사용자 질의 패턴을 이용한 효율적인 오디오 색인 기법

(An Efficient Audio Indexing Scheme based on User Query Patterns)

노 승 민 † 박 동 문 †† 황 인 준 †††
(Seungmin Rho) (Dongmoon Park) (Eenjun Hwang)

요 약 디지털 오디오 컨텐츠의 활용이 보편화되면서 오디오 데이타베이스에 대해 컨텐츠를 효과적으로 질의하고 검색하는 기능이 절실해졌다. 본 논문에서는 사용자들이 자주 질의하는 멜로디 부분을 이용하여 오디오 검색을 효과적으로 수행할 수 있는 새로운 인덱스 기법을 제안한다. 이 기법은 어떤 오디오에 대해 사용자가 기억하고 질의하는 내용이 대개 오디오의 특정 부분에 한정되어 있다는 사실에 기반하고 있다. 사용자의 이러한 질의 패턴을 이용하여 자주 질의되는 부분을 감지하고 인덱스로 사용함으로써 사용자가 원하는 곡을 빠르게 찾아낼 수 있게 해준다. 이러한 방법은 데이타베이스의 모든 컨텐츠를 순차적으로 검색하는 기존의 방법에 비해 적은 비용으로 검색 속도를 크게 향상시키며 특히 기존의 검색 시스템 상위 모듈로 사용이 가능하다. 프로토타입 시스템을 구현하고 다양한 실험을 통하여 논문에서 제안하는 기법의 우수성을 보인다.

키워드 : 오디오 검색, 패턴 매칭, 인덱싱, 멀티미디어 데이타베이스

Abstract With the popularity of digital audio contents, querying and retrieving audio contents efficiently from database has become essential. In this paper, we propose a new index scheme for retrieving audio contents efficiently using audio portions that have been queried frequently. This scheme is based on the observation that users have a tendency to memorize and query a small number of audio portions. Detecting and indexing such portions enables fast retrieval and shows better performance than sequential search-based audio retrieval. Moreover, this scheme is independent of underlying retrieval system, which means this scheme can work together with any other audio retrieval system. We have implemented a prototype system and showed its performance gain through experiments.

Key words : Audio Retrieval, Pattern Matching, Indexing, Multimedia Database

1. 서 론

기존의 웹에서는 대부분 텍스트나 HTML(Hyper-Text Markup Language)을 이용해서 정보를 표현하였으나, 요즘은 이미지나 오디오, 비디오와 같은 멀티미디어 컨텐츠를 이용하는 경우가 많아졌다. 이렇게 멀티미디어 컨텐츠의 종류와 양이 증가하면서 멀티미디어 컨텐츠와 관련된 기술들도 발전하게 되었다. 특히, 멀티미디어 컨텐츠를 검색하는 분야에서 많은 연구들이 진행되었으며 그 흐름은 크게 주석 기반(Annotation-based) 검색과 내용 기반(Content-based) 검색으로 나누어 볼 수 있다. 주석 기반 검색은 컨텐츠에 대한 주석을 미리 달아놓아야 하고 사용자가 찾고자 하는 멀티미디어 데이타를 질의로 작성하기가 쉽지 않기 때문에 내용 기반 검색 방법이 선호되고 있다.

초기의 내용 기반 오디오 검색에서는 MIDI 키보드, 휘파람, 허밍 등을 이용한 질의 표현이 일반적이었다[1]. MIDI 키보드를 이용하면 정확한 질의를 작성할 수 있는 반면 MIDI 키보드 조작에 익숙하지 못한 사용자에게는 질의 작성이 쉽지 않다. 한편, 휘파람이나 허밍을 이용하면 질의를 쉽게 표현할 수 있지만 검색 시스템이

· 본 연구는 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음
† 학생회원 : 아주대학교 정보통신전문대학원
anycall@ajou.ac.kr
†† 비 회 원 : 모빌링크텔레콤 무선통신연구소 연구원
minerva@molink.co.kr
††† 종신회원 : 아주대학교 정보통신전문대학원 교수
ehwang@ajou.ac.kr
논문접수 : 2002년 10월 15일
심사완료 : 2004년 4월 27일

질의 멜로디 작성자의 의도를 정확히 파악하지 못하는 경우가 생길 수 있다. 하지만 이런 질의 인터페이스의 문제에서 발생하는 부정확한 검색은 쉽게 해결되리라고 예상된다.

대개 오디오 검색 시스템은 다음과 같은 일련의 과정을 거친다. 일단, 오디오 컨텐츠를 데이타베이스에 저장하기에 앞서, 각 오디오 컨텐츠의 특징들을 뽑아낸 다음 그 특징들을 가공해서 오디오 특징 데이타베이스(Audio Feature Database)에 저장하고 실제 오디오 데이타베이스(Audio Database)의 인덱스로 사용한다[2,3]. 이 때 인덱스로 많이 사용되는 음향 특징(Acoustic Feature)으로는 음색(timbre), 음높이(pitch) 등이 있다[3,4]. 오디오 특징 정보를 뽑아내기 위해서는 시그널 프로세싱을 이용하거나 MIDI 파일 안에 들어있는 오디오 정보를 이용한다. 시그널 프로세싱을 이용하면 모든 오디오 파일 포멧에 적용할 수 있는 반면 구현하기가 힘들고 검색 시간이 길어질 수 있기 때문에 대부분의 오디오 검색 시스템은 MIDI 파일을 이용하는 방법을 사용한다.

오디오 데이타베이스에서 질의 멜로디와 유사한 오디오 컨텐츠를 찾기 위한 방법으로 UDR 스트링 비교 기법이 있다[2,3,5]. UDR 스트링은 다음의 규칙에 따라 컨텐츠로부터 생성되는 문자열이다:

- **U (Up)**: 현재 음표의 음높이가 이전 음표의 음높이보다 높을 경우
- **D (Down)**: 현재 음표의 음높이가 이전 음표의 음높이보다 낮을 경우
- **R (Repeat)**: 현재 음표의 음높이가 이전 음표의 음높이와 같을 경우

모든 오디오 컨텐츠를 데이타베이스에 저장하기에 앞서 UDR 스트링으로 변환해서 UDR 정보 데이타베이스에 넣어둔다. 사용자의 질의 멜로디가 들어오면 해당 UDR 스트링을 만들고 UDR 정보 데이타베이스에서 유사한 UDR 스트링을 가진 오디오 컨텐츠를 찾는 것이다. UDR 스트링은 오디오 컨텐츠를 텍스트로 표현한 정보이므로 검색을 위해서는 서브스트링 매칭(Substring Matching) 기술이 필요하다. 특히 사용자가 기억해서 질의하는 휘파람이나, 허밍의 경우 원곡의 UDR 스트링과 조금 다를 수 있다. 따라서 이러한 사용자의 부정확한 질의 멜로디를 고려해서 유사 매칭(Approximate Matching) 방법을 이용할 수도 있다[6].

본 논문에서는 FAI(Frequently Accessed Index)라는 새로운 내용 기반 오디오 검색 기법을 제안한다. 이 기법은 대부분의 사람들이 어떤 노래에 대해 기억하는 부분이 그 노래의 메인 멜로디(반복되는 구간)나 클라이맥스 등 특정 부분에 한정되어 있다는 관측을 이용한다. 이것은 어떤 노래를 찾기 위해 작성하는 질의는 결국 자신이 기억하는 특정 멜로디에 의존하게 된다. 그렇기 때문에 자주 질의되는 멜로디 부분을 기억하고 관리하면, 가장 많이 검색되는 멜로디 부분을 학습을 통해 알 수 있게 된다. 이 정보를 모아둔 곳이 FAI이며 오디오를 검색할 때 멜로디 데이타베이스 전체를 검색하는 대신 FAI의 항목을 먼저 검색한다. 따라서 질의 멜로디가 FAI에서 발견되면 바로 원하는 오디오 컨텐츠를 검색할 수 있으므로 검색 속도가 향상된다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 기초가 되었던 연구들과 함께 기존의 오디오 검색 시스템에 대해서 살펴보도록 하겠다. 3장에서는 FAI 기반 오디오 검색 시스템의 구조와 인덱싱 메카니즘에 대해서 살펴보고 4장에서는 FAI을 기반으로 한 오디오 검색과 일반적인 방법을 사용한 오디오 검색을 실험을 통해서 비교하며, 마지막으로 결론 및 향후 연구 방향에 대해 언급한다.

2. 관련 연구

내용 기반 오디오 검색 시스템의 기본적인 패러다임은 소위, QBH(Query-By-Humming) 혹은 WYHIWYG (What You Hum Is What You Get)이다. [1]은 내용 기반 오디오 검색 분야에서 선구적인 연구 내용을 소개하고 있다. 마이크로 폰을 통하여 받은 사용자의 허밍에서 음높이 변화(Pitch Contour)를 감지하여 UDR 스트링으로 표현하고 오디오 데이타베이스의 컨텐츠와 비교해서 멜로디를 찾아내는 기법을 소개하고 있다. [6]에서는 MELody inDEX라는 웹기반 오디오 검색 시스템를 소개하고 있다. 역시 마이크로 폰을 이용해서 사용자의 질의를 받고 질의 멜로디와 오디오 데이타베이스 내의 멜로디를 비교해서 유사한 정도에 따라 후보 멜로디의 리스트를 보여 주고 재생할 수 있도록 하였다. 오디오 검색에서는 멜로디의 음향이나 간격, 리듬을 이용했고 멜로디 UDR 스트링 매칭을 위해서 다이나믹 프로그래밍 알고리즘을 이용한 유사 검색 기법을 사용하였다. Wold[7]은 신호 처리를 이용해서 음의 크기(loudness), 높이(pitch), 밝기(brightness), 대역폭(bandwidth), 조화도(harmonicity) 등과 같은 음향 특징을 뽑아내고 이 특징을 벡터로 표현해서 대상 오디오의 음향 특징 정보와 질의 오디오의 특징 벡터 거리(vector distance)를 비교하여 오디오 컨텐츠를 분류했다. Foote[3]는 시그널에서 오디오 특성을 뽑아내며 유사 매칭에 있어서 일반적으로 사용되는 음높이 변화나 스팩트럼 변화를 이용하지 않고, 트리 기반의 벡터 양자기(quantizer)를 이용해서 계산 시간을 줄이려는 시도를 하였다. Blackburn[8]은 기존의 접근 방식과는 다르게, 오디오 데이타가 선형(linear) 구조가 아닌 구조적이고 분기하는 구조

를 가져서 한 오디오에서 다른 오디오의 특정 부분으로 링크할 수 있다는 가정 하에, 멜로디 음색의 변화를 이용해서 멀티미디어 문서 안팎에 있는 MIDI 포맷, 디지털 오디오 파일을 대상으로 특정 부분에 접근할 수 있는 내용 기반 항해(Content-Based Navigation) 도구를 개발했다.

MIDI 포맷의 특성상 MIDI 파일은 여러 개의 트랙과 채널을 가질 수 있다. 이런 음악을 다성음악(polyphonic)이라고 하는데, 여러 악기를 동시에 이용하여 연주된 MIDI 파일의 경우, 어떤 부분은 여러 악기의 소리가 겹쳐져서 각 악기의 원음과는 다른 음으로 인식되기도 한다. 이런 문제점을 해결하기 위해 Ghais[1]는 단순히 퍼커션, 드럼 같은 리듬 악기의 채널을 없애는 방법을 사용했고, Uitdenbogerd[2]는 여러 악기 중 가장 높은 음을 내는 악기의 멜로디를 주요 멜로디라 가정하고 그 멜로디를 이용하여 질의 멜로디와 비교하는 방법을 제안했다.

기존에 제안된 내용기반 음악 검색 시스템에서는 전체 멜로디를 색인으로 이용하지 않고, 일부분의 멜로디만을 색인으로 구성하여 이를 검색에 이용하여 검색시간을 줄이려는데 노력을 기울였다. [9]에서는 사용자가 주로 기억하는 처음 멜로디 부분을 트라이(trie) 구조의 색인으로 구성하여 검색하고자 하였다. 하지만, 기억되는 첫 멜로디가 아닌 다른 멜로디로 질의를 하게 될 경우 전체 데이타베이스를 모두 검색해야하는 단점이 있다. 마찬가지로 [10]에서는 멜로디를 suffix tree 구조의 색인으로 구성하였고, 검색 알고리즘은 정확 매칭 알고리즘 중에서 빠른 성능을 가지는 BM(Boyer-Moore) 알고리즘을 사용하였다. [11]에서는 색인으로 구성될 멜로디의 크기를 음악에서 독립성을 지니는 최소단위인 동기로 보고, 동기 단위의 유사도를 계산하고, 계산된 값을 이용해 유사한 동기들을 클러스터링하여, 반복되는 주제 선율을 추출하여 색인을 하였다.

3. 오디오 검색 시스템

본 논문에서 제안하는 내용 기반 오디오 검색 시스템은 이식성과 플랫폼 독립성을 위해 자바로 구현하였다. 현재 자바 애플리케이션으로 개발되었지만, 애플릿으로 변환하면 어떠한 시스템에서라도 브라우저를 통해 이용할 수 있다. 오디오 컨텐츠는 모두 MIDI 파일을 사용했고 MIDI 파일을 처리하는 기능을 제공하는 jMusic[12] 라이브러리를 이용해서 오디오 음향 특징 정보를 추출했다.

3.1 시스템 구조

그림 1은 FAI 기반 오디오 검색 프로토타입 시스템의 대략적인 구조를 보여준다. 시스템은 크게 오디오 데이타베이스(Audio Database), 오디오 질의 입력 처리기(Audio Query Input Interface), 오디오 분석기(Audio Analyzer), 오디오 매칭 엔진(Audio Matching Engine) 그리고 오디오 재생기(Audio Player)로 구성되어 있다.

오디오 데이타베이스(Audio Database): 사용자들로부터 질의를 받아들이기 전에 오디오 데이타베이스에는 오디오 컨텐츠들을 저장해 두고 미리 UDR 스트링으로 변환해서 실제 오디오 컨텐츠에 대한 포인터와 함께 저장해 둔다.

오디오 질의 입력 처리기(Audio Query Input Interface): 오디오 질의는 마이크로 폰을 이용한 허밍이나, 질의하고자 하는 멜로디를 오선지에 기입하는 CPN(Common Practice Notation) 기반의 방식으로 표현된다. 사용자가 작성한 질의 멜로디는 MIDI 포맷으로 변환되어 저장된다. 사용자의 입력과는 다르게 미리 작성된 샘플 MIDI 파일을 질의로 입력하는 방식도 가능하다. MIDI 포맷의 질의 파일은 오디오 분석기로 넘어

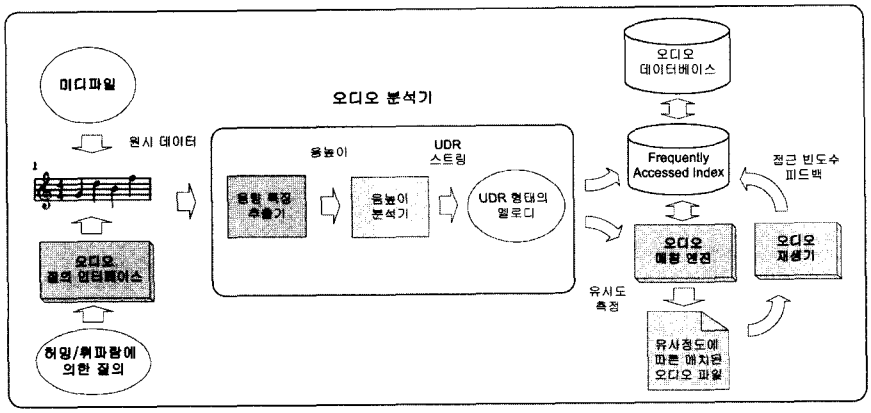


그림 1 FAI 기반 오디오 검색 시스템 구조

가서 음향 특징을 추출하고 UDR 스트링으로 변환하는데 사용된다. 오디오 분석기는 크게 두 가지 모듈로 구성된다.

음향 특징 추출기(Acoustic Feature Extractor): MIDI 파일 안에는 음악을 재생하는데 필요한 모든 정보가 들어있다. 여러 가지 음향 정보 중에서 음높이 값을 뽑아내는 것이 이 모듈의 기능이다. 사용자가 작성한 질의 MIDI 파일을 받아들이고 모든 음표에 대한 음높이 값을 추출하여 순서대로 임시공간에 저장한다.

음높이 분석기(Pitch Analyzer): 음향 특징 추출기가 저장한 음높이의 값을 비교하는 것이 이 음높이 분석기의 기능이다. 앞에서도 언급했듯이, 앞 음표의 음높이과 현재 음표의 음높이를 비교해서 현재 음표의 음높이가 상대적으로 높으면 U, 낮으면 D, 같으면 R로 표기하여 음높이의 변화를 바탕으로 UDR 스트링을 생성한다. 결국, 사용자의 질의 멜로디는 오디오 분석기를 거쳐서 UDR 스트링으로 변환된다. 이 UDR 스트링은 오디오 매칭 엔진(Audio Matching Engine)으로 보내진다.

오디오 매칭 엔진(Audio Matching Engine): 오디오 매칭 엔진은 미리 UDR 스트링으로 저장된 오디오 컨텐츠와 질의 UDR 스트링을 비교하게 된다. 사용자의 질의가 그들의 기억에 의존하고 휘파람(Whistling)이나, 허밍(Humming)을 이용하여 질의를 생성할 경우 사용자의 부정확한 질의로 인하여 원곡의 스트링과 많이 다를 수 있기 때문에, 이와 같은 경우에는 유사 매칭(Approximate Matching)을 이용한다. 유사 매칭은 질의 UDR 스트링의 어느 부분 값을 하나씩 변화 시켜가면서 비교하는 방법인데, 만약 질의 멜로디가 길거나 오디오 데이타베이스의 컨텐츠 수가 많으면 시스템 성능 저하를 가져올 수 있다. 따라서, 본 논문에서는 샘플 미디 파일이나 CPN기반의 방식으로 질의를 생성할 경우에는 정확 매칭(Exact Matching) 방법에서 많이 쓰이고 있는 Naive와 KMP(Knuth-Morris-Pratt) 그리고 BM(Boyer-Moore) 알고리즘 중에서 가장 빠른 BM 알고리즘을 사용하여 검색 성능을 높였다.

오디오 재생기(Audio Player): 오디오 매칭 엔진이 질의 멜로디의 UDR 스트링과 부합되는 오디오 컨텐츠를 찾으면 그림 2에서 보는 것과 같이 화면에 검색 결과 리스트를 만들어 주고 사용자는 검색된 후보 곡을 오디오 재생기를 통해 재생해 볼 수 있다. 본 시스템에서는 오디오 컨텐츠의 재생을 위해서 QuickTime Audio Player를 사용하였다.

3.2 FAI 기반의 다이나믹 인덱싱 메커니즘

내용 기반 멀티미디어 검색 시스템은 어떠한 인덱싱 방법과 유사도 검색 방법을 사용하느냐에 따라서 그 성능이 좌우된다. 일반적인 오디오 검색 시스템에서는 오디오 컨텐츠의 음향 특징들을 인덱스로 사용한다. 본 시스템 역시 음향 특징 중 하나인 음높이의 변화를 이용해서 UDR 스트링을 만들고 검색하는데 이용했다.

앞에서 언급했듯이, 어떤 노래에 대해서 사람들이 기억하는 부분은 반복되는 구간 혹은 클라이맥스 등 특정 부분에 한정되는 경우가 많다. 따라서 본 시스템은 사용자가 A 라는 곡을 찾기 위해 a 라는 질의 멜로디를 작성하면, 이 질의 멜로디의 위치를 따로 저장하고 이 멜로디에 대한 접근 회수를 위해 접근 카운트(Access Count) 변수를 초기화한다. 만약 다른 사람이 A 곡을 찾기 위해 같은 멜로디로 질의를 한다면 이 노래에서 멜로디 a 에 대한 접근 카운트 값을 높여주는 것이다. 이후 이 접근 카운트의 값이 특정 임계치(본 시스템에서는 '10'으로 설정)보다 커지면 이 질의 멜로디와 해당

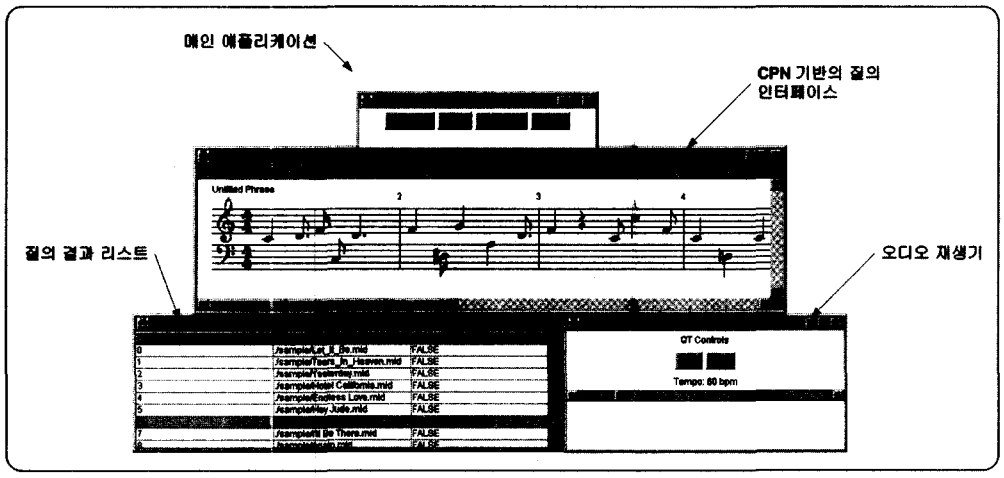


그림 2 질의 입력기와 결과리스트 및 오디오 재생기 인터페이스

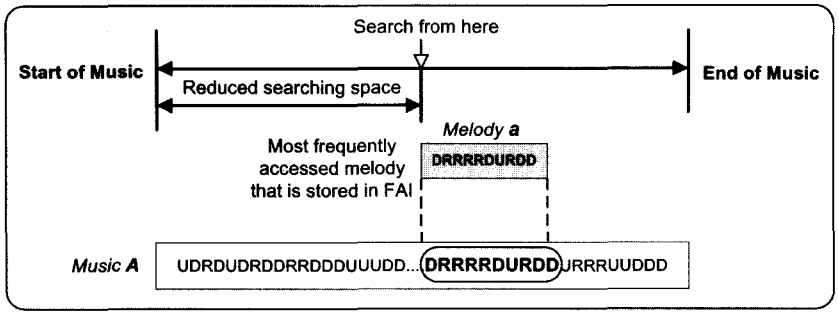


그림 3 FAI 기반 오디오 검색의 특징

오디오에 대한 링크를 FAI(Frequently Accessed Index)라는 인덱스 안에 저장한다. 그 다음부터 사용자 질의를 받고 이 멜로디 질의를 찾기 위해 오디오 데이타베이스를 검색을 할 때는 모든 오디오 컨텐츠의 UDR 스트링을 검색하는 게 아니라 FAI를 먼저 검색하고, 질의 멜로디가 FAI 항목 안에 없을 경우에만 나머지 부분을 검색하게 된다. FAI는 사람들이 가장 많이 기억하는(질의하는) 멜로디의 UDR 스트링을 모아둔 것이기 때문에, 이 FAI 항목 안에서 사용자의 질의 멜로디가 검색될 가능성이 많다. 따라서, 모든 오디오 컨텐츠를 검색해야하는 경우를 줄여준다. 이 FAI 기반 오디오 검색은 사용자의 질의 패턴를 이용하기 때문에 사용자의 질의 회수가 많아질수록 축적되는 정보가 증가하고 정확해진다.

그림 3은 FAI 기반 오디오 검색의 특징을 보여준다. 만일 그림 3에서 가장 자주 질의되었던 멜로디인 a가 FAI 항목으로 할당되어있고 사용자가 작성한 질의 멜로디가 a와 같은 멜로디라면 멜로디 a 앞 부분은 검색하지 않아도 된다. 물론, 사용자가 찾고자하는 멜로디가 FAI 항목 안에 있지 않다면 나머지 부분도 검색을 하게 되어 FAI를 사용하는 효과가 줄어들 수도 있다.

3.3 FAI 인덱스 관리

3.3.1 FAI 인덱스의 생성과정

그림 4는 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 데이타베이스에서 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 카운트 변수도 포함되어있다. 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, ③

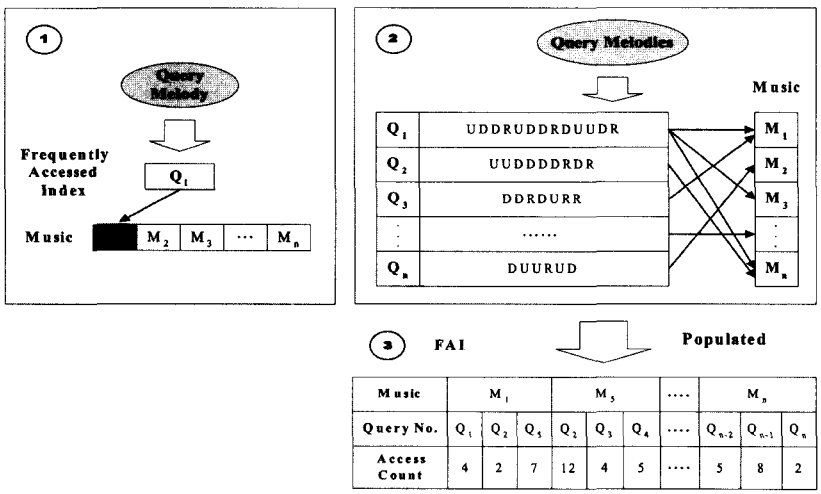


그림 4 FAI 기반 인덱싱의 메카니즘

은 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 것이 아니라, 기존의 FAI 엔트리와 비교해서 질의 멜로디와 중복되는 것은 엔트리에 추가하지 않는다. 즉, 한 노래를 가리키는 FAI 항목은 여러 개가 될 수 있고 서로 중복되지 않으며, 그 노래에 대한 서브 UDR 스트링의 집합이라고 할 수 있겠다. 또한, 질의와 FAI 엔트리 멜로디 사이의 시작과 끝이 일치하지 않은 경우에 대해서는 부분 매칭을 해야하는데, 원본 멜로디와 질의 멜로디에서 매칭하는 부분보다 매칭하지 않는 부분이 많거나 매칭되는 질의 멜로디가 원본 멜로디의 극히 일부분에 지나지 않는 경우는 제외한다. 그외의 경우에는 두 멜로디를 확장하여 FAI 엔트리에 넣는다.

3.3.2 FAI 인덱스 항목의 확장

경우에 따라 사용자의 질의 멜로디가 기존의 FAI 인덱스 항목과 정확하게 일치하지 않을 수 있다. 다시 말해서, 질의 멜로디가 FAI 항목보다 조금 앞에서부터 시작하거나 조금 길어서 뒤에서 끝날 경우, 오디오 매칭 엔진은 이 두 멜로디가 같은 멜로디를 나타냄에도 불구하고 시작이나 종료 위치가 약간 다름으로 인하여 매칭되지 않는다는 결론을 내릴 수 있다. 이러한 문제점을 해결하기 위해서 다음과 같이 FAI 항목을 확장한다.

그림 5에서 a_f 와 b_f는 A곡에 대한 FAI 항목이고, a_q 와 b_q는 사용자가 작성한 질의 멜로디이다. 그림에서도 알 수 있듯이 이 멜로디들은 상당히 비슷하다. a_f 와 a_q는 서로 겹치고 b_q는 b_f를 포함하고있다. a_q와 b_q는 각각 a_f 와 b_f를 찾으려는 의도임에도 불구하고 시작 혹은 종료 위치가 조금 틀려 오디오 매칭 엔진에 의해 다른 멜로디로 인식되고 결과적으로 데이타베이스의 컨텐츠를 뒤지게 되는 것이다. 따라서, 오디오 데이타베이스 전체의 검색으로 찾은 질의 멜로디의 위치가 기존 FAI 항목 멜로디의 위치와 비슷할 경우 질의 멜로디에만 나타나있는 스트링을 FAI 항목에 삽입하여 확장한다. 결과적으로 a_f 와 a_q는 a_{ext}로, b_f 와 b_q는 b_{ext}로 확장된다.

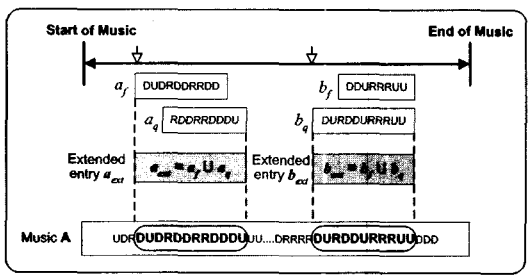


그림 5 FAI 항목의 확장

3.3.3 FAI 인덱스 항목의 수정 및 소멸

빠르고 효과적인 검색을 위해서 FAI 항목의 개수를 제한할 필요가 있다. 만약 FAI에 새로운 항목을 위한 공간이 넘을 경우 새로운 항목을 위해 기존의 항목중에서 불필요한 항목을 없애야 한다. 본 시스템에서는 이런 이유로 한 곡 당 FAI 항목의 개수가 3을 넘지 못하게 하였다. 다음은 FAI 항목의 개수를 3이라고 가정하고 예제를 통해 FAI 내부 항목의 유지 및 관리에 대해서 설명하겠다.

FAI 항목의 수정 및 삭제: FAI 항목들은 특정 멜로디에 대한 빈도수를 유지하기 위한 접근 회수(Access Count)라는 변수를 가진다. 이 변수에서는 FAI 항목에 대한 질의가 있을 때마다 질의된 멜로디의 접근 값을 하나씩 증가시킨다. 새로운 항목이 추가될 경우 그 항목의 접근 회수 값은 1로 설정되며 병합이 되는 경우에는 병합되는 항목 중에 가장 큰 접근 회수값+1의 값으로 설정된다.

FAI가 가득 찼을 경우 새로운 항목에 대한 공간 할당을 위해 기존의 항목중 하나를 삭제해야 한다. 그 경우 오래된 항목을 파악할 용도로 FAI 항목에 나이(Age) 변수를 두었다. 나이 변수는 FAI 항목에 대한 질의가 있으면 모든 FAI 항목들의 그 값을 하나씩 증가시키며 FAI 항목은 오래된 것일수록 큰 Age 값을 가지게 되며 각 항목의 Age 값은 중복되지 않는다. 따라서 FAI 항목에 처음 추가되는 항목이나 병합되는 항목의 Age 값은 항상 1로 설정한다. 기존 항목의 접근회수는 그대로 유지되고 Age 값은 하나씩 증가한다.

한 곡에 대한 FAI 항목의 개수가 3 개미만으로 여분의 FAI 항목 공간이 있을 경우 질의 멜로디가 들어오면 그 질의 멜로디를 바로 FAI에 추가되며 항목의 접근 회수와 Age 값은 각각 1로 설정된다.

한 곡에 대한 FAI 항목이 3 개로 모두 다 찼을 경우, 새로운 질의 멜로디가 들어갈 공간이 없다. 따라서 FAI에 새로운 질의를 반영하면서 FAI 항목 개수를 유지하기 위해서는 기존의 FAI 항목의 접근 회수와 Age 값을 고려해서 되도록이면 접근 회수가 낮고 오래된 항목을 삭제해야 한다. 따라서 FAI 항목이 모두 다 찼을 경우 우선 접근 회수가 가장 낮은 항목을 찾기 위해 각 항목의 접근 회수를 먼저 비교한다.

(1) 한 곡에 대한 FAI 항목들이 모두 같은 접근 회수 값을 가진 경우: 입력된 질의 멜로디가 기존의 FAI 항목의 멜로디와 겹치지 않으면, 기존의 FAI 항목 중에 가장 오래된 항목을 삭제하고 질의 멜로디를 새로운 항목으로 추가한다. 반면, 입력된 질의 멜로디가 기

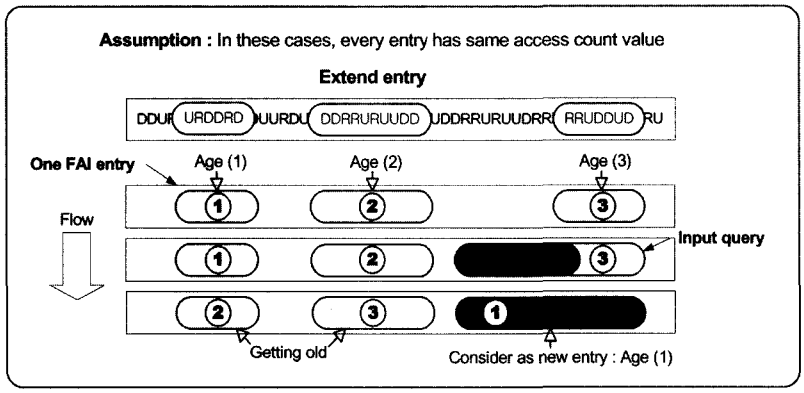


그림 6 Age 값을 고려한 FAI 항목들의 확장

존 FAI 항목 중 어느 항목과 겹치는 경우 FAI 항목과 질의 멜로디를 병합하여 새로운 항목으로 삽입한다. 이 경우, 기존 항목의 Age 값은 소멸되고 1로 설정된다. 병합되지 않는 나머지 FAI 항목들은 연대적 관계에 따라서 Age 값을 적절하게 증가시킨다.

만일 두 개 이상의 FAI 항목이 입력된 질의 멜로디와 겹치게 되면, 역시 FAI 항목들과 질의 멜로디가 하나로 병합되기 때문에, 기존 FAI 항목의 수는 질의 멜로디가 얼마나 많은 FAI 항목과 병합되느냐에 따라서 기존의 항목이 3 개에서 2 개 혹은 3 개에서 1 개까지 줄어들 수가 있다.

그림 6은 FAI 항목들의 접근 회수가 모두 같고 항목의 개수가 3 개로 모두 차 있는 상태에서 항목이 확장되는 예를 보여주고 있다. 기존 항목들은 Age 값으로 각각 1, 2, 3을 가지고 있다. 이 경우 질의 멜로디와 Age가 3인 항목이 겹치므로(이후로, Age(n)은 Age의 값이 n임을 뜻한다), 질의 멜로디와 Age(3) 항목이 병합된다. 병합된 항목은 새로운 항목으로 간주되어 Age 값이 1로 설정되고 나머지 항목 Age(1)과 Age(2)는 각각 Age(2)와 Age(3)로 Age 값을 증가시켰다.

그림 7은 질의 멜로디가 어떠한 멜로디와도 겹치지 않는 경우인데, 이때는 가장 오래된 항목이 삭제되고 질의 멜로디가 새로운 항목으로 삽입된다. 항목의 Age 값이 3, 1, 2이므로 가장 오래된 Age(3)의 항목이 삭제된다. 나머지 항목의 Age 값들은 하나씩 증가하게 되며 역시 새로운 항목의 Age 값은 1로 설정된다.

(2) FAI 항목들의 접근 회수가 각각 다른 경우: 기존 FAI 항목 중에서 가장 낮은 접근 회수를 가지는 항목을 삭제하고 질의 멜로디를 추가한다. 만일, 가장 낮은 접근 회수를 가지는 항목이 2 개 혹은 3 개 있을 때에는 같은 접근 회수를 가지는 항목 중에서 가장 오래된 항목을 삭제한다. 새로 추가되는 항목의 Age 값은 1

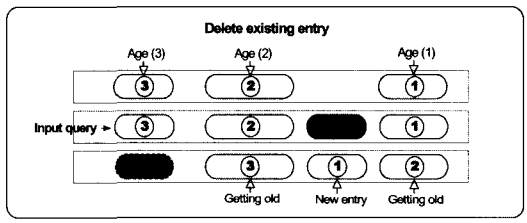


그림 7 Age 값을 고려한 FAI 항목들의 삭제

로 설정한다. FAI 항목을 삭제하고 나면 각 항목들의 Age 값을 적절하게 변경한다.

가장 오래된 FAI 항목일지라도 접근 회수의 값이 가장 작지만 않다면 삭제되지 않는다. 즉, FAI 항목의 삭제에 있어서 오래된 항목보다는 접근 회수 값이 낮은 항목이 우선적으로 지워진다.

그림 8은 FAI 항목의 접근 회수와 Age 값 모두를 고려해서 FAI 항목을 확장, 삭제하는 예제이다. 세 개의 FAI 항목이 모두 채워져 있는 상태에서 질의 멜로디를 새로운 항목으로 추가하기 위해서는 삭제 대상을 먼저 찾아야 하는데, 이를 위해 FAI 항목의 접근 회수를 비교해 본다. 각 FAI 항목의 접근 회수가 순서대로 3, 2, 4이다. 따라서 접근 회수 값이 2로 가장 낮은 두 번째 항목을 삭제하고 질의 멜로디를 새로운 항목으로 삽입하게 된다. 새로운 항목의 접근 회수와 Age 는 각각 1로 되고 나머지 Age(1), Age(3) 항목은 Age 값을 각각 Age(2), Age(4)로 변경한다.

그림 9는 접근 회수와 나이를 같이 고려한 FAI 항목 삽입의 예제다. 기존 FAI 항목의 접근 회수가 각각 4, 2, 2로 되어 있다. FAI 항목은 모두 채워져 있으므로 삭제를 위해 가장 작은 접근 회수를 가지는 항목을 찾는 데, Age(2)와 Age(1)이 여기에 해당된다. 따라서 두 항목 중에서 보다 오래된 항목인 Entry(2, 2) 항목이 삭제되고 질의 멜로디가 새로운 항목으로 삽입된다

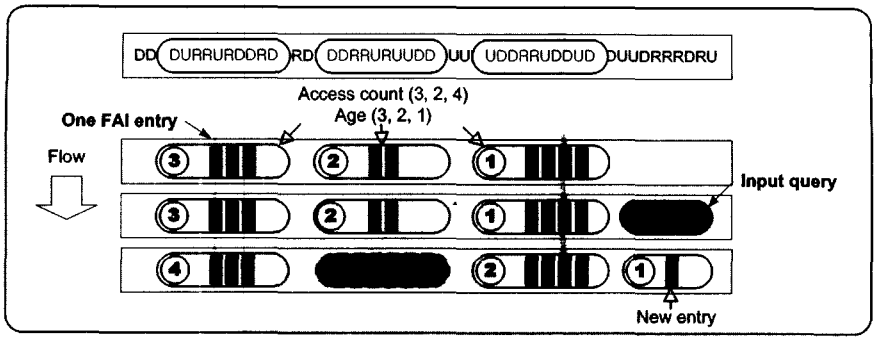


그림 8 접근 회수를 고려한 FAI 항목의 삽입

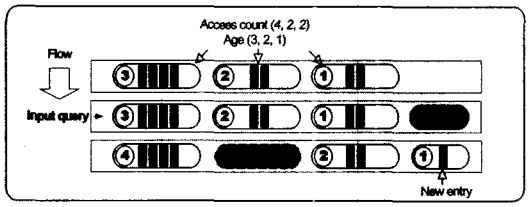


그림 9 접근 회수와 나이를 같이 고려한 FAI 항목 삽입

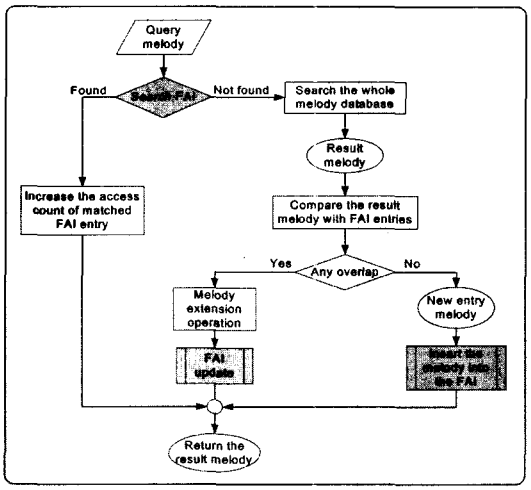


그림 10 오디오 검색 시스템의 흐름도

(Entry(*a*, *c*)는 Age 값이 *a* 이고 Access Count 값이 *c* 인 FAI 항목을 의미한다).

그림 10은 FAI 기반 오디오 검색을 위한 알고리즘의 흐름도로 보여준다. 질의 멜로디가 입력되면 전체 오디오 데이타베이스를 검색하기에 앞서 FAI 인덱스를 먼저 검색한다. 질의와 같은 멜로디가 FAI에서 발견되면, 그 항목의 접근 회수를 높여주고 그 항목을 결과 리스트로 보낸다. 만약 FAI에서 멜로디가 발견되지 않으면 오디오 데이타베이스를 검색하게 된다. 이 오디오 데이터베이스를 검색해서 나온 멜로디에 대한 정보를 FAI 인덱스에 반영해야 하기 때문에 이 결과 멜로디와 기존의 FAI를 한번 더 비교하게 된다. 이 때 기존의 FAI 항목과 결과 멜로디가 겹치는 부분이 있는지 확인하는 과정을 거쳐 겹치는 부분이 있을 경우 두 멜로디를 하나의 더 큰 멜로디로 병합하게 된다. 병합된 멜로디는 새로운 FAI 항목으로 삽입되면서 기존 항목을 대체한다. 만약 FAI 항목과 겹치는 부분이 없을 경우에 결과 멜로디는 FAI에 새로운 항목으로 삽입된다.

```
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
(1) Find the entry which is overlapped with query melody;
(2) Merge overlapped entries with query melody into one melody;
(3) Call deleteEntryfromFAI (musicNum, entryTobeDeleted);
(4) Call insertMelodyintoFAI (musicNum, queryMelody);

Function  InsertMelodyintoFAI
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
(1) Find the position of new entry;
(2) music[n].FAI[f+1] = q ;
    // Insert the new entry to the proper position
(3) music[n].FAI[f+1].access_count = 1;
    music[n].FAI[f+1].access_count = 1;
     // Set the new entry access_count, age

Function DeleteEntryfromFAI
Input: current music number n,
       number of FAI entry to be deleted d
(1) music[n].FAI[d] ;   // Delete the selected entry from FAI
(2) music[n].FAI.max=music[n].FAI.max-1; // Decrease max FAI value

Function MelodyCompare
Input: Result result
output: Overlapped_melody
Variable: FAI temp
(1) Copy FAI entry into temp;
(2) For each melody
       For each FAI entry with temp[n].FAI.max
          //compare the result with every FAI entries
           if(result=temp[n].FAI[j])
              return temp[n].FAI[j];  //return the entry
           else return;
```

그림 11 FAI에서 사용되는 내부함수 알고리즘

그림 11은 FAI 항목을 갱신할 때 호출하는 FAI_Update()와 FAI 인덱스 갱신 처리에 사용되는 내부 함수인 InsertMelodyintoFAI(), DeleteEntryfromFAI(), MelodyCompare()에 대한 알고리즘이다.

그림 12는 그림 11에서 설명한 알고리즘 흐름도에서 나타나는 프로세스들에서처럼 허밍이 들어오면서부터 멜로디가 도출되기까지 거치는 과정을 보여준다.

```
Algorithm FAI_Opertation

(1) Get a humming query from input device;
(2) While (any query comes into the system)
      If (same melody is found in FAI)
          Increase the access_count value of matched FAI entry;
          Return matched_melody;
      // There is no matching melody in FAI
      Else
          Result = search the whole melody database;
          // Searching for any overlapped melody
          Overlapped_melody = MelodyCompare (result, FAI);
          If (any overlapped_melody exist)
              FAI_Update();
          Else    // No overlapped melody
              If (FAI entry has same access_count)
                  Check age;
              Delete oldest FAI entry;
              Else
              Delete low access_count FAI entry
                  InsertMelodyintoFAI();
(3) List up the candidate result melody;
(4) Play the retrieved melody;
```

그림 12 FAI 인덱스 유지 알고리즘

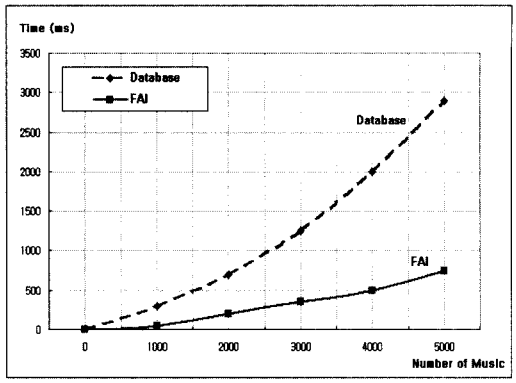
4. 실 험

본 시스템의 성능을 평가하기 위해 인터넷에서 모은 5000 곡의 MIDI 파일을 대상으로 실험을 실시하였다. 실험은 본 논문에서 제안하는 FAI를 기반한 오디오 검색과 FAI를 사용하지 않는 일반적인 검색의 질의 처리 시간을 비교하였다. 이 실험은 Sun Sparc의 Solaris 5.7 에서 수행되었다.

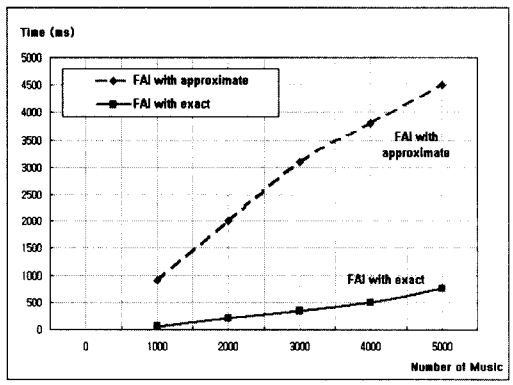
4.1 데이타 및 성능 분석

사용된 오디오 데이타베이스는 5000 여개의 MIDI 파일들로 구성되어 있다. 평균적으로 한 MIDI 파일 당 5737 개의 음표를 가지고 있고 파일에 따라서 그 개수가 1000 개부터 10000 개까지 차이가 난다. 오디오 데이타베이스에 대한 사용자 요청은 Zipf 분산을 따른다고 가정했는데, 이것은 비디오 데이타에 대한 사용자 요청의 분포에서 유추한 것으로 현실 세계와 큰 차이가 없을 것으로 예상된다. 따라서 대부분의 오디오 컨텐츠에는 사용자의 접근이 드물고 상위 20%의 오디오 컨텐츠가 전체 사용자 질의 요청의 70% 가량을 차지하도록 사용자 질의를 생성했다.

본 실험에서는 오디오 데이타베이스의 컨텐츠 수를 증가시키면서 질의 처리 시간을 측정했다. 그림 13(a)는 FAI 기반과 데이타베이스 기반의 평균 질의 처리 시간을 보여준다. 실선으로 표시된 선이 일반적인 오디오 검색의 평균 질의 처리 시간을 나타내고 점선으로 표시된 선이 FAI를 이용한 오디오 검색 질의 처리 시간을 나타낸다. 두 실험 모두 컨텐츠의 수가 증가함에 따라 처리 시간도 같이 증가했다. 하지만 FAI를 기반으로 한 오디오 검색이 모든 데이타베이스를 검색하는 일반 검색보다 월등히 나은 성능을 보였고, 컨텐츠의 수가 증가할수록 FAI 기반 오디오 검색이 얻는 이득은 커졌다. 그림 13(b)는 FAI 항목의 검색에서 정확 매칭과 유사 매칭을 사용할 경우 소요된 평균 질의 처리 시간을 비교하고 있다.



(a) FAI vs. Database



(b) Approximate vs. Exact

그림 13 FAI 기반 검색의 성능 분석

그림 14는 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, 데이타베이스내 멜로디들의 평균 음표수는 56.8개이며 5000 여개의 데이타베이스내 모든 음표수는 284,000 개이다. 그림 14에서 알 수 있듯이 본 논문에서 제안한 FAI 기반의 시스템이 SEMEX 시스템의 *Myers'bit-parallel* 알고리즘과 대부분의 기존 오디오 검색 시스템에서 사용된 *Dynamic programming* 알고리즘에 비해 성능이 우수함을 알 수 있다.

이러한 성능 향상과 관련하여 다음과 같은 수식을 고려해보자:

P_{hit} : 질의 멜로디가 *FAI* 항목 안에서 발견될 확률

P_{miss} : 질의 멜로디가 *FAI* 항목 안에서 발견되지 않을 확률

C_{FAI} : *FAI* 항목을 검색하는데 소요되는 시간

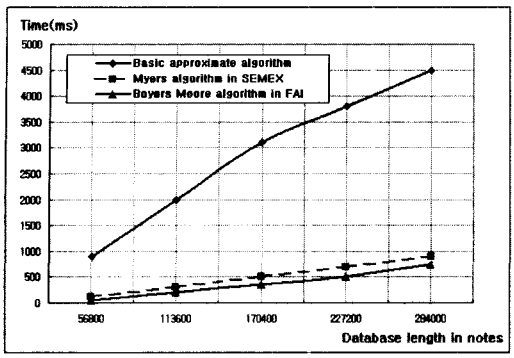


그림 14 SEMEX 시스템과 FAI 기반 검색의 성능 분석

C_{DB} : 데이타베이스를 검색하는데 소요되는 시간

$P_{hit} \times C_{FAI} + P_{miss} \times (C_{FAI} + C_{DB}) << C_{DB}$

사용자 질의 멜로디가 FAI 항목에서 발견될 확률이 그렇지 않을 확률 보다 훨씬 높고 FAI 항목을 검색하는데 소요되는 시간은 데이타베이스를 검색하는데 드는 시간보다 짧다. 대부분의 질의 멜로디가 FAI 항목에서 발견되고 그렇지 않을 경우에만 데이타베이스를 검색하기 때문에, 항상 모든 데이타베이스를 다 검색해야하는 일반적인 검색 방법보다 FAI 기반의 검색 방법이 더 효율적임을 알 수 있다.

5. 결 론

본 논문에서는 사용자가 자주 질의하는 부분을 이용하여 보다 효과적인 오디오 검색을 할 수 있는 기법을 제안하였다. 사용자에 의해 자주 질의되는 멜로디의 위치를 FAI라는 인덱스에 저장해 두고 관리하며 사용자 질의에 대해 FAI를 먼저 검색하게 하였다. 이것은 사용자의 질의가 인기있는 상위 몇 곡에 집중되며 더욱이 질의되는 멜로디 부분이 소수의 특정 부분에 집중되는 특성을 이용하였다. 실험을 통해서 선형적으로 모든 데이타베이스를 검색하는 기존의 방법보다 FAI를 기반한 검색이 훨씬 빠르게 질의를 처리할 수 있음을 보여주었다.

참 고 문 헌

[1] A. Ghias, J.Logan, D. Chamberlin, and B. Smith, "Query by humming-musical information retrieval in an audio database," *Proc. of ACM Multimedia Conference*, San Francisco, 1995.

[2] A. Uitdenbogerd and J. Zobel, "Melodic matching techniques for large music databases," Proc. of ACM Multimedia Conference, Orlando, pp. 57-66, 1999.

[3] J.T. Foote, "Content-Based Retrieval of Music and Audio," Multimedia Storage and Archiving Systems II - Proc. of SPIE, Vol. 3229, pp. 138-147, 1997.

[4] N. Kosugi, Y. Nishihara, and T. Sakata, "A Practical Query-By-Humming System for a Large Music Database," Proc. of ACM Multimedia 2000 Conference, November 2000.

[5] Y.H. Tseng, "Content-Based Retrieval for Music Collections," Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Berkeley, CA, USA, pp. 176-182, August 1999.

[6] R.J. McNab, L.A. Smith, D. Bainbridge, and I.H. Witten, "The New Zealand digital library MELody inDEX," D-Lib Magazine, May 1997.

[7] E. Wold, T. Blum, D. Keislar, and J. Wheaton, "Content-based Classification, Search and Retrieval of Audio," IEEE Multimedia 3(3), pp. 27-36, 1996.

[8] S. Blackburn and D. DeRoure, "A Tool for Content Based Navigation of Music," Proc. of ACM Multimedia, 1998.

[9] Komstadt, Andreas. "Themefinder: A Web-based Melodic Search Tool," Melodic Comparison: Concepts, Procedures, and Applications, Computing in Musicology 11, pp. 231-236, 1998.

[10] Lemstrm, K. and Laine, P., "Musical Information Retrieval Using Musical Parameters," Proc. of International Computer Music Conference (ICMC 98), pp. 341-348, Oct. 1998.

[11] K.I. Ku, et al., "A Content-based Music Information Retrieval System Using Theme Melody Index," Journal of Special Interest Group on Database, Vol. 19, Number 3, pp. 34-45, September 2003.

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

[13] Guojun Lu, "Indexing and Retrieval of Audio: A Survey," Journal of Multimedia Tools and Applications, Vol. 15, pp. 269-290, 2001.

[14] M. Melucci and Nicola Orio, "Musical Information Retrieval using Melodic Surface," Proc. of the ACM Digital Libraries Conference, Berkeley, CA, 1999.

[15] L. Prechelt and R. Typke, "An Interface for melody input," ACM Trans. on Computer-Human Interaction, Vol. 8, Issue 2, pp. 133-149, 2001.

[16] P.Y. Rolland, et al., "Musical Content-Based Retrieval: an Overview of the Melodiscov Approach and System," Proc. of ACM Multimedia Conference, Orlando, pp. 81-84, November 1999.

[17] P.Y. Rolland, "Music Information Retrieval: a brief Overview of Current and Forthcoming Research," Proc. of Human Supervision and Control in Engineering and Music, Kassel, Germany, September 2001.

[18] K. Lemstorm and S. Perttu, "SEMEX-An efficient Music Retrieval Prototype," Proc. of Symposium on Music Information Retrieval, Plymouth, MA, October 2000.

[19] T. Zhang and C. Kuo, "Content-based Classification and Retrieval of Audio," SPIE's Annual Meeting-Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations VIII, San Diego, July 1998.

노 승 민

2001년 아주대학교 정보 및 컴퓨터 공학과(학사). 2003년 8월 아주대학교 정보통신 전문대학원(석사). 2003년 9월~현재 아주대학교 정보통신 전문대학원 박사과정. 관심분야는 데이타베이스, 멀티미디어 시스템, XML 응용, DRM

박 동 문

2001년 아주대학교 정보 및 컴퓨터 공학과(학사). 2001년~2003년 아주대학교 정보통신 전문대학원 석사. 2003년~현재 모빌링크텔레콤 무선통신연구소 주임연구원. 관심분야는 데이타베이스, 모바일 데이타베이스, 데이타동기화, 유무선통합 시스템

황 인 준

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 정보통신 전문대학원 조교수. 2003년~현재 아주대학교 정보통신전문대학원 부교수. 관심분야는 데이타베이스, 멀티미디어 시스템, 정보 통합, 전자 상거래, XML 응용