멜로디 접근 빈도를 이용한 오디오 검색 시스템의 설계 및 구현

박 동 문°, 황 인 준
아주대학교 정보통신전문대학원

Fast Audio Retrieval For Frequently Accessed Melodies

Dongmoon Park, Eenjun Hwang
Graduate School of Information and Communication, Ajou University
E-mail : {minerva, ehwang}@ajou.ac.kr

요 약

웹의 급속한 발전과 함께 오디오 컨텐츠의 양이 증가하게 되었고 수많은 오디오 데이터베이스로부터 특정한 오디오 컨텐츠를 찾아내는 오디오 검색 시스템의 필요성이 증가했다. 본 논문에서는 효율적인 오디오 검색 시스템을 위해 사용자들이 질의하는 멜로디 패턴을 이용해서 FAI(Frequently Accessed Index)라는 새로운 인덱싱 메카니즘을 제안하고 이 FAI를 기반으로 구현한 프로토타입 시스템의 구조와 원리에 대해서 설명한다. 실험을 통해서 FAI 기반 오디오 검색 시스템의 성능을 일반적인 오디오 검색 방법과 비교해 보았다.

1. 서론

일반적으로 내용 기반 오디오 검색은 MIDI 키보드, 휘파람, 허밍 등을 이용해서 질의에 필요한 멜로디를 입력받는다. MIDI 키보드를 이용하면 정확한 질의를 작성할 수 있는 반면 MIDI 키보드 조작에 익숙하지 못한 사용자에게는 질의 작성이 쉽지않다. 휘파람이나 허밍을 이용하면 질의를 쉽게 표현할 수 있지만 검색 시스템이 질의 멜로디의 의도를 정확히 파악하지 못하는 경우가 생길 수 있다. 본 논문에서는 이런 질의 인터페이스의 문제로부터 발생하는 부정확한 검색은 어렵지않게 해결되리라고 가정하고, 오디오 정보를 효율적으로 인덱싱 함으로써 보다 빠른 검색하는 방법에 초점을 맞추었다.

대부분의 오디오 검색 시스템은 각 오디오 컨텐츠의 특징을 뽑아낸 다음 그 특징들을 가공해서 오디오 특징 데이터베이스(Audio Feature Database)에 넣어두고 그 특징을 실제 오디오 데이터베이스의 인덱스로 쓰는 방법을 사용한다. 이 때 가장 많이 사용되는 음향 특징(Acoustic Feature)들로는 음색(Timbre), 음향(Pitch) 등이 사용된다. 오디오 특징 정보는 시그널 프로세싱을 이용해서 뽑아내는 경우가 있고 MIDI 파일 안에 들어있는 오디오 정보를 이용하는 경우가 있다. 시그널 프로세싱을 이용하면 모든 오디오 파일 포멧에 적용할 수 있는 반면 구현이 어렵고 검색처리 시간을 길게 만들 수도 있어서 보통 오디오 검색 시스템은 MIDI 파일을 이용하는 방법을 주로 사용한다.

질의 멜로디와 유사한 오디오 컨텐츠를 찾기위한 방법인 UDR 스트링 비교 기법을 소개하겠다. UDR

스트링은 다음과 같이 오디오 정보를 분석해서 표현한다:

- U (Up) : 현재 음표의 음높이(Pitch)가 전 음표의 음높이보다 높을 경우.
- D (Down) : 현재 음표의 음높이가 전 음표의 음높이보다 낮을 경우.
- R (Repeat) : 현재 음표의 음높이가 전 음표의 음높이와 같을 경우.

모든 오디오 컨텐츠는 미리 UDR 스트링으로 변환되어 UDR 정보 데이터베이스에 넣어둔다. 사용자의 질의 멜로디가 들어오면 그 멜로디에 대한 UDR 스트링을 만들고 UDR 정보 데이터베이스의 내용과 비교해서 유사한 UDR 스트링을 가진 오디오 컨텐츠를 찾는 것이다. UDR 스트링은 텍스트로 표현한 정보이기 때문에 단문 정보검색(Sub-string Matching) 기술이 사용된다. 이 때 고려해야 할 것은, 사용자의 질의가 그들의 기억에 의존해서 만들어지는 멜로디이기 때문에 원곡의 UDR 스트링과 조금 다를 수 있다는 것이다. 따라서 사용자의 부정확한 질의 멜로디 때문에 정확(Exact) 검색 보다는 유사 검색(Approximate) 방법을 이용하는게 일반적이다[3].

본 논문에서는 FAI(Frequently Accessed Index) 기반의 새로운 내용 기반 오디오 검색 시스템을 제안한다. 이 기법은 사용자들의 질의로 들어오는 멜로디 대부분이 유사한 패턴을 보이는 것에서 착안한 것이다. 즉, 대부분의 사람들이 어떤 노래에 대해서 기억하는 부분은 그 노래의 메인 멜로디(반복되는 구간)이거나 클라이맥스 부분으로 제한된다. 따라서 그들이 작성하는 질의 멜로디 역시 자신이 기억하는 그 부분을 이용하는 것이다. 결국, 이렇게 자주 질의되는 멜로디의 위치를 저장해 두고 사용자의 질의가 들어올 때마다 그 멜로디에 대한 접속회수를 증가시키면, 가장 많이 검색되는 멜로디를 알 수 있게된다. 이 정보를 가지고 있다면 오디오 검색 시 무조건 데이터베이스 전체를 검색하는게 아니라 접속회수가 많은 부분을 먼저 검색하는 방법을 이용함으로써 검색 속도의 향상을 기대할 수 있다.

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 기반으로한 오디오 검색과 보통 방법을 사용한 오디오 검색을 실험을 통해서 비교해 보겠으며, 마지막으로 결론 및 향후 연구 방향에 대해 언급하겠다.

2. 관련연구

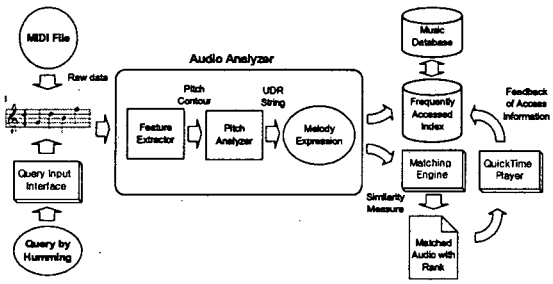
내용 기반 오디오 검색시스템들의 패러다임을 일컬어서 소위, QBH(Query-By-Humming) 혹은 WYHIWYG(What You Hum Is What You Get)이라고 한다[5]. Ghias [2]는 내용 기반 오디오 검색 분야에 대한연구 중에 선구적으로 꼽히는 논문으로써, 마이크로 폰을 이용해서 사용자의 허밍을 받고 그 허밍의 음높이 변화(Pitch Contour)를 UDR 스트링으로 표현하고 오디오 데이터베이스의 컨텐츠와 비교해서 멜로디를 찾아내는 시스템을 제안했다. McNab[4]은 MELody inDEX라는 웹기반 오디오 검색 시스템을 개발했다. 멜로디의 유사한 정도에 따라서 후보 멜로디를 리스트로 보여주고 재생해볼 수 있도록 했다. 오디오 검색은 멜로디의 음향, 간격, 리듬을 이용했고 멜로디 UDR 스트링 매칭은 다이나믹 프로그래밍 알고리즘을 사용해서 유사 검색 기법을 구현했다. Wold[6]는 시그널 프로세싱을 이용해서 음의 크기(Loudness), 높이(Pitch), 밝기(Brightness), 대역폭(Bandwidth), 조화도(Harmonicity) 등의 음향 특징을 뽑아내고 이 특징을 백터로 표현해서 대상 오디오의 음향 특징 정보와 질의 오디오의 특징 백터 간 거리(Vector Distance)를 비교하는 방법을 써서 오디오 컨텐츠를 분류했다. Foote[1] 역시 시그널로부터 오디오 특성을 뽑아내는 방식을 사용했고 유사도 매칭에 있어서 일반적으로 사용되는 음높이 변화나 스팩트럼 변화를 이용하지 않고, 트리 기반의 백터 양자기(Quantizer)를 이용해서 계산시간을 줄이는

시도를 했다.

3. 오디오 검색 시스템

3.1 시스템 구현 환경

본 시스템은 웹 기반의 FAI를 이용한 내용 기반 오디오 검색 프로토타입이다. Windows 98/NT 환경에서 개발되었고 자바로 구현되었기 때문에 이식성과 플랫폼 독립성을 가진다. 오디오 컨텐트는 모두 MIDI 파일을 사용했고, MIDI 파일을 처리하는 기능을 제공하는 jMusic[7] 라이브러리를 이용해서 오디오 음향 특징 정보를 추출했다.



[그림 1] FAI 기반 오디오 검색 시스템의 구조

3.2 시스템 구조

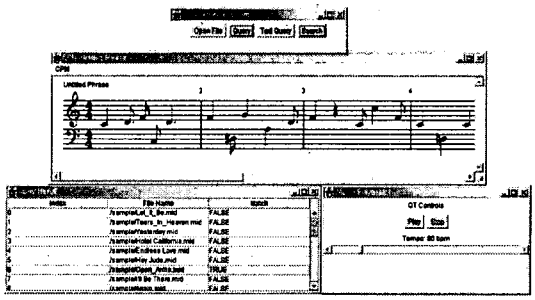
그림 1은 구현한 프로토타입 시스템의 대략적인 구조이다. 본 시스템은 크게 오디오 데이터베이스, 오디오 질의 입력 처리기, 오디오 분석기, 오디오 매칭 엔진 그리고 오디오 재생기의 세부 컴포넌트로 나누어 진다:

■ 오디오 데이터베이스(Audio Database)

사용자들로부터 질의를 받아들이기 전에 오디오 컨텐츠들을 저장해 두고 미리 UDR 스트링으로 변환해서 실제 오디오 컨텐츠를 포인팅하는 링크와 함께 따로 저장해 둔다.

■ 오디오 질의 입력 처리기(Audio Query Input Interface)

사용자의 질의는 찾고자하는 멜로디를 오선지에 기입하는 CPN(Common Practice Notation)기반의 방식을 사용해서 받아들인다. 사용자가 작성한 질의 멜로디는 MIDI 포멧으로 변환이 되어 저장된다. 이와는 다르게, 미리 작성된 샘플 MIDI 파일을 질의로 입력하는 것도 가능하다.



[그림 2] 질의 입력기와 결과리스트 및 오디오 재생기

MIDI 포멧의 질의 파일은 오디오 분석기(Audio Analyzer)로 넘어가서 음향 특징을 추출하고 UDR 스트링으로 표현하는데 사용된다. 오디오 분석기는 크게 두 가지 모듈을 거친다:

■ 음향 특징 추출이기(Acoustic Feature Extractor)

여러가지 음향 정보 중에서 음높이(pitch) 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 추출이기가 저장한 음높이값을 하나씩 비교하는 것이 이 분석기의 기능이다. 앞 음표의 음높이과 현재 음표의 음높이를 비교해서 현재 음표의 음높이가 상대적으로 높으면 U, 낮으면 D, 같으면 R로 표시하는 방식으로 UDR 스트링을 생성한다.

결국, 사용자의 질의 멜로디는 오디오 분석기를 거쳐서 UDR 스트링으로 변환된다. 이 UDR 스트링은 매칭엔진(Matching Engine)으로 보내진다:

■ 오디오 매칭 엔진(Audio Matching Engine)

오디오 매칭 엔진은 미리 UDR 스트링으로 변환되어있는 오디오 컨텐츠와 질의 UDR 스트링을 비교한다. UDR 스트링끼리의 비교에는 사용자 질의의 오류를 허용하기 위해 유사검색을 이용할 수 있다. 유사검색(Approximate Matching)은 질의 UDR 스트링

의 어느 부분 값을 하나씩 변화 시켜가면서 비교하는 방법인데, 만약 질의 멜로디가 길거나 오디오 데이터베이스의 컨텐츠 수가 많으면 시스템 성능저하를 가져올 수 있기 때문에 본 시스템에서는 정확 검색(Exact Matching) 방법을 사용했다.

■ 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 이용해서 재생해 볼 수 있다.

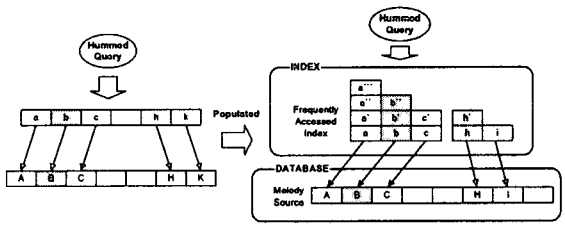
3.3 FAI 기반의 다이나믹 인덱싱 메커니즘

앞에서 언급했다시피, 어떤 노래에 대해서 사람들이 기억하는 부분은 반복되는 구간 혹은 클라이맥스이다. 따라서 본 시스템은 사용자가 A 라는 곡을 찾기 위해 a 라는 질의 멜로디를 작성하면, 이 질의 멜로디 a 의 위치를 따로 저장하고 이 멜로디에 대한 접근 회수를 알아내기 위해 하나의 접근회수(Access Count) 변수를 초기화 한다. 만약 다른 사람이 A 곡을 찾기 위해 a 와 같은 멜로디로 질의를 한다면 이 노래에서 멜로디 a 라는 부분에 대한 접근회수 값을 높여주는 것이다. 이후 사용자의 질의가 많이 있었을 때, 이 접근회수의 값이 특정 임계값 이상이 되면 이 멜로디의 내용과 멜로디를 포함하는 오디오를 포인트하는 링크를 FAI(Frequently Accessed Index)라는 인덱스 안에 저장한다. 그 다음부터 사용자 질의를 받고 이 멜로디 질의를 찾기 위해 오디오 데이터베이스를 검색을 할 때는 모든 오디오 컨텐츠의 UDR 스트링을 검색하는게 아니라 FAI를 먼저 검색하고, 질의 멜로디가 FAI 엔트리 안에 없을 경우에는 나머지 부분을 검색하게 된다.

FAI는 사람들이 가장 많이 질의하는 멜로디의 UDR 스트링을 모아둔 것이기 때문에, 이 FAI 엔트리 안에서 사용자의 질의 멜로디가 검색될 가능성이 많아 모든 오디오 컨텐츠를 검색해야하는 경우를 줄여준다. 이 FAI 기반 오디오 검색은 사용자의 질의 패턴를 이용하기 때문에 사용자의 질의가 회수가 많아질수록 축적되는 정보가 증가하고 정확해진다.

3.4 FAI 인덱스 관리

3.4.1 FAI 인덱스의 생성과정



[그림 3] FAI 기반 인덱싱 메커니즘

그림 3은 본 논문에서 제안하는 FAI 기반 인덱싱 구조를 개괄적으로 보여주는 그림이다. 그림 3의 왼쪽은 FAI의 생성 단계을 보여준다. FAI 생성 초기에는 FAI가 비어있는 상태로 존재한다. 사용자의 질의가 들어오면, 그 멜로디를 오디오 데이터베이스에서 찾고 부합하는 오디오 컨텐트가 있을경우, FAI 엔트리에 오디오의 링크와 함께 질의 멜로디가 들어갈 공간을 하나 할당해준다. 이 공간에는 질의 멜로디에 대한 정보뿐만이 아니라 접속회수도 포함이 되어있다. 다른 질의 멜로디가 오디오 컨텐트와 매치하면 새로운 FAI 엔트리 공간을 할당해 준다. 초기의 FAI 엔트리는 이런식으로 오디오 컨텐츠와 매칭되는 모든 질의 멜로디를 넣어준다.

사용자의 질의 회수가 증가하면 FAI 엔트리의 수도 비례해서 증가하고 축적되는 정보도 증가한다. 여러 사용자의 질의를 받다보면 한 노래에 대한 질의가 여러번 있을 수 있다. 이렇게 되면 그 노래를 가리키는 FAI 엔트리의 개수가 여러 개가 될 수도 있다. 하지만, 이때는 질의 멜로디를 무조건 FAI 엔트리에 넣는 것이 아니라, 기존의 FAI 엔트리와 비교해보고 질의 멜로디와 중복되는 것은 추가하지 않는다.

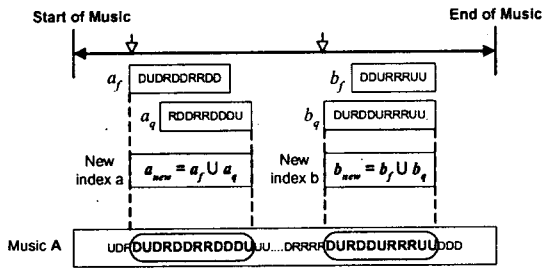
그림 3의 오른쪽은 이것을 설명하는 것이다. 오디오 데이터베이스 안에 있는 A라는 곡을 가리키는 멜로디가 a', a'', a''', a''''로 네 가지가 존재한다. 이것들은 A곡의 서브 UDR 스트링이고 a', a'', a''', a'''' 서로는 중복되지 않는다. 이들의 관계를 도식화하면 다음과 같다:

$i \neq j$ 에 대하여 $a_i \cap a_j = \varnothing$ 이고 $a_i, a_j \subset A$ 이다.

즉, 한 노래를 가리키는 FAI 엔트리는 여러 개가 될 수 있고 서로 중복되지 않으며, 그 노래에 대한 서브 UDR 스트링의 집합이라고 할 수 있겠다.

3.4.2 FAI 인덱스 엔트리의 확장

매칭엔진이 FAI 엔트리와 질의 멜로디를 비교할 때, 질의 멜로디가 FAI 엔트리 멜로디보다 조금 앞에서부터 시작하거나 멜로디가 조금 길어서 FAI 엔트리 멜로디보다 뒤에서 끝날 경우, 오디오 매칭 엔진은 이 두 멜로디가 같은 멜로디를 나타냄에도 불구하고 시작, 종료 위치가 약간 다르다는 이유로 매칭되지 않다는 결과를 보여주게 된다. 따라서, 본 시스템은 이점을 고치기 위해서 다음과 같은 FAI 엔트리 확장 규칙을 적용했다.



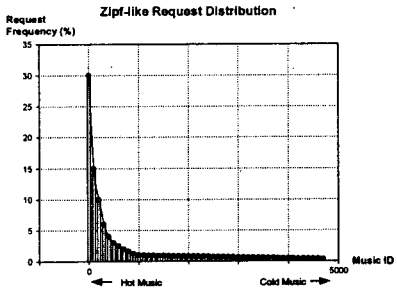
[그림 4] FAI 엔트리의 확장 규칙

그림 4에서 a_f 와 b_f 는 A곡에 대한 FAI 엔트리 멜로디이고, a_q 와 b_q 는 사용자의 새로운 질의 멜로디이다. 그림에서도 알 수 있듯이 이 멜로디들은 상당히 비슷하다; a_f 와 a_q 는 서로 포개지는 관계이고 b_f 와 b_q 는 b_q 가 b_f 를 포함하는 관계를 보여주고 있다. 이 a_q 와 b_q 질의는 각각 a_f 와 b_f 멜로디를 찾으려는 의도임에도 불구하고 시작점 혹은 종료 위치가 조금 틀리다는 이유로 오디오 매칭 엔진은 FAI 엔트리 안에서 찾지 못하고 모든 데이터베이스의 컨텐츠를 뒤지게 되는 것이다. 따라서, 오디오 데이터베이스 전체의 검색으로 찾은 질의 멜로디의 위치가 기존의 FAI 엔트리의 멜로디와 위치가 비슷할 경우 질의 멜로디와 FAI 엔트리를 합쳐서(Union Operation) 확장 한다. 결과적으로 a_f 와 a_q 는 a_{new} 로, b_f 와 b_q 는 b_{new} 로 확장된다.

4. 실험

이번 장에서는 인터넷에서 모은 5000 곡의 MIDI 파일을 이용해 본 시스템의 성능을 평가한 결과를 보여준다. 실험은 본 논문에서 제안하는 FAI을 기반한 오디오 검색과 FAI를 사용하지 않고 일반적인(데이터베이스를 모두 다 검색하는) 방법을 사용한 검색의 질의 처리 시간 비교해 보았다.

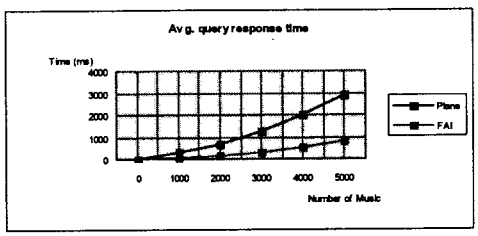
4.1 데이터 분석



[그림 5] *Zipf* 분산을 보이는 사용자 질의요청

본 오디오 데이터베이스는 5000 개의 다른 종류의 MIDI 파일들로 구성이 되어 있다: 평균적으로 한 MIDI 파일 당 5737 개의 음표를 가지고 있고 파일에 따라서 그 개수가 1000 개부터 10000 개까지 차이가 난다. 실험에 앞서 우리는 오디오 컨텐츠에 대한 사용자의 요청이 *Zipf* 분산을 따른다고 가정했다. 따라서 대부분의 오디오 컨텐츠에는 사용자의 접근이 드물고 상위 20%의 오디오 컨텐츠가 전체 사용자 질의 요청의 70% 가량 차지하도록 사용자 질의를 생성했다.

4.2 성능 분석



[그림 6] FAI 기반 오디오 검색의 성능 비교

본 실험에서는 오디오 데이터베이스에 오디오 컨텐츠의 수를 증가시켜가면서 질의 처리 시간을 측정했다. 그림 5는 사용자 질의에 대한 평균 질의처리 시간을 보여주는 것이다. 가파르게 증가하는 선이 FAI를 사용하지 않은 일반적인 오디오 검색 방법의 평균 질의처리 시간이고 완만하게 증가하는 선이 FAI를 이용해서 오디오를 검색한 질의 처리 시간이다. 두 실험 모두 컨텐츠의 수가 증가함에따라 처리 시간도 같이 증가했다. 하지만 FAI를 기반으로한 오디오 검색이 모든 데이터베이스를 검색하는 일반적인 방법보다 우월함을 알 수 있었고, 오디오 컨텐츠의 수가 증가할수록 FAI 기반 오디오 검색이 얻는 이득은 증가하는 것으로 나타났다. 본 실험 결과를 바탕으로 다음과 같은 공식을 유도할 수 있다:

P_{hit}: 질의 멜로디가 FAI 엔트리 안에서 발견될 확률

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

C_{FAI} : FAI 엔트리를 검색하는데 소요되는 계산

C_{Total}: 모든 데이터베이스를 검색하는데 소요되는 계산

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

사용자 질의 멜로디가 FAI 엔트리 안에서 발견될 확률이 그렇지 않을 확률 보다 훨씬 높고, FAI 엔트리를 검색하는데 소요되는 계산 비용은 모든 데이터베이스를 검색하는데 드는 계산 비용보다 적다. 대부분의 질의 멜로디가 짧은 시간 안에 FAI 엔트리에서 발견되고 그렇지 않을 경우에만 나머지 데이터베이스를 검색하기 때문에, 항상 모든 데이터베이스를 다 검색해야하는 일반적인 검색 방법보다 FAI 엔트리를 우선 검색하고 부합하는 엔트리가 없을경우에만 모든 데이터베이스를 검색하는 FAI 기반의 검색 방법이 더 효율적이라고 할수 있겠다.

5. 결론 및 향후 연구

본 논문에서는 사용자의 질의 패턴을 이용하는 오디오 검색 시스템에 대한 구조 및 원리에 대해서 설명했다. 사용자가 자주 질의하는 멜로디의 위치를 FAI라는 인덱스에 저장해 놓고 이 FAI를 먼저 검색하는 방법을 사용했다. 실험을 통해서 선형적으로 모든 데이터베이스를 검색하는 방법보다 FAI를 기반한 검색 방법이 짧은 질의처리 시간이 걸려 더 좋은 성능을 발휘함을 확인했다. 향후 연구 과제로는 질의 멜로디가 FAI 안에서 얼마나 많이 찾아지느냐가 이 시스템의 성능을 좌우하기 때문에, FAI 엔트리의 개수와 시스템 성능과의 관계를 통해 최적의 성능을 발휘하는 FAI 엔트리 개수를 찾는 연구가 남았다.

[참고문헌]

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] *jMusic*, Available at http://jmusic.ci.qut.edu.au/