Time(DISK)는 감소한 반면, IS-Time(CPU)는 거의 유사한 것으로 나타났다[4]. Dual-Match에서는 MBR이 아닌 점을 R^*-트리 내의 단말 노드(leaf node)에 직접 저장하므로, 내부 노드(internal node) 내의 MBR이 포함하는 죽은 공간의 크기가 FRM에 비하여 상대적으로 작다. 따라서 Dual-Match에서는 R^*-트리에 대한 범위 질의를 처리할 때 액세스되는 노드의 수가 작으므로 FRM에 비하여 IS-Time(DISK)가 작다.

FRM에서는 내부 노드와 대응되는 MBR의 큰 죽은 공간으로 인하여 질의 영역과 비교해야 할 MBR들의 수가 Dual-Match에 비하여 크게 증가한다. 반면, Dual-Match에서는 인덱스 검색 단계에서 반환된 각 후보 데이타 윈도우 점이 실제로 임의의 질의 윈도우 점과 $\varepsilon/\sqrt{p}$-매치 하는가를 확인하는 *인덱스 수준 여과(index-level filtering)*[5]를 추가적으로 수행한다. Dual-Match에서는 질의 시퀀스로부터 슬라이딩 윈도우들이 추출되므로, 비교해야 하는 질의 윈도우 점들의 수는 매우 많다. 참고로, 본 실험에서는 인덱스 수준의 여과를 위하여 각각의 후보 데이타 윈도우 점과 513(=768-256+1)개의 질의 윈도우 점들간의 유클리드 거리를 계산해야 한다. 이러한 결과, 두 기법의 IS Time(CPU)는 유사한 것으로 나타났다.

Dual-Match의 후처리 단계 처리 시간(PP Time(DISK)+PP Time(CPU))은 FRM의 약 48%로 감소한 것으로 나타났다. 이러한 결과는 R 트리에 MBR을 저장하는 FRM과는 달리 Dual Match에서는 윈도우 점을 저장하기 때문이다. 즉, Dual Match의 인덱스 검색 단계에서 발생하는 *점 여과 효과(point filtering effect)*[5]로 인하여 착오 채택의 수는 FRM에 비하여 현저히

4) 전술한 바와 같이, FRM과 Dual-Match를 위한 두 R^*-트리의 크기가 서로 유사하도록 설정되었음을 유의해야 한다.

줄어든다. 본 실험에서 수행한 서브시퀀스 매칭의 인덱스 검색 단계에서 반환되는 후보의 개수는 FRM가 556,021인 반면, Dual-Match는 30,796개로 나타났다[5]. 이 결과, Dual-Match의 PP-Time(DISK) 및 PP-Time (CPU)가 FRM에 비하여 크게 줄어든다. 요약하면, Dual-Match는 점 여과 효과를 이용함으로써 본 실험에서는 FRM과 비교하여 서브시퀀스 매칭의 전체 처리 시간을 약 50%로 감소시킬 수 있었다.

3.2 서브시퀀스 매칭의 성능 병목 및 해결 방안

제 3.1절의 사전 실험 결과를 보면, Dual-Match가 FRM의 성능을 현저히 개선하였으나, 서브시퀀스 매칭을 위하여 소요되는 시간은 약 5초로서 사용자가 기다리기에는 아직도 지루한 긴 시간으로 나타났다. 이러한 처리 시간은 데이타베이스의 크기가 커질수록 더욱 커지게 되므로 서브시퀀스 매칭의 성능을 추가적으로 개선하기 위한 노력이 필요하다. 본 절에서는 서브시퀀스 매칭의 성능 병목을 지적하고, 이러한 성능 병목을 해결함으로써 효과적으로 전체 서브시퀀스 매칭의 성능을 개선하는 방안에 관하여 논의한다.

서브시퀀스 매칭의 처리 시간 중 후처리 단계에서 소요되는 시간은 FRM과 Dual-Match에서 각각 85%, 77%로 나타났다. 이것은 서브시퀀스 매칭의 성능 병목이 후처리 단계에 있음을 의미한다. 또한, 후처리 단계에서 소요되는 시간을 줄임으로써 전체 서브시퀀스 매칭의 성능을 크게 개선할 수 있음을 의미하는 것이다.

후처리 단계에서는 인덱스 검색 단계에서 반환되는 후보 서브시퀀스들을 디스크로부터 액세스하여 질의 시퀀스와의 실제 유클리드 거리를 계산한다. PP Time (DISK)와 PP-Time(CPU)는 착오 채택의 수에 비례하므로 후처리 단계에서 소요되는 시간을 줄일 수 있는 중요한 전략의 하나는 착오 채택의 수를 줄이는 것이다. FRM 혹은 Dual-Match에서 착오 채택의 수를 추가로 줄이기 위하여 아래의 방법들을 고려할 수 있다.

3.2.1 높은 R^*-트리 차원의 사용

길이 w의 윈도우를 DFT함으로써 생성된 w개의 DFT 계수들 중 R^*-트리 구성에는 앞쪽 f(<<w)개만 사용되고 나머지 (w-f)개는 무시된다. R^*-트리 구성 시 무시되는 정보량이 클수록 착오 채택의 수는 많아진다. 따라서 높은 f값을 사용함으로써 착오 채택의 수를 줄일 수 있다. 그러나, 이는 R^*-트리의 고차원 문제[Web98][1][18]를 새롭게 유발시켜 인덱스 검색 단계의 소요 시간을 크게 증가시킨다는 문제가 있다.

참고 문헌 [24]는 주어진 데이타베이스의 특성을 분석함으로써 인덱스 검색 단계와 후처리 단계 두 곳에서 소요되는 전체 시간을 함께 최소화 할 수 있는 최적의 차원 수를 결정하는 방법을 제안하고 있다. 그러나 이 경우에도 R^*-트리의 고차원 문제로 인하여 f값을 높이는 데에는 한계가 있다. 그러므로 나머지 (w-f)개의 무시되는 정보로 인한 착오 채택의 발생은 불가피하다.

3.2.2 좋은 변환 함수의 사용

좋은 변환 함수란 변환 전 길이 w의 윈도우가 가지는 정보량이 변환 후에 생성되는 계수들 중 일부에 집중되도록 하는 함수를 의미한다. 좋은 변환 함수를 사용하면, 저차원의 R^*-트리를 구성하더라도 무시되는 정보량이 작아지므로 착오 채택의 수를 줄일 수 있다. 참고 문헌 [8]은 이러한 좋은 변환 함수를 이용하고자 하는 연구의 예이며, 현재 웨이블릿(wavelet), DFT, DCT 등이 좋은 변환 함수로 알려져 있다. 그러나 데이타베이스의 분포 특성에 따라 좋은 변환 함수의 판정 기준이 달라지므로 모든 경우에 가장 좋은 변환 함수는 존재하지 않는다. 따라서 좋은 변환 함수를 선정함으로써 착오 채택을 줄이는 데에는 역시 한계가 있다.

3.2.3 큰 윈도우의 사용

FRM 및 Dual-Match를 이용한 서브시퀀스 매칭에서는 R^*-트리에 저장된 윈도우의 길이가 작을수록 착오 채택의 수가 증가하는 경향이 있다. 이러한 경향을 윈도우 크기 효과(window size effect)라 한다[5]. 예를 들어, 두 R^*-트리 R1과 R2에 대하여 R1이 R2보다 큰 윈도우를 이용하여 구성되었다고 가정하자. 이때, R2를 이용한 인덱스 검색 단계를 통하여 반환된 후보 집합 S2 내에는 R1을 이용한 인덱스 검색 단계를 통하여 반환된 후보 집합 S1에는 포함되지 않는 후보 서브시퀀스가 존재할 수 있다. 따라서 큰 윈도우를 사용하여 R^*-트리를 구성하는 것은 착오 채택의 수를 줄이는데 좋은 효과가 있다.

그러나 FRM에서 윈도우 길이가 질의 시퀀스 길이 미만인 경우에는 R^*-트리를 사용할 수 없게 되므로[4], 사용 가능한 윈도우의 길이의 상한은 해당 응용에서 사용할 최소 질의 시퀀스의 길이가 된다. 유사한 이유로 인하여 Dual-Match에서 사용 가능한 윈도우의 길이의

5) FRM과 Dual-Match에서 발생하는 착오 채택의 수의 차이가 이렇게 현저한 반면, 그림 3.1에 나타난 IS-Time(DISK)의 차이는 상대적으로 매우 작다. 그 이유는 다음과 같이 설명될 수 있다. FRM에서는 다수의 윈도우 점들을 묶어 하나의 MBR로 만든 후 R^*-트리에 저장하므로, 동일한 시퀀스로부터 추출된 다수의 후보 서브시퀀스들이 인덱스 검색 단계에서 바로 인접하여 반환된다. 이 결과, 이러한 서브시퀀스들에 대한 후처리 과정에서는 해당 시퀀스를 단 한번만 액세스하면 된다. 반면, Dual-Match에서는 이러한 현상이 발생할 가능성이 FRM과 비교하여 상대적으로 낮다.

상한은 해당 응용에서 사용할 최소 질의 시퀀스의 길이의 1/2이 된다[5]. 실제 응용에서 사용되는 질의 시퀀스의 길이는 매우 다양하므로 이와 같이 윈도우의 길이를 증가함으로써 착오 채택의 수를 줄이는 데에는 한계가 있다.

요약하면, 위의 모든 방법을 최대한 동원하더라도 결국 착오 채택의 발생은 불가피하며, 실제로 발생하는 착오 채택의 수는 최종 질의 결과 수와 비교하여 매우 크다. 착오 채택의 수를 줄이기 위한 좋은 기법으로 평가받는 Dual-Match도 전술한 21개의 최종 질의 결과를 가지는 사전 실험에서 30,781개라는 엄청난 수의 착오 채택을 유발시킨 것을 참고할 필요가 있다.

본 연구에서는 후처리 단계에서 소요되는 시간을 줄일 수 있는 또 하나의 중요한 전략으로서 ***후처리 단계의 수행 방식*** 자체를 개선할 것을 제시한다. 이 전략은 동일한 수의 후보 서브시퀀스들이 반환된 경우라도, 후처리 단계가 효과적으로 수행된다면 처리 성능을 개선시킬 수 있다는 점에서 착안한 것이다. 기존의 기법들[4][5] 등은 주로 위와 같이 착오 채택의 수를 줄이는 방안에 관해서만 연구의 초점을 맞추었으며, 좋은 후처리 단계의 처리 방식을 고안하는 전략은 간과하였다. 본 논문에서는 좋은 후처리 단계의 수행 방식을 제시하고, 이를 통하여 전체 서브시퀀스 매칭의 성능을 크게 향상시키고자 한다.

4. 제안하는 기법

본 장에서는 전체 서브시퀀스 매칭의 성능 병목을 해결할 수 있는 새로운 기법을 제안한다. 제안하는 기법에서 추구하는 궁극적인 목표는 후처리 단계에서 발생하는 디스크 액세스 및 CPU 처리의 불필요한 중복을 완전히 제거하는 것이다. 제 4.1절에서는 FRM과 Dual-Match의 후처리 단계에서 공통적으로 발생하는 문제점을 지적하고, 제 4.2절에서는 이를 해결하는 윈도우 순서 기법을 제시한다. 제 4.3절에서는 제안된 기법이 착오 기각을 발생시키지 않음과 후처리 단계를 처리하기 위한 최적의 방식임을 증명한다.

4.1 기존 기법들의 문제점

그림 3은 인덱스 검색 단계와 후처리 단계로 구성되는 FRM과 Dual-Match의 공통적인 처리 과정을 도면화 한 것이다. 큰 이등변 삼각형은 R*-트리를 나타내며, 아래쪽의 작은 사각형들은 데이타베이스 내에 저장된 시퀀스들을 의미한다.

그림 3에서 점으로 채워진 네 개의 삼각형은 인덱스 검색 단계에서 범위 질의에 의하여 액세스되는 R*-트리의 일부분을 나타낸다. 각 범위 질의의 결과는 그 범위 질의에서 사용된 질의 윈도우 접과의 유클리드 거리가 $\varepsilon/\sqrt{p}$ 이하인 데이타 윈도우 점들의 집합이다. 후처리 단계에서는 먼저 인덱스 검색에서 반환되는 이러한 각 후보 윈도우에 대하여 그 윈도우가 속하는 후보 서브시퀀스를 파악한다. 이 후보 서브시퀀스가 포함되는 데이타 시퀀스를 디스크로부터 액세스함으로써 이 후보 서브시퀀스의 착오 채택 여부를 확인한다. 그림 3에서 각 삼각형과 실선으로 연결된 아래쪽의 각 사각형은 이러한 후보 서브시퀀스를 포함하는 데이타 시퀀스를 나타내며, 이들은 후처리 단계에서 디스크로부터 액세스된다.

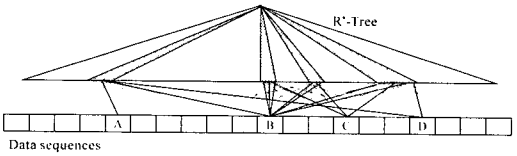

그림 3 서브시퀀스 매칭의 처리 과정

R*-트리 내에 저장된 데이타 윈도우 점들은 동일한 시퀀스로부터 추출되었는지의 여부에 관계없이 모두 독립적으로 관리된다. 또한, 기존의 FRM 및 Dual-Match의 후처리 과정에서는 ***인덱스 검색을 통하여 반환되는 순서***로 각 후보 윈도우가 포함되는 후보 서브시퀀스를 질의 시퀀스와 비교한다[4][5]. 이러한 처리 방식은 다음과 같은 두 가지 측면에서 성능상의 문제들을 야기 시킨다.

4.1.1 디스크 액세스 오버헤드

이것은 동일한 데이타 시퀀스를 디스크로부터 반복적으로 액세스함으로써 발생하는 성능상의 문제이다. 이 문제는 인덱스 검색의 결과, 같은 데이타 시퀀스에 속하는 서로 다른 윈도우들이 후보 윈도우로 선택되는 경우에 발생한다. 즉, 같은 시퀀스에 속하는 후보 윈도우들이라 할 지라도 인덱스 검색의 결과로 반환되는 시점은 서로 다르므로 각각의 처리를 위하여 같은 데이타 시퀀스를 디스크로부터 여러 번 액세스해야 하는 것이다.

그림 3의 예에서 A, B, C, D는 모두 후보 윈도우를 포함하는 후보 시퀀스들이며, 이들은 후처리 단계에서 각각 1, 8, 3, 2회 디스크로부터 액세스되어야 함을 알 수 있다. 물론, 인덱스 검색 단계에서 동일한 시퀀스에 속하는 후보 윈도우들의 반환 시점이 인접한 경우에는 직전에 디스크로부터 액세스하였던 해당 데이타 시퀀스를 추가의 디스크 액세스 없이 그대로 이용할 수 있다. 그러나 그렇지 않은 많은 경우에는 디스크로부터의 동일한 시퀀스의 중복 액세스가 불가피하며, 이러한 현상

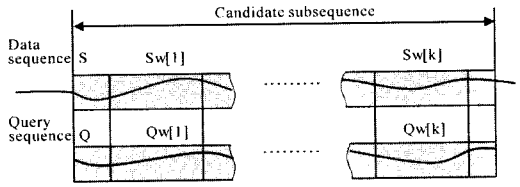

그림 4 질의 시퀀스와 데이타 서브시퀀스의 비교

은 전체 서브시퀀스 매칭의 PP-Time(DISK)를 증가시키는 중요한 원인이 된다.

4.1.2 CPU 처리 오버헤드

이것은 동일한 서브시퀀스를 질의 시퀀스와 두 번 이상 비교함으로써 발생하는 CPU 성능상의 문제이다. 이 문제는 인덱스 검색의 결과, 동일한 데이타 서브시퀀스에 속하는 서로 다른 윈도우들이 후보 윈도우로 선택되는 경우에 발생한다. 그림 4의 경우를 예로 들어보자[6]. FRM 및 Dual Match에서 질의 시퀀스 Q내 Qw[1]에서 Qw[k]까지의 k개의 연속적인 디스조인트 윈도우들과 대응되는 데이타 시퀀스 S내 Sw[1]에서 Sw[k]까지의 디스조인트 윈도우들이 모두 후보 윈도우로 반환되면, 질의 시퀀스는 동일한 이 서브시퀀스와 k번 중복하여 비교함으로써 ε 매치 여부를 확인하게 된다. 이러한 불필요한 중복 확인은 서브시퀀스 매칭을 위한 PP-Time(CPU)의 증가를 초래한다.

4.2 해결 방안

디스크 액세스 및 CPU 처리의 중복 문제가 발생하는 근본적인 원인은 (1) 데이타 윈도우들이 동일한 시퀀스 혹은 서브시퀀스로부터 추출되었는지의 여부에 관계없이 R 트리 내에 독립적으로 저장되며, (2) 후처리 단계에서 인덱스 검색의 결과로 반환되는 순서 그대로 후보 서브시퀀스의 착오 채택 여부를 검사한다는 데에 있다. 본 연구에서는 이러한 문제를 해결하기 위한 간단하면서도 매우 효과적인 윈도우 순서 기법을 제안한다. 제안하는 기법의 기본 전략은 인덱스 검색의 결과로 반환되는 후보 윈도우들의 처리 순서를 조정함으로써 동일한 시퀀스에 속하는 후보 윈도우들, 같은 서브시퀀스에 속하는 후보 윈도우들을 연속적으로 처리한다는 것이다.

이를 위한 구체적인 방법은 알고리즘 5와 같다. 먼저, 단계 1에서는 각 후보 윈도우와 대응되는 후보 서브시퀀스의 식별자 <seqID, subseqOffset>를 인덱스 검색 단계에 의하여 반환되는 순서대로 주기억장치 내에 저장한다. 여기서 seqID는 이 후보 윈도우가 속하는 데이타 시퀀스의 식별자이다. 또한, subseqOffset은 seqID가 의미하는 시퀀스 내의 이 후보 윈도우와 대응되는 후보 서브시퀀스의 시작 위치(offset)이다. 이 값은 이 후보 윈도우와 $\varepsilon/\sqrt{p}$-매치 되는 질의 윈도우의 위치 정보를 이용하여 쉽게 계산할 수 있다.

단계 2에서는 <seqID, subseqOffset>를 정렬 키(sort key)로 사용하여 단계 1에서 저장한 모든 후보 서브시퀀스 식별자들을 정렬한다. 단계 3에서는 각 후보 서브시퀀스를 단계 2에 의하여 정렬된 순서로 액세스하고, 유클리드 거리를 계산함으로써 이것이 질의 시퀀스와 ε-매치 하는가를 실제로 확인한다. 이 과정에서 해당 후보 서브시퀀스 식별자가 직전 루프에서 처리된 것과 동일한 경우에는 이러한 착오 채택 여부의 확인 작업을 수행하지 않는다. 또한, 해당 후보 서브시퀀스가 속하는 시퀀스가 직전 루프에서 처리되지 않은 경우에 한하여 이 시퀀스를 디스크로부터 읽어들인다.

4.3 논의 사항

본 절에서는 보조 정리 1을 통하여 제안된 기법의 견고성(robustness)을 보이고, 보조 정리 2와 3을 통하여 제안된 기법의 효율성(efficiency)을 보인다.

보조 정리 1:

후처리 단계의 처리 방식을 알고리즘 1로 대체한 FRM 및 Dual-Match는 착오 기각을 유발시키지 않는다.

증명:

원래의 FRM과 Dual-Match는 모두 착오 기각을 유발하지 않는다[Fal94][Moo01]. 알고리즘 1은 단지 FRM과 Dual-Match의 후처리 단계를 새롭게 처리하기 위한 것이다. FRM과 Dual-Match에서 사용되던 원래의 후처리 방식과의 차이점은 단지 후보 서브시퀀스의 처리 순서를 재조정한다는 것(단계 2)과 중복된 후보 서브시퀀스를 비교 대상에서 제거한다는 것(단계 3.2)에 있다. 따라서 FRM과 Dual-Match에서 사용되던 원래의 후처리 방식에서 다루는 모든 후보 서브시퀀스들을 알고리즘 1에서도 빠짐없이 다루게 된다. 따라서 제안된 윈도우 순서 기법을 채택하는 새로운 FRM 및 Dual-Match에서 착오 기각은 발생하지 않는다. □

1. **FOR** each *candidate window* returned by the index search step, store the identifier <seqID, subseqOffset> of the *candidate subsequence* corresponding to the candidate window;
2. Sort the identifiers of candidate subsequences by using <seqID, subseqOffset> as a sort key;
3. **FOR** each identifier in the sorted order,
   3.1. **IF** it is exactly the same as the one processed

6) 이 그림은 다른 현상을 설명하기 위하여 참고 문헌 [Moo01]에서 사용하였던 것이다. 본 논문에서는 CPU 처리의 오버헤드를 설명하기 위하여 이 그림을 그대로 인용한다.

in the previous loop, skip it and go to the next loop;

3.2. **ELSE**

IF the seqID is different from the one in the previous loop, access the corresponding subsequence from disk;

IF the candidate subsequence is actually ε-matched with the query sequence, insert it in the final answer;

알고리즘 1 윈도우 순서 기법을 이용한 후처리

보조 정리 2:

알고리즘 1은 후보 서브시퀀스를 포함하는 각 데이타 시퀀스를 디스크로부터 단 한번만 액세스한다.

증명:

단계 2에 의하여 알고리즘 1은 <seqID, subseqOffset> 순서에 따라 각 후보 서브시퀀스가 질의 시퀀스와 ε-매치 하는가를 확인한다. 따라서 같은 데이타 시퀀스에 속하는 후보 서브시퀀스들은 연속적으로 비교된다. 따라서 데이타 시퀀스가 다수의 후보 서브시퀀스들을 포함하는 경우에도 이것은 첫 번째 서브시퀀스를 처리할 때만 디스크로부터 액세스되며, 이후에는 디스크 액세스 없이 처리된다(단계 3.2). 따라서 보조 정리 2는 항상 성립한다. □

보조 정리 3:

알고리즘 1은 두 개 이상의 후보 윈도우들에 의하여 추천되는 각 후보 서브시퀀스를 질의 시퀀스와 단 한번만 비교한다.

증명:

단계 2에 의하여 알고리즘 1은 <seqID, subseqOffset> 순서에 따라 각 후보 서브시퀀스가 질의 시퀀스와 ε-매치 하는가를 확인한다. 또한, 단계 3.1에 의하여 후보 서브시퀀스 식별자가 직전 루프에서 처리된 것과 동일한 경우에는 ε-매치 여부의 확인 없이 바로 무시한다. 따라서 동일한 후보 서브시퀀스에 대한 중복 비교 현상은 발생하지 않는다. □

보조 정리 2와 보조 정리 3에 의하여 알고리즘 4가 후처리 단계에서의 중복을 완전히 제거함을 보였다. 따라서 제안된 윈도우 순서 기법은 서브시퀀스 매칭의 후처리 단계를 위한 최적의 방법이다. 예를 들어, 그림 3에서 후보 시퀀스들을 포함하는 데이타 시퀀스 A, B, C, D는 모두 디스크로부터 단 한 번만 액세스되며, 또한 그림 4에서 k개의 후보 윈도우들은 모두 동일한 <seqID, subseqOffset>을 후보 서브시퀀스 식별자로 가지므로 단 한번만 질의 시퀀스와 비교된다. 물론, 제안된 윈도우 순서 기법은 서브시퀀스 식별자들을 정렬하는 과정(단계 2)을 추가로 요구하지만, 이 오버헤드는 전체 성능 개선 효과를 고려할 때 상대적으로 미미하다. 제 5장에서는 제안된 윈도우 순서 기법의 성능 개선 효과를 정량적으로 검증한다.

5. 성능 평가

본 장에서는 실험에 의한 성능 분석을 통하여 제안하는 기법의 우수성을 규명한다. 먼저, 제 5.1절에서는 성능 평가를 위한 실험 환경을 설명하고, 제 5.2절에서는 기존 기법과의 비교 실험을 통한 제안된 기법의 성능 개선 효과를 제시한다.

5.1 실험 환경

본 연구에서는 성능 분석을 위하여 실제 데이타베이스 K_Stock_Data와 합성 데이타 Syn_Data를 사용하였다. K_Stock_Data는 사전 실험에서 사용하였던 한국의 실제 주식 데이타로서 길이가 1,024인 620개의 데이타 시퀀스로 구성된다. 합성 데이타 Syn_Data내의 각 시퀀스 $S = <s_1, s_2, ..., s_n>$는 다음과 같은 랜덤 워크(random walk) 형태를 가진다[1].

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

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. 제안된 기법의 견고성(robustness)을 규명하기 위하여 각각 5,000개, 10,000개, 15,000개, 20,000개, 25,000개의 길이가 1,000인 데이타 시퀀스들로 구성된 다섯 가지 Syn_Data들과 길이가 각각 1,000, 1,500, 2,000, 2,500, 3,000인 10,000개의 데이타 시퀀스들로 구성된 다섯 가지 Syn_Data들을 생성하였다[7].

질의 시퀀스 Q는 데이타베이스로부터 선택한 시퀀스로부터 길이가 Len(Q)인 임의의 서브시퀀스를 선택하여 각 요소 값에 적절한 범위[8] 내의 임의의 값을 더하는 방식으로 변형하여 생성하였다. 질의 구성 시에는 참고 문헌 [5]와 동일한 방식으로 질의 선택률(query selectivity)[25]를 아래의 식과 같이 정의하고, 각 질의에 대하여 원하는 선택률을 만족하도록 허용치 ε을 조

7) 참고 문헌 [5]의 성능 평가에서는 매우 긴 하나의 데이타 시퀀스로 구성된 데이타베이스를 사용하였다. 그러나 이 경우, 각 데이타 시퀀스를 액세스의 단위로 사용하는 실제 시계열 데이타베이스의 특성을 올바르게 반영하지 못하는 상황이 발생한다. 따라서 본 연구에서는 성능 평가를 위하여 실제 환경과 동일하게 다양한 길이를 가지는 다수의 데이타 시퀀스들로 구성되는 데이타베이스를 사용하였다.

8) 선택된 시퀀스 내에 속하는 요소 값들의 표준 편차를 std라 할 때, 이 범위는 [-std/10, std/10]이다.

정하였다.

선택률 = 데이타베이스 내에서 질의 시퀀스 Q와 ε-매칭하는 모든 서브시퀀스들의 수 / 데이타베이스 내에서 길이가 Len(Q)인 가능한 모든 데이타 서브시퀀스들의 수

성능 평가를 위한 하드웨어 플랫폼은 700MHz Pentium III와 768MB의 주기억장치가 장착된 PC이며, 소프트웨어 플랫폼은 MS Windows 2000 및 Visual C++ 6.0이다. 실험 중 다른 프로세스들과의 상호 간섭을 방지하기 위하여 모든 사용자 프로세스들을 제거한 상황에서 실험하였다. 사용된 데이타 및 인덱스 페이지의 크기는 각각 4KB이다. 데이타 및 인덱스 파일을 액세스 할 때, 버퍼링 효과를 제거한 Windows 2000의 I/O 시스템 콜(system call)을 사용함으로써 매 페이지 액세스 시 실제 디스크 액세스를 보장하였다. 또한, 데이타베이스 내의 시퀀스들을 식별자의 순서대로 실제 디스크 상에 연속적으로 저장하였다.

인덱싱을 위한 자료 구조로서 R^* 트리를 사용하였다. 인덱싱을 위한 특성을 시퀀스로부터 추출하기 위한 변환 함수로는 DFT를 사용하였으며, 각 시퀀스로부터 6개의 특성을 추출하도록 하였다. 첫 번째 DFT 계수에서 항상 0으로 나타나는 허수 부(imaginary part) 대신, 네 번째 DFT 계수의 실수 부(real part)를 특성으로 사용하였다.

제안된 윈도우 순서 기법의 성능 비교 대상은 기존의 인덱스 검색 단계에서 반환하는 순서대로 후처리를 수행하는 기존의 기법이다. 참고 문헌 [5]에서 이미 Dual-Match가 FRM보다 월등하게 우수한 성능을 가지는 것으로 판명되었으므로, 본 성능 평가에서는 제안된 기법과 기존의 기법을 Dual-Match에 적용함으로써 두 기법의 성능을 비교하였다. 각 실험에 대해서 동일한 길이를 갖는 10개의 서로 다른 질의 시퀀스에 대해서 실험한 후 평균을 취한 값을 실험 결과로 하였다.

실험 결과를 평가하기 위한 성능 지수로서 (1) 후처리 단계에서 비교되는 후보 서브시퀀스들의 수, (2) 후처리 단계에서 디스크로부터 액세스되는 시퀀스 액세스 수, (3) PP-Time(CPU), (4) PP-Time(DISK), (5) 인덱스 검색 단계와 후처리 단계에서 소요되는 전체 처리 시간을 측정하였다.

5.2 실험 결과

먼저, 실험 1에서는 K_Stock_Data로부터 추출한 길이 256의 디스조인트 윈도우들을 포함하는 R^* 트리에 대하여 제안된 기법과 기존 기법을 채택하는 서로 다른 두 가지 Dual Match 기반 서브시퀀스 매칭 수행하였다. 사용된 질의 선택률은 각각 10^{-4}(최종 결과 수: 31), 5×10^{-4}(최종 결과 수: 159), 10^{-3}(최종 결과 수: 318)이며, 각 질의 시퀀스의 길이는 512이다.

표 1은 실험 결과를 나타낸 것이다. 먼저, 선택률이 10^{-4}인 경우의 PP-Time(DISK)와 PP-Time(CPU)를 분석한다. PP-Time(DISK)의 경우, 제안된 기법이 기존 기법과 비교하여 30(=5.896/0.195) 배 이상의 성능 개선 효과를 가지는 것으로 나타났다. 표 1에 나타난 바와 같이, 제안된 기법은 실제로 디스크로부터 액세스되는 시퀀스들의 수(# of sequences accessed)를 기존 기법의 약 1/3(=333/938)로 줄였다. 그러나 각 시퀀스를 디스크로부터 랜덤하게 액세스하는 기존의 기법과는 달리, 제안된 기법은 시퀀스 식별자 순서대로 저장된 시퀀스들을 디스크로부터 연속적으로 액세스하는 특성을 갖는다. 이러한 특성으로 제안된 기법은 이 외에도 디스크의 탐색 시간(seek time)을 크게 줄일 수 있으며, 이것이 성능 개선 효과에 반영된 것이다. 이러한 디스크를 랜덤하게 액세스하는 경우와 연속하여 액세스하는 경우의 성능 비는 약 10 배로 보고되고 있으며[19], 본 실험 결과는 이와 상응하는 것이다.

PP-Time(CPU)의 경우, 후보 서브시퀀스 식별자들을 정렬해야 하는 추가의 오버헤드가 있음에도 제안된 기법이 기존 기법과 비교하여 더 나은 성능을 보였다. 이것은 동일한 후보 서브시퀀스를 질의 시퀀스와 중복하여 비교하는 기존의 기법의 문제점을 제안된 기법이 해결하였기 때문이다. 즉, 제안된 기법은 실제로 질의 시퀀스와 비교하는 서브시퀀스들의 수(# of candidates compared)를 기존 기법의 약 15/100(=81,245/96,183)로

표 1 선택률의 변화에 따르는 제안된 기법의 성능 개선 효과

selectivity	# of candidates compared		# of sequences accessed		PP-Time(CPU) (unit: second)		PP-Time(DISK) (unit: second)		Total (unit: second)	
	our method	previous method	our method	previous method	our method	previous method	our method	previous method	our method	previous method
1.0^{-4}	81,245	96,183	333	938	0.733	0.780	0.195	5.896	1.473	7.232
5×10^{-4}	125,710	155,038	364	1,247	1.118	1.228	0.191	7.758	1.914	9.600
1.0^{-3}	143,404	180,871	374	1,338	1.276	1.367	0.184	8.335	2.090	10.330

줄였으며, 이것이 정렬의 오버헤드를 보상한 결과로 나타난 것이다. 이것은 제안된 기법에서 요구하는 정렬 오버헤드가 전체 성능에 부정적인 영향을 미치지 않음을 보여주는 것이다.

전체 후처리 단계 수행 시간(PP-Time(CPU)+PP-Time(DISK))의 경우, 제안된 기법은 기존 기법의 성능을 7.19(=(0.780+5.896)/(0.733+0.195)) 배 개선한 것으로 나타났다. 또한, 전체 서브시퀀스 매칭의 수행 시간(Total)의 경우, 제안된 기법을 후처리 단계에 적용함으로써 얻게 되는 성능 개선 효과는 4.91(=7.232/1.473) 배로 나타났다.

선택률 5×10^{-4} 및 10^{-3}에 대한 실험 결과를 보자. 기존의 기법에서는 선택률이 커짐에 따라 PP-Time(DISK)이 점차 커지는 경향을 보인 반면, 제안된 기법은 PP-Time(DISK)의 변화가 거의 없는 것으로 나타났다. 이것은 선택률이 커짐에 따라 후보 서브시퀀스들의 수가 많아지므로 기존의 기법에서는 액세스해야 하는 시퀀스들의 중복이 더욱 심화됨을 의미하는 것이다. 반면, 제안된 기법에서는 이러한 중복을 피할 수 있으므로 성능의 변화가 거의 없는 것이다. 제안된 기법에서 실제로 액세스해야 하는 시퀀스(# of sequences accessed)의 수는 각각 364와 374로 큰 차이가 없는 것으로 나타났다.

PP-Time(CPU)의 경우, 선택률 10^{-4}의 실험 결과와 그 경향이 거의 동일한 것으로 나타났다. 전체 후처리 단계 수행 시간의 경우, 선택률 5×10^{-4} 및 10^{-3}에 대한 실험에서 제안된 기법은 기존 기법의 성능을 각각 6.86(=(1.228+7.758)/(1.118+0.191)) 배와 6.65(=(1.367+8.335)/(1.276+0.184)) 배 개선한 것으로 나타났다. 서브시퀀스 매칭을 위한 전체 수행 시간(Total)의 경우, 제안된 기법을 사용함으로써 얻게 되는 성능 개선 효과는 선택률 5×10^{-4} 및 10^{-3}에 대한 실험에서 각각 5.02(=9.600/1.914) 배와 4.94(=10.330/2.090) 배로 나타났다.

실험 2에서는 먼저 K_Stock_Data로부터 길이 64, 128, 256의 디스조인트 윈도우들을 각각 추출하여 서로 다른 세 개의 R^*-트리를 구성하였다. 그리고 각 R^*-트리에 대하여 제안된 기법과 기존 기법을 채택하는 서로 다른 두 가지 Dual-Match 기반 서브시퀀스 매칭 수행하였다. 사용된 질의 시퀀스의 길이는 512이며, 질의 선택률은 10^{-4}(최종 결과 수: 31)이다.

표 2는 실험 결과를 나타낸 것이다. PP-Time(DISK)의 경우, 제안된 기법이 기존의 기법과 비교하여 7.34(=1.702/0.232) 배에서 67.10(=13.420/0.200) 배까지 성능이 개선되는 것으로 나타났다. PP-Time(CPU)의 경우, 제안된 기법이 후보 서브시퀀스 식별자들을 정렬해야 하는 추가의 오버헤드가 있음에도 기존의 기법과 비교하여 1.04(=0.279/0.269) 배에서 1.38(=2.033/1.473) 배까지 미미하나마 성능이 개선되는 것으로 나타났다. PP-Time(DISK) 및 PP-Time(CPU) 측면에서 제안된 기법의 성능 개선 효과는 윈도우의 길이가 작아질수록 급격하게 커지는 것으로 나타났다. 그 이유는 윈도우 길이가 작아질수록 인덱스 검색 단계에서 더 많은 후보 서브시퀀스들이 후처리 단계로 반환되며, 이 결과 기존 기법에서 더욱 많이 발생하는 시퀀스 액세스 및 서브시퀀스 비교의 중복을 제안된 기법이 효과적으로 제거하기 때문이다.

전체 후처리 단계 수행 시간의 경우, 제안된 기법은 기존 기법의 성능을 3.91(=(0.259+1.702)/(0.269+0.232)) 배에서 9.24(=(2.033+13.420)/(1.473+0.200)) 배까지 개선한 것으로 나타났다. 또한, 전체 서브시퀀스 매칭의 수행 시간(Total)의 경우, 제안된 기법을 사용함으로써 얻게 되는 성능 개선 효과는 3.05(=2.172/0.711) 배에서 5.24(=17.026/3.249) 배로 나타났다.

실험 3에서는 K_Stock_Data로부터 길이 256의 디스조인트 윈도우들을 추출하여 구성한 R^*-트리에 대하여 제안된 기법과 기존 기법을 채택하는 서로 다른 두 가지 Dual-Match 기반 서브시퀀스 매칭 수행하였다. 사용된 질의 시퀀스의 길이는 256, 512, 768의 세 가지 종류이며, 질의 선택률은 10-4이다. 이때, 각 길이의 질의 시퀀스에 대하여 최종 결과로 반환되는 서브시퀀스들의

표 2 윈도우의 길이 변화에 따르는 제안된 기법의 성능 개선 효과

window size	# of candidates compared		# of sequences accessed		PP-Time(CPU) (unit: second)		PP-Time(DISK) (unit: second)		Total (unit: second)	
	our method	previous method	our method	previous method	our method	previous method	our method	previous method	our method	previous method
64	161,855	258,519	411	2,501	1.473	2.033	0.200	13.420	3.249	17.026
128	81,245	96,183	333	938	0.733	0.780	0.195	5.896	1.473	7.232
256	28,587	28,602	185	277	0.269	0.279	0.232	1.702	0.711	2.172

수는 각각 47, 31, 15개로 나타났다.

표 3은 실험 결과를 나타낸 것이다. PP-Time (DISK)의 경우, 제안된 기법은 기존의 기법과 비교하여 21.63(=4.325/0.200) 배에서 30.20(=6.161/0.204) 배까지의 성능 개선 효과를 보였다. 또한, 이러한 성능 개선 효과는 질의 시퀀스의 길이가 길어질수록 커지는 것으로 나타났다. 이것은 질의 시퀀스와 윈도우의 길이 차가 커질수록 후처리 단계로 반환되는 후보 서브시퀀스들이 많아지므로, 제안된 기법이 동일한 시퀀스를 디스크로부터 중복하여 액세스하는 현상을 제거하는 효과가 더욱 두드러지게 나타나기 때문이다.

PP-Time(CPU)의 결과를 보면, 질의 시퀀스의 길이가 512 및 768인 경우에는 제안된 기법이 기존의 기법과 비교하여 각각 1.06(=0.780/0.733) 배에서 1.11(=0.871/0.784) 배까지 작게나마 성능이 개선되는 것으로 나타났다. 그러나 질의 시퀀스의 길이가 256인 경우에는 정렬의 오버헤드로 인하여 제안된 기법이 기존의 기법과 비교하여 성능이 1.25(=0.283/0.227) 배 떨어지는 것으로 나타났다. 이것은 질의 시퀀스와 윈도우의 길이 차가 없으므로, 기존 기법에서도 서브시퀀스 비교의 중복 현상이 거의 발생하지 않기 때문이다. PP-Time (DISK)의 경우와 동일한 이유로 인하여 PP-Time (CPU)의 성능 개선 효과는 질의 시퀀스의 길이가 길어질수록 커지는 것으로 나타났다.

전체 후처리 단계 수행 시간의 경우, 기존 기법과 비교한 제안된 기법의 성능 개선 효과는 7.12(=(0.871+6.161)/(0.784+0.204)) 배에서 9.42(=(0.227+4.325)/(0.283 +0.200)) 배까지로 나타났다. 또한, 전체 서브시퀀스 매칭의 수행 시간(Total)의 경우, 제안된 기법은 기존 기법과 비교하여 각각 4.41(=7.812/1.772) 배와 5.60(=4.958/0.886) 배의 성능 개선 효과를 보였다.

K_Stock_Data는 비교적 소규모의 데이타베이스이다. 본 성능 평가에서는 제안된 기법이 대규모 데이타베이스에서도 효과적으로 동작함을 규명하기 위하여 다양한 시퀀스의 수 및 시퀀스 길이를 갖도록 생성한 대규모 Syn_Data를 이용한 두 가지 종류의 실험을 추가로 수행하였다.

실험 4에서는 데이타 시퀀스의 길이는 1,000으로 고정시킨 상태에서 데이타 시퀀스들의 수를 5,000, 10,000, 15,000, 20,000, 25,000으로 변화하면서 제안된 기법과 기존 기법을 채택하는 서로 다른 두 가지 Dual-Match 기반 서브시퀀스 매칭의 성능을 비교하였다. 사용된 질의 시퀀스의 길이는 500이며, 질의 선택률은 10^{-5}이다. 또한, 인덱싱을 위하여 사용된 윈도우의 길이는 250이다.

그림 5는 실험 결과를 나타낸 것이다. 가로축은 데이타 시퀀스의 수를 나타내며, 세로축은 서브시퀀스 매칭을 위한 후처리 단계 수행 시간(PP-Total) 및 전체 수행 시간(Total)을 나타낸다. 데이타 시퀀스의 수가 증가함에 따라 두 가지 기법을 위한 후처리 단계 수행 시간은 거의 선형적으로 증가하는 것으로 나타났다. 이것은 동일한 선택률을 갖는 서브시퀀스 매칭을 수행하는 경우, 데이타 시퀀스들의 수가 많은 대규모 데이타베이스에서 최종 결과 서브시퀀스들의 수와 후보 서브시퀀스들의 수가 모두 많이 나타나기 때문이다. 단, 후처리 단계 수행 시간의 증가율에서는 두 기법이 큰 차이를 보였다. 이것은 제안된 기법을 채택하는 경우, 후처리 과정에서 나타나는 시퀀스 액세스의 중복과 서브시퀀스 비교의 중복을 완전히 제거할 수 있기 때문이다. 후처리 단계 수행 시간에 대한 제안된 기법의 성능 개선 효과

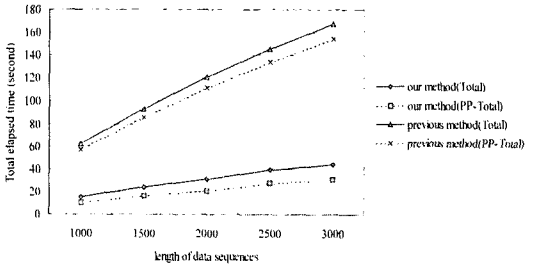

그림 5 데이타 시퀀스 수의 변화에 따르는 제안된 기법의 성능 개선 효과

표 3 질의 시퀀스 길이 변화에 따르는 제안된 기법의 성능 개선 효과

query sequence size	# of candidates compared		# of sequences accessed		PP-Time(CPU) (unit: second)		PP-Time(DISK) (unit: second)		Total (unit: second)	
	our method	previous method	our method	previous method	our method	previous method	our method	previous method	our method	previous method
256	40,219	40,278	284	684	0.283	0.227	0.200	4.325	0.886	4.958
512	81,245	96,183	333	938	0.733	0.780	0.195	5.896	1.473	7.232
768	88,312	106,663	335	986	0.784	0.871	0.204	6.161	1.772	7.812

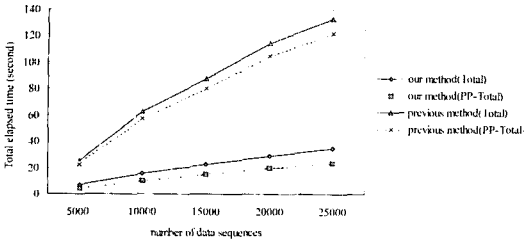

그림 6 데이타 시퀀스 길이의 변화에 따르는 제안된 기법의 성능 개선 효과

는 약 5.14 배에서 약 5.22 배 사이로 나타났다.

그림 5에서 기존 기법을 사용하는 후처리 단계의 수행 시간은 모든 경우에서 전체 수행 시간의 90% 이상인 반면, 제안된 기법을 사용하는 후처리 단계의 수행 시간은 모든 경우에서 전체 수행 시간의 70% 이하인 것으로 나타났다. 이것은 제안된 기법이 서브시퀀스 매칭의 성능 병목을 성공적으로 해결하였음을 보여주는 것이다. 제안된 기법을 통한 성능 병목의 해결은 전체 수행 시간을 약 3.88 배에서 4.21배 줄이는 결과로 나타났다.

실험 5에서는 데이타 시퀀스들의 수는 10,000으로 고정시킨 상태에서 데이타 시퀀스의 길이를 1,000, 1,500, 2,000, 2,500, 3,000으로 변화하면서 제안된 기법과 기존 기법을 채택하는 서로 다른 두 가지 Dual-Match 기반 서브시퀀스 매칭의 성능을 비교하였다. 사용된 질의 시퀀스의 길이, 윈도우의 길이, 그리고 질의 선택률은 각각 500, 250, 10^{-5}로서 실험 4와 동일하게 설정하였다.

그림 6은 실험 결과를 나타낸 것이다. 가로축은 데이타 시퀀스의 길이를 나타내며, 세로축은 서브시퀀스 매칭을 위한 후처리 단계 수행 시간(PP-Total) 및 전체 수행 시간(Total)을 나타낸다. 두 가지 기법을 이용한 후처리 단계 수행 시간과 전체 수행 시간의 전체적인 경향은 실험 4와 거의 동일한 것으로 나타났다. 후처리 단계 수행 시간에 대한 제안된 기법의 성능은 기존의 기법과 비교하여 약 4.97 배에서 약 5.61 배 사이의 향상을 보였으며, 전체 수행 시간에 대한 제안된 기법의 성능은 기존의 기법과 비교하여 약 3.68 배에서 3.81배 사이의 향상을 보였다.

6. 결 론

본 논문에서는 질의 시퀀스와 변화의 패턴이 유사한 서브시퀀스들을 시계열 데이타베이스로부터 찾아내는 서브시퀀스 매칭 문제에 대하여 논의하였다. 본 논문에서는 기존의 서브시퀀스 매칭 처리의 성능 병목을 파악하고, 이를 신속하게 처리하기 위한 간단하면서도 매우 효과적인 윈도우 순서 기법을 제안하였다. 본 논문의 주요 공헌은 다음과 같이 요약된다.

- 전체 서브시퀀스 매칭의 처리 과정에서 소요되는 시간을 IS-Time(CPU), IS-Time(DISK), PP-Time(CPU), PP-Time(DISK)의 네 가지 요소들로 분류하고, 사전 실험을 통하여 전체 처리 시간 중 각 요소가 차지하는 비중을 규명하였다.
- 후처리 단계가 서브시퀀스 매칭의 성능 병목이며, 후처리 단계의 최적화가 FRM 및 Dual-Match 등 기존 기법들에서 간과한 매우 중요한 이슈임을 지적하였다.
- 성능 병목을 해결하기 위하여 후처리 단계를 최적으로 처리할 수 있는 윈도우 순서 기법을 제안하였다. 윈도우 순서 기법은 후보 서브시퀀스들의 질의 시퀀스와의 ε-매치 여부를 판단하는 순서를 조정함으로써 후처리 단계의 처리 성능을 극대화할 수 있는 방법이다.
- 윈도우 순서 기법이 착오 기각을 발생시키지 않음과 후처리 단계를 처리하기 위한 최적의 기법임을 이론적으로 증명하였다.
- 다양한 실험들을 기반으로 하는 성능 평가를 통하여 제안된 기법이 얼마만큼의 성능 개선 효과를 갖는가를 정량적으로 검증하였으며, 실험 결과에서 나타난 다양한 현상들을 상세히 분석하였다.

실험 결과에 의하면, 제안된 기법은 기존의 기법의 성능을 크게 개선하는 것으로 나타났다. 실제 주식 데이타를 사용한 실험 결과, 제안된 기법은 기존 기법의 후처리 단계 수행 시간을 3.91 배에서 9.42 배까지 개선할 수 있었으며, 대규모의 생성 데이타를 이용한 실험 결과에서 제안된 기법은 기존 기법의 후처리 단계 수행 시간을 4.97 배에서 5.61 배까지 향상시킬 수 있었다.

또한, 제안된 기법을 채택함으로써 전체 서브시퀀스 매칭에 소요되는 시간의 90%에 이르던 후처리 단계의 비중을 70%이하로 내릴 수 있었다. 이것은 제안된 기법이 서브시퀀스 매칭의 성능 병목을 성공적으로 해결하였음을 시사하는 것이다. 이 결과, 제안된 기법은 전체 서브시퀀스 매칭의 성능을 실제 주식 데이타를 사용한 실험의 경우 3.05 배에서 5.60 배까지, 대규모의 생성 데이타를 이용한 실험의 경우 3.68 배에서 4.21 배까지의 성능 향상의 결과를 보였다. 이것은 제안된 기법이 성능의 병목인 후처리 단계의 성능을 최적화함으로써 서브시퀀스 매칭의 전체 성능을 현저하게 개선하였음을 보여주는 것이다.

제안된 윈도우 순서 기법은 Dual-Match와 FRM 등 변환을 제공하지 않는 서브시퀀스 매칭뿐만 아니라 스케일링(scaling)[3][9], 시프팅(shifting)[3][9], 정규화(normalization)[20][10][21], 이동 평균(moving average)[7][22], 타임 워핑(time warping)[13][14][15] 등의 다양한 변환을 지원하는 서브시퀀스 매칭 기법들에도 적용이 가능하다. 현재, 제안된 기법을 다양한 변환을 지원하는 서브시퀀스 매칭에 적용하는 방안과 그 성능 개선 효과를 정량적으로 규명하는 문제에 대하여 연구하고 있다.

감사의 글

본 연구에서 성능 평가를 위하여 다양한 실험에 참여한 티맥스소프트(주)의 오세봉 연구원에게 감사를 드립니다.

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, 3rd 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] 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.

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

[7] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] N. Beckmann et al., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In Proc. Int'l. Conf. on Management of Data, ACM SIGMOD, pp. 322-331, May 1990.

[18] S. Berchtold, D. A. Keim, and H. P. Kriegel, "The X tree: An Index Structure for High-Dimensional Data," In Proc Int'l. Conf. on Very Large Data Bases, VLDB, pp. 28 39, 1996.

[19] R. Weber, H. J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity Search Methods in High-Dimensional Spaces," In Proc. Int'l. Conf. on Very Large Data Bases, VLDB, pp. 194 205, 1998.

[20] 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.

[21] 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.
[22] 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.
[23] 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 Intl. Conf. on Information and Knowledge Management, ACM CIKM, pp. 255-262, 2001.
[24] S. W. Kim et al., Optimal Construction of a Multi-dimensional Index for Efficient Similarity Search, 2002. (unpublished manuscript)
[25] P. G. Selinger et al., "Access Path Selection in a Relational Database Management System," In Proc. Int'l. Conf. on Management of Data, ACM SIGMOD, pp.23-34, May 1979.

김 상 욱

1989년 2월 서울대학교 컴퓨터공학과 졸업(학사). 1991년 2월 한국과학기술원 전산학과 졸업(석사). 1994년 2월 한국과학기술원 전산학과 졸업(박사). 1991년 7월~8월 미국 Stanford University, Computer Science Department 방문 연구원. 1994년 2월~1995년 2월 KAIST 정보전자연구소 전문 연구원. 1999년 8월~2000년 8월 미국 IBM T.J. Watson Research Center Post-Doc. 1995년 3월~2000년 8월 강원대학교 컴퓨터정보통신공학부 부교수. 2003년 3월~현재 한양대학교 정보통신대학 정보통신학부 부교수. 관심분야는 데이타베이스 시스템, 데이타 마이닝, 멀티미디어 정보 검색, 공간 데이타베이스/GIS, 실시간 주기억장치 데이타베이스, 트랜잭션 관리