사용자 질의 패턴 분석을 통한 효율적인 음악 검색 시스템의 구현

노 승 민† · 황 인 준††

요 약

디지털 음악 컨텐츠의 활용이 보편화되고 음악 데이터의 양이 방대해지면서 데이터베이스로부터 음악 컨텐츠를 효과적으로 질의하고 검색하는 기능이 절실해졌다. 본 논문에서는 사용자들이 자주 질의하는 멜로디 부분을 이용하여 음악 검색을 보다 빠르고 효과적으로 수행할 수 있는 음악 검색 시스템인 *FMF*(Fast Melody Finder)를 제안한다. 이 기법은 어떤 음악에 대해 사용자가 기억하고 질의하는 내용이 대개 음악의 특정 선율에 한정되어 있다는 사실에 기반하고 있으며, 사용자의 이러한 질의 패턴을 이용하여 자주 질의되는 부분을 인덱스로 사용함으로써 사용자가 원하는 곡을 빠르게 찾아낼 수 있게 해준다. 또한 이러한 선율들을 다루기 위해서, 사용자의 허밍에 의한 음향 질의와 오선지를 통한 질의로부터 추출된 음높이와 음의 길이를 분석하여 UDR과 LSR 스트링으로 변환함으로써 더욱 정확한 질의를 할 수 있다. 프로토타입 시스템을 구현하고 다양한 실험을 통하여 논문에서 제안하는 기법의 우수성을 보인다.

Implementation of an Efficient Music Retrieval System based on the Analysis of User Query Pattern

Seungmin Rho† · Eenjun Hwang††

ABSTRACT

With the popularity of digital music contents, querying and retrieving music contents efficiently from database has become essential. In this paper, we propose a Fast Melody Finder (*FMF*) that can retrieve melodies fast and efficiently from music database using frequently queried tunes. This scheme is based on the observation that users have a tendency to memorize and query a small number of melody segments, and indexing such segments enables fast retrieval. To handle those tunes, *FMF* transcribes all the acoustic and common music notational inputs into a specific string such as UDR and LSR. We have implemented a prototype system and showed on its performance through various experiments.

키워드 : 멀티미디어 데이터베이스(Multimedia Database), 음악(Music), 인덱싱(Indexing), XML

1. 서 론

최근 웹의 급속한 발전으로 인한 웹 컨텐츠의 증가와 네트워크의 고속화 그리고 오디오 및 비디오 압축기술의 발전에 힘입어 멀티미디어 정보의 활용이 일반화 되었다. 멀티미디어 정보 사용의 증가와 함께 멀티미디어 정보를 생성하고 처리하며, 특히 필요한 정보를 효과적으로 검색하는 등의 요구가 증가하게 되었다. 사용자가 원하는 멀티미디어 정보를 보다 효과적으로 찾을 수 있는 검색기능은 VoD(Video on Demand) 서비스나 원격 교육, 전자 도서관, 디지털 방송 등의 많은 응용분야에서 필수적이다. 본 논문에서는 이러한 멀티미디어 정보 중 음악 컨텐츠를 효과적으로 검색할 수 있는 내용 기반 검색 시스템을 제안하고 구현해 보고자 한다.

최근 대부분의 디지털 음악 컨텐츠는 제목, 작곡자, 장르 등에 의해서 목록화[1, 2] 되어있다. 그러나, 이런 방대한 음악 컨텐츠 목록들은 사용자가 음악의 제목이나 작곡자를 정확하게 모르는 경우 거의 검색이 불가능하다. 또한 제목이나 작곡자의 입력오류나 잘못된 장르 분류가 존재할 경우에는 정확한 검색이 어려운 단점이 있다. 그러므로, 이런 문제를 해결하기 위해서는 내용 기반의 음악 검색 기법이 요구된다. 내용 기반 음악 검색 시스템은 음악 컨텐츠를 데이터베이스에 저장하기 전에 각 음악 컨텐츠의 특징들을 추출한 다음, 그 특징들을 가공해서 음악 특징 데이터베이스(Music Feature Database)에 저장하고 이를 데이터베이스의 인덱스로 사용한다. 이때 많이 사용되는 음악 특징으로는 음높이(pitch), 음정(interval), 음색(timbre), 음계(scale)

※ 본 연구는 대학 IT 연구센터 육성·지원사업의 연구결과로 수행되었음.
† 준 회 원 : 아주대학교 정보통신전문대학원
†† 정 회 원 : 아주대학교 정보통신전문대학원 교수
논문접수 : 2003년 7월 14일, 심사완료 : 2003년 12월 11일

등이 있다. 음악 특징 정보를 추출하기 위해서는 시그널 프로세싱(Signal Processing)을 이용하거나 MIDI 파일 안에 들어있는 음악 표기 정보를 이용한다. 시그널 프로세싱을 이용하면 모든 음악 파일 포맷에 적용할 수 있는 반면 구현하기가 힘들고 검색 시간이 길어질 수 있기 때문에, 대부분의 음악 검색 시스템은 MIDI 파일을 이용하여 음악 특징 정보를 추출한다.

음악 데이터베이스에서 질의 멜로디와 유사한 음악 컨텐츠를 찾기 위한 방법으로 음악 특징 중 음향 정보를 이용한다. 멜로디의 음표는 음향 정보 중 음높이 변화를 비교하여 이를 스트링으로 나타내며 다음의 세 가지 값을 가지게 된다. 이전 음표의 음높이와 현재 음표의 음높이를 비교해서 현재 음표의 음높이가 상대적으로 높으면 U(p), 낮으면 D(own), 같으면 R(epeat)로 표기하여 UDR 스트링[3-6]을 생성한다. UDR 스트링은 음악 컨텐츠를 텍스트로 표현한 정보이기 때문에, 필요한 정보를 찾기 위해서는 정보 검색 기법 중 스트링 검색 기법이 필요하다. 특히 사용자의 질의가 그들의 기억에 의존하기 때문에, 휘파람이나 허밍을 이용해서 질의를 생성할 경우 원곡의 UDR 스트링과 조금 다를 수 있다. 따라서 이러한 사용자의 부정확한 질의 멜로디를 고려해서 정확 매칭(Exact Matching) 보다는 유사 매칭(Approximate Matching) 방법을 이용하는 것이 일반적이다[4, 5, 7-9].

본 논문에서는 다양한 음악 특징의 추출과 XML을 이용한 음악 데이터 모델링을 통하여 확장된 음악 검색 시스템인 FMF(Fast Melody Finder)를 제안한다. 자주 질의되는 멜로디 패턴의 위치를 FAI(Frequently Accessed Index)라는 인덱스에 저장해 두고, 사용자의 질의가 들어올 때마다 멜로디 데이터베이스 전체를 검색하는 대신 FAI의 엔트리를 먼저 검색한다. 자주 질의되는 멜로디의 위치를 저장해 두고 사용자의 질의가 들어올 때마다 그 멜로디에 대한 빈도수를 관리하면, 가장 많이 검색되는 멜로디를 알 수 있다. 따라서 사용자가 작성한 질의 멜로디가 FAI 안에 있는 엔트리와 같다면 멜로디 데이터베이스 전체를 검색할 필요없이 바로 결과를 얻어낼 수 있어 검색 속도의 향상을 기대할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 통하여 기존의 음악 정보 검색 (Music Information Retrieval) 시스템에서 사용된 음악 포맷 및 인코딩 방법들에 대해 기술한다. 3장에서는 내용 기반 검색 기법에 사용된 음악 컨텐츠들과 기존 시스템에 구현된 기법에 확장된 기법들과 검색시 사용되는 매칭 알고리즘에 대해 설명한다. 4장에서는 모델링을 통한 음악 데이터의 저장 구조 설계 및 확장된 인덱싱 방식인 FAI를 소개한다. 5장에서는 본 논문에서 구현한 음악 검색 시스템의 구조와 구현 및 실험에 대해서 기술하며 마지막으로 6장에서는 결론과 향후 계획에 대해서 논의한다.

2. 관련 연구

대부분의 시스템에 사용되는 음악 데이터 컬랙션은 MIDI, Humdrum 및 MuseData 형식의 전자 음악 문서로부터 음높이 변화나 음의 길이를 스트링으로 표현한 멜로디들로 구성되어 있다. 최근 이러한 음악 데이터 컬랙션으로부터 관심있는 멜로디를 검색하는 시스템들이 개발되어 왔다. 가장 대표적인 방법으로는 사용자의 휘파람이나 허밍을 이용한 질의를 사용하는 QBH(Query By Humming)[3] 시스템으로, 마이크로 폰을 통해 입력 받은 사용자의 허밍에서 음높이 변화(pitch contour)를 감지하여 UDR 스트링으로 표현하고 음악 데이터베이스의 컨텐츠와 비교 검색하는 시스템이다. Wold[10]는 신호 처리를 이용해서 음의 크기(loudness), 밝기(brightness), 대역폭(bandwidth), 조화도(harmonicity) 등과 같은 음향 특징을 추출하고 이러한 특징을 백터로 표현하여 대상 음악의 음향 특징 정보와 질의 음악의 특징 백터간 거리(vector distance)를 비교하는 방법을 사용하여 음악 컨텐츠를 분류하였다. MELDEX[5]에서는 QBH와 마찬가지로 마이크로 폰을 통한 사용자의 허밍으로부터 추출된 멜로디의 음높이 변화나 음의 길이를 이용하여 UDR 스트링 매칭을 하고, 동적 프로그래밍(Dynamic Programming) 알고리즘을 이용한 유사 검색 기법을 사용하였다. 또한, 흔히 "Tuneserver"라고 알려져 있는 MelodyHound [11] 시스템은 사용자의 휘파람을 통한 입력으로부터 UDR 스트링을 얻어내고 이를 검색에 이용한다.

이와 같은 사용자의 허밍이나 휘파람 등의 질의를 사용하는 QBH 시스템과는 달리, Foote[12]는 유사 매칭을 위하여 일반적으로 사용되는 음높이 변화나 스팩트럼 변화를 이용하지 않고, 트리 기반의 백터 양자기(quantizer)를 이용하여 계산 시간을 줄이려는 시도를 하였다. Themefinder [13, 14]는 16세기의 서양 클래식 음악, 포크송들을 대상으로 Humdrum 명령을 사용하여 웹상에서 사용자가 원하는 곡의 테마를 찾을 수 있는 시스템이다. Blackburn[6]은 기존의 접근 방식과는 다르게, 음악 데이터가 선형(linear)구조가 아닌 구조적이고 분기(branch)하는 구조를 가져서 한 음악에서 다른 음악의 특정 부분으로 링크할 수 있다는 가정하에 멜로디 음색의 변화를 이용해서 멀티미디어 문서 안팎에 있는 MIDI 포맷, 디지털 음악 파일을 대상으로 특정 부분에 접근할 수 있는 내용 기반 항해(Content Based Navigation) 도구를 개발했다.

<표 1>은 최근의 내용 기반 음악 검색 시스템의 간략한 소개를 하고 있다. 각 시스템의 개발년도와 개발시 사용된 음악 데이터의 포맷과 주석처리에 사용된 음악 데이터로부터 얻어낸 음향 특징 정보들에 대한 비교를 하고 있다.

〈표 1〉 기존 음악 정보 검색 시스템

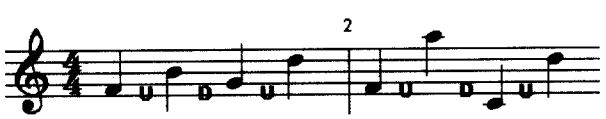
시 스 템 명	개 발 자 / 개 발 년 도	음 악 포 맷	추출되는 특징들
QBH System [3]	Cornell Univ. (Ghias) / 1995	미디 / 단성음악	음조곡선
MELDEX [5, 29]	Univ. of Waikato (McNab, et al) / 1997	미디 / 단성음악	음조곡선, 음정, 음길이
MiDiLiB PROMS [23]	Univ. of Bonn (Clausen) / 1998	미디 / 단성음악	음조곡선
Themefinder [13, 14]	Stanford & Ohio State University (Huron, et al) / 1998	미디, Humdrum / 단성음악	음조곡선, 음정, 음계
CBR and Navigation Music [4]	Univ. of Southampton (Steven G. Blackburn) / 1998	미디 / 단성, 다성음악	음조곡선
Melodic Matching Techniques [7, 8]	Univ. of Australia (Uitdenbogerd & Zobel) / 1999	미디 / 단성음악	음조곡선
MelodyHound [10]	Univ. of Karlsruhe (R. Typke) / 2001	단성음악	음조곡선, 음길이
Fast Melody Finder [20]	Ajou University (S. Rho, et al.) / 2003	미디 / 단성음악	확장된 음조곡선, 음정, 음길이, 음계

3. 내용 기반 분석 및 검색 방법

3.1 내용 기반 분석

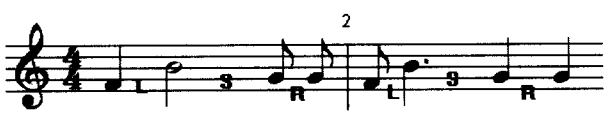
본 논문에서는 음악의 특징 중 음향과 음의 길이 정보를 추출하여 데이터베이스에 저장하여 내용 기반 검색에 이용한다. 대부분의 기존 연구에서는 음향 정보 중 음높이의 변화를 이용한 세가지 타입의 UDR 스트링 비교 기법을 많이 이용하였다. 하지만, 기존의 UDR 스트링은 다음과 같은 문제점을 가진다.

첫째, 전혀 다른 음정을 가지는 음악 파일이나 질의를 동일하게 인식한다는 것이다. (그림 1)에서 보는 것과 같이 첫 번째 소절과 두 번째 소절이 확연한 차이를 보임에도 불구하고 UDR 스트링 기법에만 의존할 경우 두 소절 모두 "UDU" 스트링으로 변환 된다. 이러한 문제를 해결하고자, 본 논문에서는 기존의 UDR 스트링을 세분화하여 검색의 정확도를 높였다. 기존의 Up, Down, Repeat과 두 번째 소절에서 보여지듯이 음정이 급격하게 높아지거나(Up a lot), 낮아지거나(Down a lot) 하는 경우를 추가하여 u, U, d, D, r과 같이 세분화하였다. 이때, 대문자는 음정의 변화가 급격한 경우를 나타낸다. 음정의 급격한 변화의 정도는 각 음표의 수치 값들의 변화 정도가 일정 임계치를 넘게 되는 경우나 이전의 음정의 변화폭보다 더 크게 변화되는 경우로 판단할 수 있다. (그림 1)의 두 소절의 음정 변화를 스트링으로 표기하면 "udu"와 "UDU"로 나타낼 수 있다.



(그림 1) 동일한 UDR 스트링을 가지는 두 소절

둘째, UDR 스트링 표기법은 음 높이의 변화가 아닌 음표가 가지는 길이 정보와 다른 수식 정보들을 표현할 수 없다는 것이다. (그림 2)의 두 소절을 확장된 UDR 스트링 표기법에 따라 표기하면 둘 다 "udr"로 표현되기 때문에 음의 길이(duration)가 서로 다른 멜로디의 구분이 힘들어진다. 이와 같은 경우에는 음의 길이 정보를 이용하여 문제를 해결할 수 있다. 음의 길이 정보는 Longer, Shorter, Repeat의 세 가지 타입으로 구분되며 L, S, R의 스트링으로 표현한다. (그림 2)의 두 소절의 음의 길이 변화를 스트링으로 표기하면 둘 다 "LSR"로 나타낼 수 있다. 하지만, (그림 2)에서 보여지듯이 음정의 변화와 음의 길이를 이용하더라도 두 소절의 정확한 차이를 구분할 수가 없다.



(그림 2) 동일한 LSR 스트링을 가지는 두 소절

그래서 음표의 길이를 좀 더 확장한 방법으로 아래와 같이 음표의 길이만큼 문자열로 표시한다. 각 음표에 해당하는 문자 "도레미파솔라시"는 "CDEFGAB"로 대응되며 음표의 길이에 대한 가중치를 16분음표(semiquaver), 8분음표(quaver), 4분음표(crotchet), 2분음표(minim), 온음표(semibreve)에 각각 "1", "2", "4", "8", "16"으로 설정한다. 점음표(dot note)의 경우에는 문자열 뒤에 "."을 추가하고 sharp과 flat은 각각 문자열 뒤에 "#", "-"을 추가한다. (그림 2)의 8개 음표에 대한 멜로디를 표기하면 아래와 같다.

F4 | B8 | G2 | G2 | F2 | B4. | G4 | G4

위에서 언급한 방식의 스트링은 음악 컨텐츠를 텍스트로 표현한 정보이므로 필요한 정보를 찾기 위해서는 서브스트링 검색(Substring Matching) 기법이 필요하게 된다. 이때 사용자의 질의가 그들의 기억에 의존하고 휘파람이나, 허밍을 이용하여 질의를 생성할 경우 원곡의 스트링과 조금 다를 수 있다. 예를 들어, 위의 스트링을 질의 멜로디라고 가정하고 데이터베이스내에 저장되어 있는 멜로디가 "F4|C4|G4|G4|F2|C4.|G4|G4"과 같다고 할 때 두 멜로디는 정확 매칭 방법으로는 일치하지 않는다.

따라서 본 논문에서는 이러한 사용자의 부정확한 질의 멜로디를 고려하여 정확 매칭 기법보다는 유사 매칭 기법을 이용하며 음악 데이터베이스의 컨텐츠 수의 증가로 인한 시스템 성능 저하를 고려하여 정확 매칭 기법을 병행하여 이용한다. 정확 매칭 기법은 Boyer-Moore 알고리즘[15]을 사용하였고 유사 매칭 기법은 동적 프로그래밍과 LCS(Longest Common Subsequence)[15] 기법을 사용하였다. 대부분의 QBH 시스템에서 사용되고 있는 동적 프로그래밍은 주어진 질의와 유사한 모든 멜로디를 찾는 기법으로 이들 간의 유사도는 거리 함수를 이용하여 측정한다. 예를 들어, 두 개의 멜로디의 길이가 같은 경우 해당 문자가 서로 다른 위치의 수로 이들 사이의 거리를 정의할 수 있다. 멜로디의 길이가 다른 경우에는 편집 거리(edit distance) 함수를 이용하게 된다. 편집 거리는 멜로디의 길이가 동일하게 되도록 임의의 한 문자열에 대해 수행해야 하는 삽입, 삭제, 치환 연산이 필요한 문자의 최소 수로 정의된다. 예를 들어, "ddudrdu"와 "ddurru" 사이의 편집 거리는 삽입과 삭제 연산을 통해서 아래와 같이 문자열의 길이를 동일하게 일치시킴으로써 구할 수 있으며 편집거리는 '2'이다. 또한 이들 두 문자열의 LCS는 "dduru"이다.

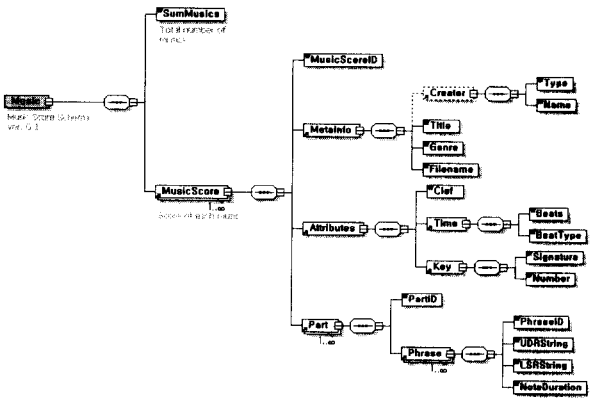
Melody in FAI : [d d u d r d u]
Query melody : [d d u _ r _ u]

4. 음악 정보 저장 공간을 위한 스키마 및 인덱싱

4.1 음악 스키마

기존의 음악을 디지털화하는 기술은 음악 정보의 복잡성 때문에 여러 가지 다양한 포맷들로 표현되어 왔다. 대표적인 포맷으로는 NIFF, SMDL, MP3, MIDI 등이 있다. NIFF (Notation Interchange File Format)는 XML과 같은 텍스트 포맷이 아닌 매우 복잡한 그래픽 형태의 바이너리 포맷으로 되어 있어 프로그램 및 디버깅이 용이하지 않다. SGML (Standard Generalized Markup Language)에 기초한 SMDL (Standard Music Description Language)은 좀 더 일반적인 목적에의 음악 표기를 위해 디자인 되어 복잡하기 때문에 대부분의 상용 소프트웨어에서 지원하지 않는다.

MP3와 그 외의 다른 음악 포맷들은 대부분 리코딩 정보를 표현하고 있으며, 매우 간단한 음악을 제외하고는 대부분의 음악 포맷들로부터 자동적으로 정확한 음악 표기 정보를 가져오기가 쉽지 않다. 반면에 MIDI는 전자 음악 산업에 많은 영향을 끼친 포맷으로서, 현재 음악 교환 포맷(Music Interchange Format)으로 가장 많이 쓰이고 있으며 다른 음악 포맷에 비해 많은 음악 표기 정보 및 곡의 타이틀, 작곡가, 트랙명 이나 다른 설명 정보를 가지고 있다. 따라서 본 논문에서는 MIDI를 사용하여 음악 데이터베이스를 구축하였다.



(그림 3) 음악 메타데이터를 위한 XML 스키마 다이어그램

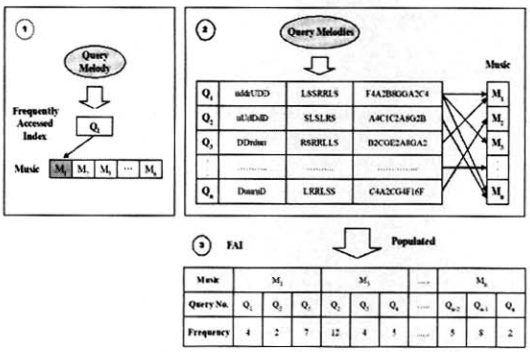
최근 인터넷에서의 문서 또는 데이터를 표현하기 위한 표준으로 많이 쓰이고 있는 XML을 이용하여 만들어진 MusicXML[16], ScoreML과 같은 포맷은 악보를 XML로 표현한 것으로 음정, 박자, 조성, 리듬, 화음 등의 모든 정보를 표현한 마크업 언어이다. 본 논문에서는 음계, 음자리표, 박자 정보 등 악보의 속성 정보와 추출된 음정 및 음의 길이 정보 등을 추가한 새로운 데이터 모델을 제안한다. (그림 3)은 음악 메타 정보를 XML 스키마를 이용하여 표현한 다이어그램이다. 각 음악 파일은 하나의 *MusicScore* 앨리먼트를 가지며 제목, 장르, 작곡자에 대한 정보를 표현한 *MetaInfo* 앨리먼트, 음계, 음자리표, 박자를 표현한 *Attributes* 앨리먼트, 변환된 음정 및 음의 길이 등에 대한 정보를 스트링을 표현한 *Part* 앨리먼트를 하위 앨리먼트로 가진다.

4.2 인덱싱 메커니즘

(그림 4)는 본 논문에서 제안하는 FAI 기반 인덱싱 메커니즘을 개괄적으로 보여준다. ①은 FAI의 생성 단계를 보여주며, FAI 생성 초기에는 FAI내에 엔트리가 없이 비어있는 상태로 존재하게 된다. 사용자의 질의 멜로디가 들어오면 해당 멜로디를 음악 데이터베이스에서 검색하고 부합하는 음악 컨텐츠가 있을 경우, FAI 엔트리에 음악 컨텐츠의 링크와 함께 질의 멜로디가 들어갈 공간을 하나 할당해 준다. 이 공간에는 질의 멜로디에 대한 정보뿐만 아니라 접속 빈도수와 음악 컨텐츠내의 질의 멜로디 반복수에 대한 정보 역시 포함되어 있다. 이후에 다른 질의 멜로디가 음악 컨텐츠와 매치하면 새로운 FAI 엔트리 공간을 할당해 준다. ②에서 보여주듯이, 초기의 FAI 엔트리에는 이러한 방식으로 음악 컨텐츠와 매칭되는 모든 질의 멜로디를 넣어주게 되며, ③은 이렇게 채워진 FAI의 엔트리들을 보여주고 있다.

사용자의 질의 회수가 증가하면 FAI 엔트리의 수도 증가하고 많은 정보가 축적하게 된다. 많은 사용자들로부터 질의를 받으면 한 노래에 대한 질의가 여러 번 있을 수 있고, 그 노래를 가리키는 FAI 엔트리의 개수가 여럿이 될 수도 있다. 이때는 질의 멜로디를 무조건 FAI 엔트리에 넣는 것

이 아니라, 기존의 FAI 엔트리와 비교해서 질의 멜로디와 중복되는 것은 엔트리에 추가하지 않는다.



(그림 4) FAI 기반의 인덱싱 메커니즘

이렇게 사용자의 질의 멜로디가 점차 증가하고 FAI 인덱스의 엔트리가 많아지게 되면 다음과 같은 문제점이 발생할 수 있다.

- Overlap : FAI 엔트리 멜로디와 질의 멜로디가 겹치는 경우
- Inclusion : FAI 엔트리 멜로디와 질의 멜로디 중 어느 한 쪽이 다른 멜로디를 포함하는 경우
- Partially Overlapped : FAI 엔트리 멜로디와 질의 멜로디의 일부분만 겹쳐지는 경우
- Partially Included : FAI 엔트리 멜로디와 질의 멜로디 중 일부분만 어느 한 쪽이 다른 멜로디를 포함하는 경우

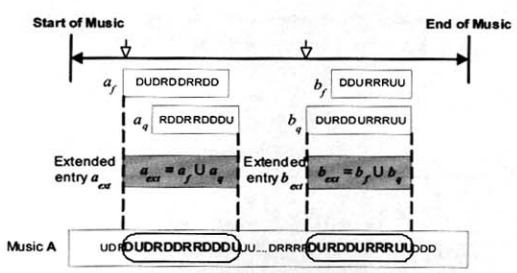
위와 같은 경우에 음악 매칭 엔진은 두 멜로디가 같거나 유사함에도 불구하고, 각 멜로디의 시작 또는 종료 위치가 약간 다르다는 이유로 매칭되지 않는다는 결과를 보여주게 된다. 따라서, 본 시스템은 FAI 엔트리의 확장 및 수정 방법을 통하여 이를 해결하였다. (그림 5)는 위의 4가지 문제점과 FAI 엔트리의 확장 및 수정을 통한 해결방안을 보여주고 있다. (그림 5)에서 a_f와 b_f는 A곡에 대한 FAI 엔트리 멜로디이고, a_q와 b_q는 사용자가 작성한 질의 멜로디이다. (그림 5)(a)에서 a_f와 a_q는 중첩의 관계이고 b_q 는 b_f 를 포함하는 관계를 보여주며 (그림 5)(b)에서는 부분적으로 겹치거나 포함하는 관계를 나타내고 있으며 이는 다음의 조건식을 만족할 때 항상 유효하다.

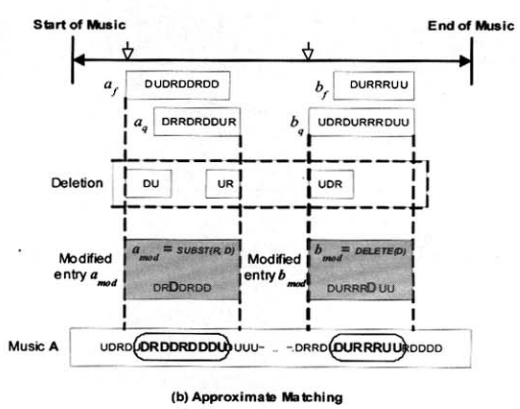
Exact Matching (a) : iff a_q/a_f is a **substring** of a_f/a_q
Approximate Matching (b) : iff a_q/a_f is a **subsequence** of a_f/a_q

(그림 5)(a)에서 보여지듯이 질의 a_q와 b_q는 각각 a_f와 b_f 멜로디를 찾으려는 의도임에도 불구하고 두 멜로디의 시작 및 종료 위치가 다른 이유 때문에, 음악 매칭 엔진은 FAI 내의 엔트리에서 찾지 못하고 모든 데이터베이스내의 컨텐츠를 검색하게 되는 것이다. 따라서, 음악 데이터베이스 전체의 검색으로 찾은 질의 멜로디의 위치가 기존 FAI 엔트리 멜로디의 위치와 비슷할 경우 질의 멜로디와 FAI 엔트리를 합쳐서 확장한다. 결과적으로 a_f와 a_q는 a_{ext}로 b_f와 b_q는 b_{ext}로 확장된다. (그림 5)(b)에서는 (그림 5)(a)와는 다르게 a_f와 a_q가 부분적으로 겹치고 b_q가 b_f를 부분적으로 포함하기 때문에 두 멜로디의 관계가 서브스트링의 관계가 아닌 서브시퀀스의 관계이다. 따라서 유사 매칭 기법을 사용하여 멜로디내의 일치하지 않는 부분을 삭제 및 치환 작업을 통한 멜로디의 수정 및 확장을 하게 된다. 우선 일치하지 않는 멜로디의 불필요한 부분을 삭제하고 첫번째로 일치하는 문자에 대해서 시작 위치 조정을 하게 된다. 조정된 두 멜로디 $a_f(b_f)$와 $a_q(b_q)$는 다음과 같다.

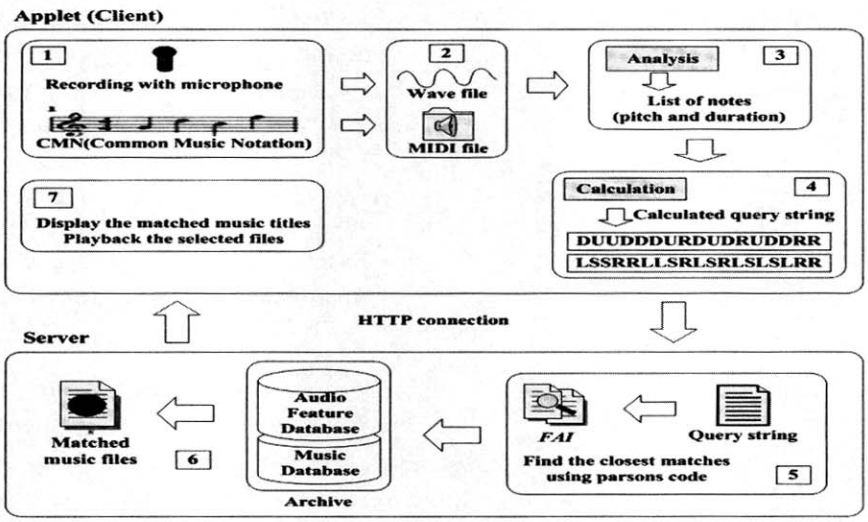
a_f : D R D D R D D b_f : D U R R R U U
a_q : D R R D R D D b_q : D U R R R D U U

A곡의 두 멜로디의 경우는 우선 a_q의 문자 R과 a_f의 문자 D가 일치하지 않기 때문에 치환 작업이 이루어져야 하





(그림 5) FAI 엔트리의 확장 및 수정



(그림 6) 음악 검색 시스템 구조와 질의 처리 과정

며, B곡의 경우에는 b_q의 문자 D가 불필요한 경우이므로 삭제 되어야 한다. 따라서 a_f와 a_q는 a_{mod}로 b_f와 b_q는 b_{mod}로 수정된다.

5. 음악 검색 시스템

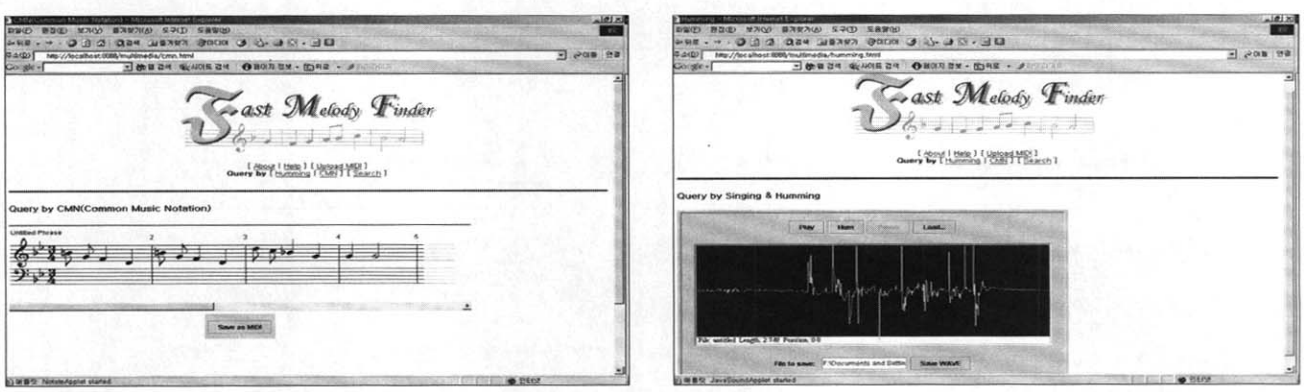
5.1 구 현

본 논문에서 제안하는 내용 기반 음악 검색 시스템에 대해서 설명한다. 클라이언트 인터페이스는 웹 기반으로 자바 애플릿과 JSP로 구현하였으며, 서버는 Java Sound와 MIDI 파일 처리를 위해서 jMusic[17] 라이브러리를 이용해서 음향 특징 정보를 추출했다. 음악 파일은 인터넷으로부터 약 12,000개의 MIDI 파일을 수집하여 실험에 이용하였다.

(그림 6)은 *FMF* 시스템의 개략적인 구조와 질의의 흐름을 순서대로 나열하였다. 클라이언트는 웹을 통해서 질의를 입력하고 검색된 음악을 재생한다. 서버는 사용자로부터 받은 질의에 대해서 우선적으로 FAI에 저장되어 있는 멜로디를 검색한 뒤 일치되는 멜로디가 없는 경우 음악 아카이브의 멜로디를 검색하게 된다. 질의는 마이크를 이용한 허밍과 오선지에 멜로디를 직접 입력할 수 있는 CMN(Common Music Notation) 인터페이스를 통해 웨이브 파일과 미디 파일로 변환된다.

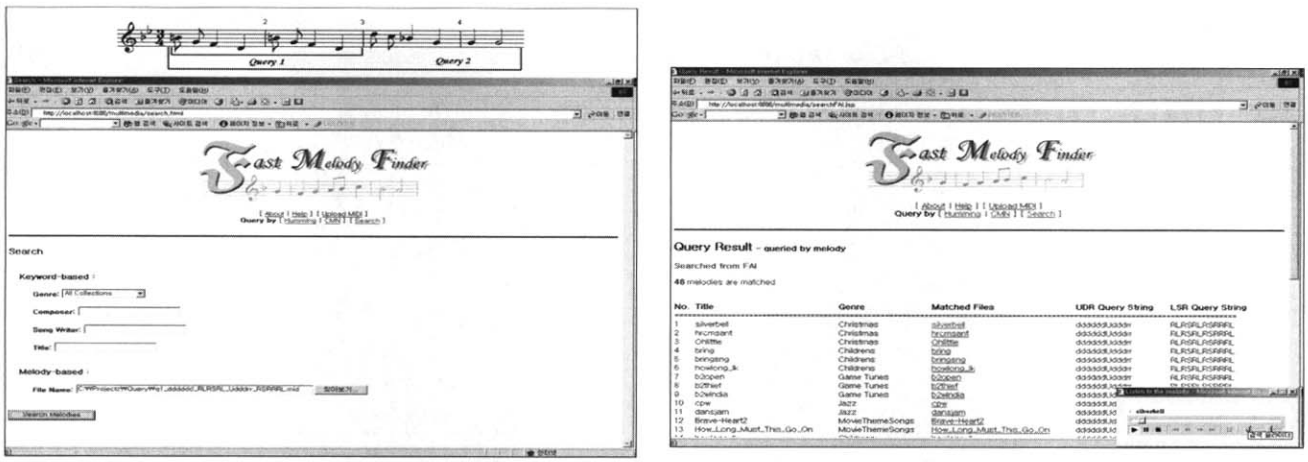
5.2 질의 및 브라우징 인터페이스

(그림 7)은 CMN과 허밍에 의한 질의 인터페이스를 보여준다. CMN 인터페이스는 사용자가 드래그 앤 드랍을 통하여 원하는 음표를 오선지 위에 그리는 것이며 음표의 길이 또한 지정할 수 있다. 허밍 인터페이스는 마이크를 통하여 약간의 흥얼거림이나 노래를 입력 받고 이를 웨이브 파일로 저장한다. 이렇게 저장된 웨이브 파일은 다시 미디파일로 변환된 후에 3장에서 설명한 UDR, LSR 등의 스트링으로 변환되어 음악 특징 데이터베이스에 저장된다.



CMN Interface　　Humming Interface

(그림 7) 질의 인터페이스



Search Interface　　Result Interface & MIDI Player

(그림 8) 검색 및 결과 인터페이스

또한 변환된 질의들은 음의 높이와 길이 분석을 통해서 스트링으로 변환되며, 변환된 질의 스트링과 FAI 내의 멜로디 스트링과 유사 비교를 통해서 질의와 일치하는 음악을 검색한다. 이때, FAI 내에 일치하는 멜로디가 없으면 음악 특징 데이터베이스내의 멜로디를 순차적으로 검색하게 된다.

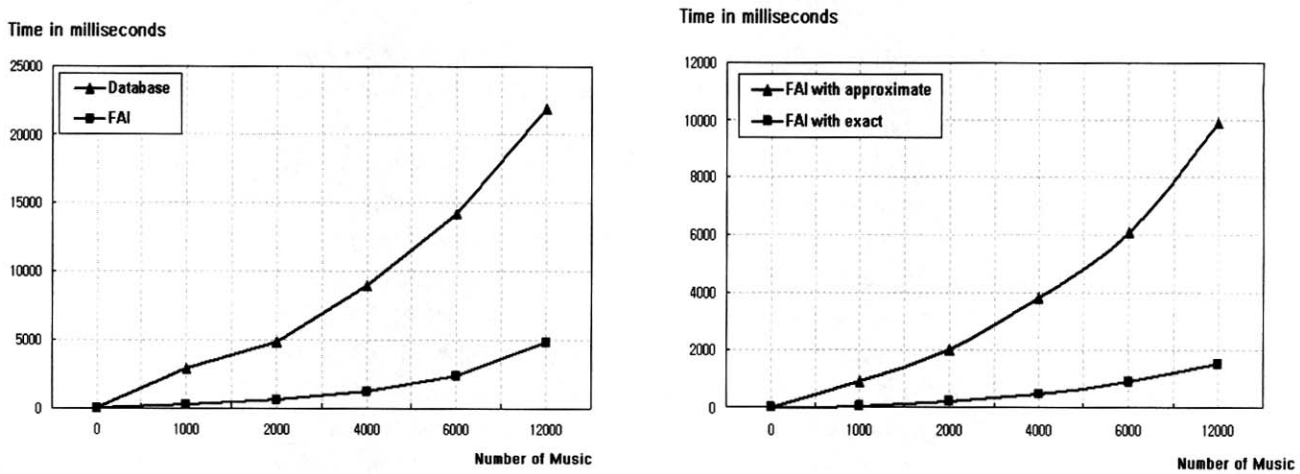
(그림 7)의 허밍과 오선지 등의 인터페이스를 통해 입력받은 질의 멜로디로부터 정확 매칭과 유사 매칭을 통하여 FAI내의 멜로디 스트링과 비교를 하는 과정을 표현한 알고리즘과 정확 및 유사 매칭 함수에 대한 슈도코드는 [부록 1]과 [부록 2]에 나타나있다.

(그림 8)은 "ddduddddUdddrr"와 "RLRSRLRSRLRRL" 스트링을 가지는 질의 멜로디에 대한 검색 결과 화면이며, 검색된 음악 컨텐츠의 재생은 해당 파일의 링크를 누르면 결과 화면 우측하단의 미디 플레이어가 실행되며 진행된다.

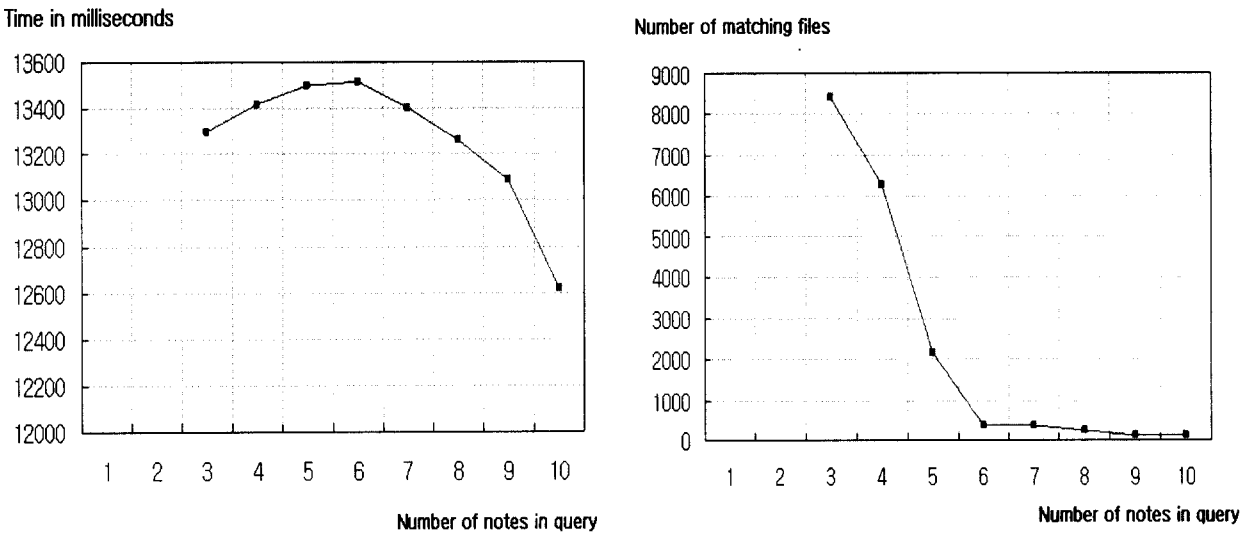
5.3 실 험

실험은 정확 매칭 방법에서 많이 쓰이고 있는 Naive와 KMP(Knuth-Morris-Pratt) 그리고 BM(Boyer-Moore) 알고리즘을 사용하였으며, 유사 매칭을 위해서 동적 프로그래밍 방법을 사용하여 성능 분석을 하였다. 본 논문에서는 세 개의 정확 매칭 알고리즘 중 가장 빠른 Boyer-Moore 알고리즘을 사용하여 검색 성능을 높였다.

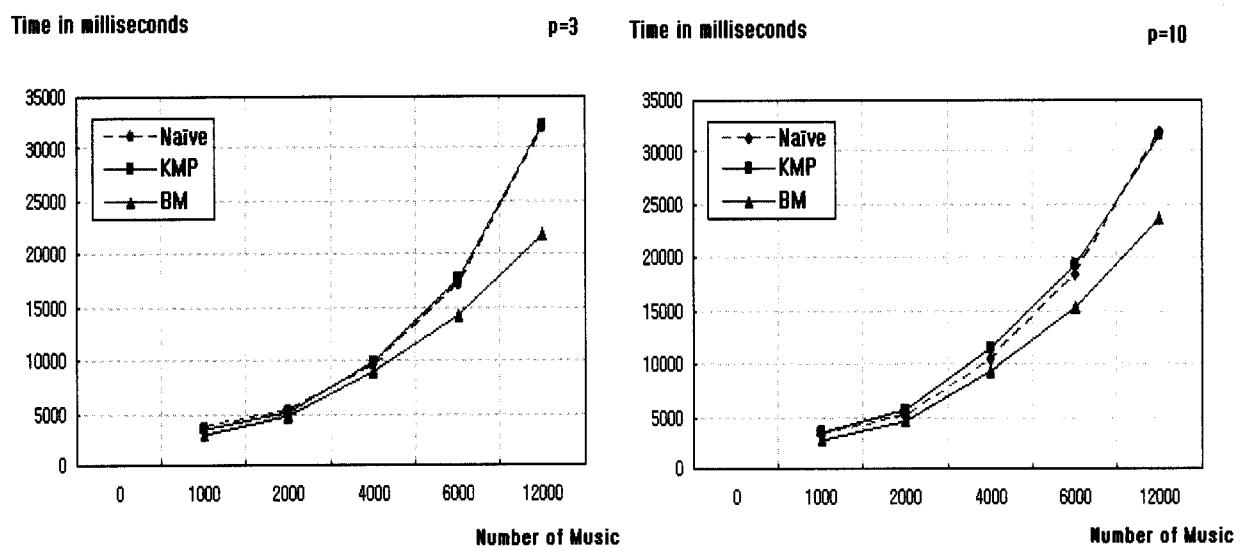
본 실험에서는 음악 데이터베이스의 컨텐츠 수를 증가시키면서 질의 처리 시간을 측정했다. (그림 9)의 첫번째 그래프는 FAI 기반과 데이터베이스 기반의 평균 질의 처리 시간을 보여준다. 기울기가 가파른 선이 일반적인 음악 검색의 평균 질의 처리 시간을 나타내고, 기울기가 비교적 완만한 선이 FAI를 이용한 음악 검색 질의 처리 시간을 나타낸다. 두 실험 모두 컨텐츠의 수가 증가함에 따라 처리 시간도 같이 증가하였지만 FAI 기반의 검색이



(그림 9) FAI 기반 검색의 성능 분석



(그림 10) 질의 패턴 길이에 따른 질의 처리 시간 및 검색 결과



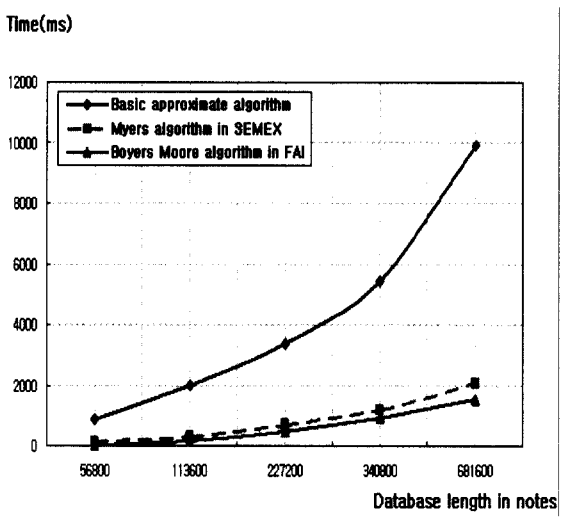
(그림 11) Naive, KMP, BM 알고리즘을 이용한 질의 응답 시간

모든 데이터베이스를 검색하는 방법보다 우월함을 알 수 있었고, 음악 컨텐츠의 수가 증가할수록 FAI 기반 검색이 얻는 이득은 증가하는 것으로 나타났다. (그림 9)의 두번째 그래프는 FAI 기반의 검색에서 정확 매칭과 유사 매칭을 사용할 경우 소요된 평균 질의 처리 시간을 비교하고 있다.

(그림 10)은 사용자의 질의 멜로디 패턴의 길이에 따른 질의 처리 시간 및 검색된 파일의 개수를 보여준다. 첫번째 그래프는 질의 멜로디의 음표의 수가 질의 처리 시간에 큰 영향을 끼치지 않음을 나타낸다. 하지만 좀 더 크고 다양한 음악 컬렉션에서의 질의 멜로디 패턴의 길이는 중요한 요소가 될 수 있다. 두번째 그래프에서는 질의 멜로디 패턴의 길이가 커질수록 검색되는 파일의 수가 적어짐을 알 수 있다. 이는 질의 멜로디의 음표의 수가 적을수록 질의 멜로디의 패턴이 데이터베이스내의 더 많은 멜로디에 나타나는 것을 나타낸다.

검색 성능을 비교하기 위해 음악 컨텐츠의 수를 증가시켜가면서 Naive, KMP 그리고 BM 알고리즘을 사용하여 각각의 질의 처리 시간을 측정하였다. (그림 11)은 사용자 질의에 대한 각 알고리즘별 평균 질의 처리 시간을 보여준다. 첫번째 그래프는 사용자의 질의 멜로디 패턴의 길이가 "3"인 경우이며, 두번째 그래프는 "10"인 경우를 나타내며 각각의 알고리즘에 대한 질의 처리 시간은 10번의 질의에 대한 평균값을 적용하였다. 두 경우 모두 BM 알고리즘이 KMP나 Naive 알고리즘보다 짧은 처리시간을 가지는 것을 알 수 있다.

(그림 12)는 기존 SEMEX[18] 시스템에서 제안한 Myers' bit-parallel 알고리즘과 Dynamic programming 알고리즘을 이용한 유사 매칭 기법 및 본 논문에서 사용된 Boyer-Moore 알고리즘을 이용한 세 가지 경우에 대해서 실험을 하였고 그 결과를 보여준다. 이때, 데이터베이스내 멜로디들의 평균 음표수는 56.8개이며 12,000 여개의 데이터베이스내 모든 음표수는 681,600개이다. (그림 12)에서 알 수 있듯이 본 논문에서 제안한 FAI 기반의 시스템이 Myers' bit-parallel 알고리즘과 Dynamic programming 알고리즘을 이용한 시스템에 비해 성능이 우수함을 알 수 있다.



(그림 12) SEMEX 시스템과 FAI 기반 검색의 성능 분석

<표 2>는 (그림 8)의 질의 인터페이스에서 크리스마스 음악인 "실버벨"의 처음 두 소절을 질의로 했을 때의 처리 결과를 보여준다. 표를 보면 알 수 있듯이, 질의 1은 첫 번째 소절을 나타내고 질의 2는 첫 번째와 두 번째 소절을 조합한 질의이다. 질의 1에서 음의 높이나 길이의 변화만을 이용한 경우보다 음의 높이와 길이를 둘 다 고려한 경우의 정확도가 증가함을 알 수 있었고, 질의의 길이 역시 정확도에 영향을 끼치는 것을 알 수 있었다. 데이터베이스 내에서 질의 1과 질의 2에 정확히 부합하는 멜로디는 각각 132개, 18개로 질의 2가 질의 1에 비해서 보다 정확하고 효율적임을 알 수 있다.

6. 결론 및 향후 계획

본 논문에서는 음악 컨텐츠의 다양한 특징 추출을 통해 사용자가 자주 질의하는 부분을 이용한 효과적인 내용 기반 음악 검색 시스템을 제안하였다. 사용자의 질의 패턴을 분석하여 자주 검색되는 멜로디의 위치를 FAI라는 새로운 개념의 인덱스에 저장하여 데이터베이스내의 모든 멜로디를 검색하지 않고 FAI를 우선적으로 검색하는 방법으로 검색 성능을 향상시켰다. 이는 선형적으로 데이터베이스내의 모든 음악을 검색하는 기존의 방법보다 FAI를 기반한 검색이 훨씬 빠르게 질의를 처리할 수 있음을 보여주었다. 향후에는 다성음악(polyphonic) 컨텐츠로부터 특정한 악기의 채널을 제거한 뒤 단성음악이 가지는 멜로디와 유사한 멜로디 생성 후 이를 검색에 이용하는 방법에 대한 연구가 이루어져야 한다. 이와 함께 제안된 FAI의 성능을 좌우하는 질의 멜로디의 발생 빈도와 FAI 내의 엔트리 개수와의 성능 관계 분석을 통한 최적의 성능을 제공하는 엔트리 개수와 알고리즘을 찾는 것이다.

참 고 문 헌

[1] 푸키, http://www.puckii.com.

[2] 야후, http://launch.yahoo.com.

[3] A. Ghias, et al., "Query by humming - musical information retrieval in an audio database," In : Proceedings of ACM Multimedia 95 - Electronic Proceedings, pp.231-236, 1995.

〈표 2〉 질의 결과

	Pitch	Time	Pitch+Time	Relevant	Retrieved		Precision	
					Exact	Appro.	Exact	Appro.
Query 1	ddduddd	-	-	132	545	2863	24.2	4.61
	-	RLRSRLR	-		337	2205	39.2	5.98
	-	-	ddduddd + RLRSRLR		132	389	100	33.9
Query 2	Query1 + Udddrr	-	-	18	61	337	29.5	5.34
	-	Query1 + SRLRRL	-		49	189	36.7	9.52
	-	-	Query1 + Udddrr + SRLRRL		18	48	100	37.5

[4] N. Kosugi, et al., "A Practical Query-By-Humming System for a Large Music Database," In : Proceedings of the 8th ACM International Conference, pp.333-342, 2000.

[5] R. J. McNab, et al., "The New Zealand digital library ME Lody index," Digital Libraries Magazine, 1997.

[6] S. Blackburn, D. DeRoure, "A Tool for Content Based Navigation of Music," In : Proceedings of ACM multimedia 98 - Electronic Proceedings, pp.361-368, 1998.

[7] A. Uitdenbogerd and J. Zobel, "Manipulation of music for melody matching," In : Proceedings of ACM Multimedia Conference, pp.235-240, 1998.

[8] A. Uitdenbogerd and J. Zobel, "Melodic matching techniques for large music databases," In : Proceedings of ACM Multimedia Conference, pp.57-66, 1999.

[9] Lu Guojun, "Indexing and Retrieval of Audio : A Survey," Journal of Multimedia Tools and Applications, pp.269-290, 2001.

[10] E. Wold, et al., "Content-based Classification, Search and Retrieval of Audio," IEEE Multimedia 3(3), pp.27-36, 1996.

[11] R. Typke and L. Prechelt, "An Interface for melody input," ACM Transactions on Computer-Human Interaction, pp. 133-149, 2001.

[12] J. T. Foote, "Content-Based Retrieval of Music and Audio," In : Multimedia Storage and Archiving Systems II - Proceedings of SPIE, pp.138-147, 1997.

[13] A. Kornstadt, "Themefinder : A Web- Based Melodic Search Tool," Computing in Musicology 11, MIT Press, 1998.

[14] D. Huron, C. S. Sapp, B. Aarden, Themefinder, http://www.themefinder.org/, 2000.

[15] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval, Addison Wesley, 1999.

[16] G. Michael, "Representing Music Using XML," International Symposium on Music Information Retrieval, 2000.

[17] jMusic Java library, http://jmusic.ci.qut.edu.au/.

[18] K. Lemstorm, et al., "SEMEX - An efficient Music Retrieval Prototype," In : Proceeding of Symposium on Music Information Retrieval, 2000.

[19] E. Hwang, D. Park, "Popularity-Adaptive Index Scheme for Fast Music Retrieval," In : Proceedings of IEEE Multimedia and Expo, 2002.

[20] E. Hwang, S. Rho, "Fast Melody Finding Based on Memorable Tunes," 1st International Symposium onComputer Music Modeling and Retrieval, Montpellier, France, pp.227-239, May, 2003.

[21] J. T. Foote, "An Overview of Audio Information Retrieval," ACM-Springer Multimedia Systems, pp.2-10, 1998.

[22] K. Lemstorm, et al., "Retrieving Music - To Index or not to Index," ACM International Multimedia Conference (MM '98), pp.64-65, 1998.

[23] MiDiLiB project, "Content-based indexing, retrieval, and compression of data in digital music libraries," http://www-mmdb.iai.uni-bonn.de/forschungprojekte/midilib/english/.

[24] M. J. Hawley, Structure out of Sound, PhD thesis, MIT. 1993.

[25] M. Melucci, et al., "Musical Information Retrieval using Melodic Surface," In : Proceedings of the ACM Digital Libraries Conference, pp.152-160, 1999.

[26] P. Salosaari, et al., "MUSIR-A Retrieval Model for Music," Technical Report RN-1998-1, University of Tampere, 1998.

[27] P. Y. Rolland, et al., "Musical Content-Based Retrieval : an Overview of the Melodiscov Approach and System," In : Proceedings of ACM multimedia 98 - Electronic Proceedings, pp.81-84, 1999.

[28] P. Y. Rolland, "Music Information Retrieval : a brief Overview of Current and Forthcoming Research," In : Proceeding of Human Supervision and Control in Engineering and Music, 1999.

[29] R. J. McNab, et al., "Toward the digital music library : tune retrieval from acoustic input," In : Proceedings of the first ACM Conference on Digital Libraries, pp.11-18, 1996.

[30] S. R. Subramanya, et al., "Transforms - Based Indexing of Audio Data for Multimedia Databases," IEEE International Conference on Multimedia Systems, 1997.

[31] S. R. Subramanya, et al., "Use of Transforms for Indexing in Audio Databases," International Conferenc e on Computational Intelligence and Multimedia Applications, 1999.

[32] Y. H. Tseng, "Content-Based Retrieval for Music Collections," In : Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.176-182, 1999.

<부 록 1>

Query Process Algorithm in FAI

(1) Get a humming query from input device - microphone, CMN ;

(2) While (any query comes into the system)

If (Exact Matching is selected)

Call **ExactMatch**() ;

Else

Call **ApproxMatch**() ;

(3) List up the candidate result melody ordered by the highest ranked melody first ;
(4) Play the retrieved melody ;

/* Definition of Functions */
Function ExactMatch(query q)
If (melody q is equal to the melody in FAI)
Increase access_count value of FAI entry ;
Return matched_melody ;
Else
ResultSet = search the whole music DB ;
If (same melody is found in ResultSet)
Add a melody into FAI ;

Function ApproxMatch(query q)

Empty Matrix M for query q and string in FAI
with i-th row and j-th column ;
M[0, j] = 0 and M[i, 0] = i ;
While (No errors found)
If (melody q is equal to the melody in FAI)
M[i, j] = M[i-1, j-1] ;
Else
M[i, j] = 1 + Min (M[i - 1, j], M[i, j - 1],
M[i - 1, j - 1]) ;

If (M[i, j] < Threshold)
Increase access_count value of FAI entry ;
Return matched_melody ;
Else
ResultSet = search the whole music DB ;
If (same melody is found in ResultSet)
Add melody into FAI ;

<부 록 2>

FAI Management Algorithm

Function FAI_*Update*
Input : query melody **q**, current music number **n**,
overlapped melody number **i**,
music[n].FAI[i],
music[n].FAI[i].access_count,
music[n].FAI[i].age
music[n].FAI[i].repetition

(1) Find the entry which is overlapped with query melody ;
(2) Merge overlapped entries with query melody into one melody ;
(3) Call ***deleteEntryfrom*FAI** (musicNum, entryTobeDeleted) ;
(4) Call ***insertMelodyinto*FAI** (musicNum, squeryMelody) ;

Function *InsertMelodyinto*FAI
Input : current music number **n**, query melody **q**
Variable : q.position,
max_FAI_num **f** = music[n].FAI.max,
music[n].FAI[f].access_count,
music[n].FAI[f].age
music[n].FAI[f].repetition
(1) Find the position of new entry ;
(2) music[n].FAI[f+1] = q ;
(3) music[n].FAI[f+1].access_count = 1 ;

Function *DeleteEntryfrom*FAI
Input : current music number **n**,
number of FAI entry to be deleted **d**
(1) music[n].FAI[d] ;
(2) music[n].FAI.max = music[n].FAI.max-1 ;

Function *MelodyCompare*
Input : Result result
output : Overlapped_melody
Variable : FAI temp
(1) Copy FAI entry into temp ;
(2) for (i = 0 ; i++ ; i = n) // loop for melody
for (j = 0 ; j++ ; j = temp[n].FAI.max)
if (result == temp[n].FAI[j] ;
return temp[n].FAI[j] ;
else return ;

노 승 민

e-mail : anycall@ajou.ac.kr

2001년 아주대학교 정보및컴퓨터 공학과 (학사)

2003년 아주대학교 정보통신 전문대학원 (석사)

2003년~현재 아주대학교 정보통신 전문대학원 박사과정

관심분야 : 데이터베이스, 멀티미디어 시스템, XML 응용, DRM

황 인 준

e-mail : ehwang@ajou.ac.kr

1988년 서울대학교 컴퓨터공학과(학사)

1990년 서울대학교 컴퓨터공학과(석사)

1998년 Univ. of Maryland at College Park 전산학과(박사)

1998년~1999년 Bowie State Univ., Assistant Professor

1999년~1999년 Hughes Research Lab. 연구교수

1999년~2002년 아주대학교 정보통신전문대학원 조교수

2003년~현재 아주대학교 정보통신전문대학원 부교수

관심분야 : 데이터베이스, 멀티미디어 시스템, 정보 통합, 전자상거래, XML 응용