허밍 질의 처리 시스템의 성능 향상을 위한 효율적인 빈번 멜로디 인덱싱 방법

(An Efficient Frequent Melody Indexing Method to Improve Performance of Query-By-Humming System)

유 진 희 † 박 상 현 ††
(Jinhee You) (Sanghyun Park)

요 약 최근 방대한 양의 음악데이타를 효율적으로 저장하고 검색하기 위한 방법의 필요성이 증대되고 있다. 현재 음악 데이타 검색에서 가장 일반적으로 쓰이는 방법은 텍스트 기반의 검색 방법이다. 그러나 이러한 방법은 사용자가 키워드를 기억하지 못할 경우 검색이 어려울 뿐만 아니라 키워드와 정확하게 일치하는 정보만 검색해 주기 때문에 유사한 내용을 가진 정보를 검색하기에 부적절하다. 이러한 문제점을 해결하기 위해 본 논문에서는 내용 기반 인덱싱 방법(Content-Based Indexing Method)을 사용하여 사용자가 부정확한 멜로디(Humming)로 질의하였을 경우라도 원하는 음악을 효율적으로 찾아주는 허밍 질의 처리 시스템(Query-By-Humming System)을 설계한다. 이를 위해 방대한 음악 데이타베이스에서 한 음악을 대표하는 의미 있는 멜로디를 추출하여 인덱싱하는 방법을 제안한다. 본 논문에서는 이러한 의미 있는 멜로디를 사용자가 자주 질의할 가능성이 높은 멜로디로서 하나의 음악에서 여러 번 나타나는 빈번 멜로디와 긴 쉼표 후에 시작되는 쉼표 단위 멜로디로 정의한다. 실험을 통해 사용자들이 이들 멜로디를 자주 질의한다는 가정을 증명하였다. 본 논문은 성능 향상을 위한 3가지 방법을 제안한다. 첫 번째는 검색 속도를 높이기 위해 인덱스에 저장할 멜로디를 문자열 형태로 변환한다. 이때 사용되는 문자 변환 방법은 허밍에 포함된 에러를 허용한 방법으로써 검색 결과의 정확도를 높일 수 있다. 두 번째는 사용자가 자주 질의할 가능성이 높은 의미 있는 멜로디를 인덱싱 하여 검색 속도를 높이고자 한다. 이를 위해 신뢰도가 높은 의미 있는 멜로디를 생성하는 빈번 멜로디 추출 알고리즘과 쉼표 단위 멜로디 추출 방법을 제안한다. 세 번째로는 정확도를 향상시키기 위한 3단계 검색 방법을 제안한다. 이는 데이타베이스 접근을 최소화하여 정확한 검색 결과를 얻기 위하여 제안되었다. 또한 기존의 허밍 질의 처리 시스템의 대표적인 인덱싱 방법으로 제안되었던 N-gram 방법과의 성능 비교를 통해 본 논문이 제안하는 방법의 성능이 보다 더 향상되었음을 검증하였다.

키워드 : 멀티미디어 데이타베이스, 허밍 질의 처리 시스템, 음악 정보 검색, 내용 기반 검색, 인덱싱

Abstract Recently, the study of efficient way to store and retrieve enormous music data is becoming the one of important issues in the multimedia database. Most general method of MIR (Music Information Retrieval) includes a text-based approach using text information to search a desired music. However, if users did not remember the keyword about the music, it can not give them correct answers. Moreover, since these types of systems are implemented only for exact matching between the query and music data, it can not mine any information on similar music data. Thus, these systems are inappropriate to achieve similarity matching of music data. In order to solve the problem, we propose an Efficient Query-By-Humming System (EQBHS) with a content-based indexing method that efficiently retrieve and store music when a user inquires with his incorrect humming. For the purpose of accelerating query processing in EQBHS, we design indices for significant melodies, which are 1) frequent melodies occurring many times in a single music, on the assumption that users are to hum what they can easily remember and 2) melodies partitioned by rests. In addition, we propose

· 본 연구는 문화관광부 및 한국문화콘텐츠진흥원의 문화콘텐츠기술연구소(CT) 육성사업의 연구결과로 수행되었음
† 학생회원 : 연세대학교 컴퓨터과학과
jhyou@cs.yonsei.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
sanghyun@cs.yonsei.ac.kr
논문접수 : 2006년 10월 24일
심사완료 : 2007년 5월 14일

an error tolerated mapping method from a note to a character to make searching efficient, and the frequent melody extraction algorithm. We verified the assumption for frequent melodies by making up questions and compared the performance of the proposed EQBHS with N-gram by executing various experiments with a number of music data.

Key words : Multimedia Database, Query-By-Humming System, Music Information Retrieval, Content-Based Searching Method, Indexing

1. 서 론

최근 멀티미디어 정보의 활용이 일반화되면서 방대한 양의 멀티미디어 정보를 효율적으로 저장하는 방법들의 필요성이 증대되고 있다. 또한 사용자들은 기존의 멀티미디어 정보 검색 방법들보다 더욱 쉽고 빠른 방법을 필요로 하고 있다. 이와 같은 사용자들의 요구에 맞추어 현재 멀티미디어 정보를 효율적으로 저장하고 빠르게 검색하는 방법에 관한 연구가 활발히 진행되고 있다.

멀티미디어 정보 검색 방법 중 기존의 가장 일반적인 방법으로 텍스트 기반 검색 방법(Text-Based Retrieving Method)이 있다. 이 방법은 사용자가 미리 선정해놓은 키워드를 사용하여 원하는 정보를 검색하는 방식이다. 그러나 이는 사용자가 키워드를 기억하지 못할 경우 검색이 어려울 뿐만 아니라 키워드와 정확하게 일치하는 정보만 검색할 수 있기 때문에 유사한 내용을 가진 정보는 검색하기 어렵다. 특히 음악 정보 검색(Music Information Retrieval)에서 가장 일반적으로 사용되는 검색 방법도 마찬가지로 텍스트 기반 검색 방법이다. 이는 사용자가 음악의 제목, 가수 명, 앨범 명 등과 같은 키워드 형식의 메타 정보를 사용하여 원하는 음악을 검색하는 방식이다. 그러나 사용자가 위와 같은 메타 정보를 기억하지 못할 경우 검색이 어려울 뿐만 아니라 키워드를 알고 있더라도 유사한 멜로디를 가진 음악을 검색하는데 어려움이 발생하게 된다.

이러한 문제점을 해결하기 위해서 제안된 방법으로 멀티미디어 데이타의 내용을 사용하여 검색하는 내용 기반 검색 방법(Content-Based Searching Method)이 있다. 이는 음악 정보 검색 뿐만 아니라 멀티미디어 정보 검색과 관련된 모든 분야에서 큰 이슈가 되고 있다[1]. 특히, 음악 정보 검색에서 사용되는 내용 기반 검색은 주어진 음악의 내용에 해당하는 멜로디를 기반으로 검색을 수행하는 방법이다[2]. 이는 기존의 텍스트 기반 검색과 달리 내용을 직접 비교하여 사용자가 원하는 내용을 가진 결과 값을 얻을 수 있다. 또한 내용의 유사성을 파악할 수 있기 때문에 유사한 멜로디를 검색하는데 적합하다. 음악 정보 검색에서 대표적인 내용 기반 검색의 연구 분야로는 허밍 질의 처리(Query-By-Humming) 방식이 있다[3]. 이 방식은 사용자가 부른 허밍(Humming)을 질의로 사용하여 이와 유사한 멜로디를 포함하고 있는 음악을 얻어내는 것이다. 이러한 허밍 질의 처리 방식의 장점은 정확하지 않은 멜로디로 질의하였을 경우에도 검색이 가능하기 때문에 노래 실력이 없는 사람들이나 언어 장애를 가진 사람들도 효과적으로 음악을 검색할 수 있다는 것이다. 그러나 기존의 허밍 질의 처리 방식의 단점은 방대한 음악 데이타베이스에서 비슷한 멜로디를 찾는 작업이 복잡한 계산과 오랜 검색 시간을 요구한다는 것이다. 이는 데이타베이스에 저장된 방대한 크기의 멜로디들과 질의 사이의 유사성을 판별하기 위해서 숫자 데이타를 사용하여 순차적으로 거리를 구해야 하기 때문이다. 따라서 본 논문은 내용 기반 검색 방법을 사용하면서 발생되는 검색 비용을 줄이기 위해 숫자 형태로 표현된 멜로디를 문자 형태로 변화하여 검색하는 방법을 사용한다. 이러한 문자 변환 방법은 사용자의 질의에 포함될 수 있는 에러를 허용(False Tolerance)함으로써 정확도를 보장해 준다. 또한, 본 논문은 검색 시간을 줄이기 위해 의미 있는 멜로디를 추출하여 인덱싱하는 효과적인 내용 기반 인덱싱 방법(Efficient Content-Based Indexing Method)을 제안한다. 여기서 의미 있는 멜로디란 사용자가 자주 질의할 가능성이 높은 멜로디이다. 본 논문에서는 의미 있는 멜로디를 두 가지로 분류하였는데 첫 번째 멜로디는 하나의 음악에서 여러 번 나타나는 빈번 멜로디이고 두 번째 멜로디는 긴 쉼표 후에 시작되는 쉼표 단위 멜로디이다. 본 논문은 신뢰도가 높은 의미 있는 멜로디를 추출하기 위해 효과적인 빈번 멜로디 추출 방법과 쉼표 단위 멜로디 추출 방법 제안한다.

현재 허밍 질의 처리 시스템의 인덱싱에 관련된 연구 활동 중 대표적인 방법으로 N개의 음표가 하나의 단위가 되도록 하여 이들을 인덱싱하는 N-gram방법이 있다[4-6]. 이 방법은 인덱싱이 쉽다는 장점을 가지고 있지만 음악의 모든 멜로디를 저장하기 때문에 사용자의 질의와 유사한 부분을 검색하기 위해서는 저장된 모든 멜로디를 차례로 비교해야 한다는 단점이 있다. 또한 정확도 측면에서 좋은 검색 결과를 보장하지 못한다는 단점도 가지고 있다. 이는 N-gram에서 사용한 특징 데이타를 문자 형태로 변환할 때 부정확한 멜로디라는 허밍의 특징을 고려하지 않고 에러 허용 없이 변환하므로

결국 사용자가 부른 허밍의 부정확한 정도가 클수록 검색 결과의 정확도가 급속도로 떨어지기 때문이다. 본 논문은 이러한 기존의 N-gram 방법과 성능 비교를 통해 본 논문이 제안하는 인덱스 방법이 N-gram방법보다 향상된 성능을 보임을 증명한다.

본 논문에서 제안하는 시스템은 크게 두 단계로 나뉘어 진다. 첫 번째는 허밍 질의 처리 시스템 구축 단계이고 두 번째는 질의 처리 단계이다. 시스템 구축 단계에서는 미디(MIDI)형식의 음악을 입력 받아 특징 데이타를 추출한 후 이를 특징 데이타베이스에 저장하고, 3개의 인덱스를 생성한 후 특징 데이타를 문자 형태로 변환하여 인덱스에 저장함으로써 전체 시스템을 구축한다. 질의 처리 단계에서는 우선적으로 질의 전처리 과정을 통해 질의를 특징 데이타베이스에 저장된 데이타 형태와 인덱스에 저장된 형태와 같도록 변환하는 작업이 필요하다. 이러한 질의 전처리 과정 이후에는 3개의 인덱스와 데이타베이스를 단계별로 검색하는 3단계 검색 방법을 실행한다. 본 논문에서 제안하는 3단계 검색 방법의 목표는 특징 데이타베이스의 접근을 최소화하여 검색 속도를 향상시키는 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 통하여 기존의 음악 정보 검색에 사용되는 특징 데이타의 종류와 기존의 내용 기반 인덱싱 기법에 대해 기술한다. 3장에서는 허밍 질의 처리 시스템의 구축 과정을 단계별로 설명한다. 4장에서는 질의 처리 과정을 보이고 5장에서는 실험과 성능 평과 결과를 기술한다. 마지막으로 6장에서는 본 논문을 요약하고 결론을 기술한다.

2. 관련 연구

2.1 허밍 질의 처리 시스템

현재 내용 기반 음악 정보 검색 시스템에서 질의 특성에 따라 효과적인 특징 데이타를 찾는 연구가 활발히 진행되고 있다. 이는 음표의 정확한 높이와 길이를 가진 원 음악과 같은 정확한 질의를 처리하기 위한 것과 허밍과 같은 부정확한 질의를 처리하기 위한 것으로 나뉘어진다. 특히, 부정확한 질의를 처리하기 위한 대표적인 시스템으로 허밍 질의 처리 시스템이 있다[3,7]. 이 시스템에서 멜로디를 표현하기 위해 일반적으로 사용되는 특징 데이타로 'UDS' 문자열[8]이 있는데 이는 인접한 두 음표의 높이를 사용하여 뒤의 음표가 앞의 음표보다 높으면 'U(Up)', 낮으면 'D(Down)', 같으면 'S(Same)'로 표현한다. 그러나 이와 같은 데이타는 단지 음 높이의 곡선을 이용할 뿐 음의 길이는 무시한 정보이므로 정확도 측면에서 효율적이지 않다. 또 다른 허밍 질의 처리 시스템으로는 음악의 템포(Tempo)를 사용하여 검색하는 방법이 있다[9]. 그러나 템포 데이타도 많은 특징 정보를 가지고 있지 않기 때문에 부정확한 허밍을 처리할 경우 정확도가 떨어지게 된다. 이와 같은 단점을 보완하기 위해 부정확한 멜로디인 허밍 질의의 특징을 고려한 효과적인 특징 데이타를 사용해야 할 필요가 있다. 기존의 여러 특징 데이타들 중 인접한 두 음표의 높이 변화량(Pitch Interval)과 길이 변화율(Duration Ratio)을 사용하였을 경우 가장 정확한 검색 결과를 보인다는 연구 결과가 발표되었다[6]. 이는 음표의 높이와 박자를 모두 고려하였으며 멜로디의 변화 곡선이 아니라 변화의 정도를 정확한 수치로 표현하였기 때문이다. [6]의 연구 결과를 바탕으로, 본 논문에서는 허밍 질의 처리 시스템에 적합한 특징으로써 두 음표의 높이 변화량과 길이 변화율을 사용한다.

2.2 음악 인덱스 구조

음악 정보 검색과 관련된 또 다른 연구 활동으로는 효율적인 인덱스 구조에 관한 것이 있다. 이러한 연구가 활발한 이유는 방대한 음악 데이타베이스를 순차적으로 검색함으로써 발생하는 오랜 검색 시간과 방대한 비교 계산 량을 줄이기 위해서이다. 이를 보완하기 위한 기존의 대표적인 인덱스 방법으로 N-gram이 있다[10,11]. 이 방법은 N개의 음표들을 하나의 단위로 취급하여 한 음표씩 오른쪽으로 이동시키며 인덱싱하는 방법으로 이는 크기가 N인 슬라이딩 윈도우(Sliding Window)와 같은 개념이다. 이러한 인덱스 설계 방식으로 인하여 N-gram은 인덱스 크기가 크고 검색 비용 면에서 좋지 않은 성능을 보인다. 또한 검색 속도를 향상시키기 위해 숫자 형태의 특징 데이타를 문자열로 변환하지만 이는 부정확한 멜로디인 허밍의 특징을 고려하지 않은 문자열로서 질의가 부정확할수록 정확도가 급격히 떨어지게 된다. 또 다른 인덱스 방법으로는 공간 접근 방식(Spatial Access Method)을 사용하는 경우가 있다[12,13]. 이 방법은 한 음표의 높이와 길이 정보를 이용하여 이를 2차원 공간상의 하나의 점으로 표현한 후 연속된 점들의 위치 정보를 인덱싱하는 방법이다. 대표적으로 R-Tree를 사용하는 방법이 있는데, 이는 검색 속도가 빠르지만 질의를 포함한 최소 영역 사각형(Minimum Bounding Rectangle)안에 질의와 거리가 가까운 음악들이 많이 나타나기 때문에 정선도(Selectivity)가 좋지 않다는 단점을 가지고 있다[12].

2.3 내용 기반 멜로디 인덱싱 방법

음악 정보 검색에서 사용되는 내용 기반 인덱싱 기법은 음악의 특정 멜로디를 인덱싱하여 검색 속도를 줄이는 방법이 대부분이다[14-16]. 이는 한 음악에서 어떤 부분의 멜로디를 인덱싱하느냐에 따라 음악 검색 시스템의 성능이 달라지기 때문에 성능을 향상시킬 수 있는

의미 있는 멜로디를 선택하는 것이 중요하다. 대표적으로 음악의 시작 부분을 인덱싱 함으로써 검색 속도를 향상시킨 방법이 있다[14]. 이는 사용자가 음악의 시작 부분을 자주 질의할 가능성이 높다는 것을 이용하였다. 그러나 이 방법은 사용자가 음악의 시작 부분을 질의하지 않을 경우 빠른 검색 속도를 보장하지 못한다. 또한 사용자가 음악의 시작 부분을 자주 질의한다는 것에 대한 충분한 근거를 제시하지 못하였다. 또 다른 내용 기반 인덱싱 방법으로는 일정하게 반복된 멜로디를 인덱싱한 연구가 있다[15]. 그러나 이 방법은 일정한 단위 없이 반복 멜로디를 찾기 때문에 반복 멜로디 추출 과정이 너무 복잡하고 후보 빈번 멜로디가 많이 발생하여 인덱스 크기가 커지는 단점을 가지고 있다. 앞의 두 방법과는 반대로 데이타베이스에 저장된 음악을 인덱싱하지 않고 사용자의 질의를 인덱싱하는 방법이 있다[16]. [16]은 사용자 질의 멜로디가 입력되면 이를 'UDS' 문자열로 변환한 후 질의들을 서로 비교하여 자주 질의되는 멜로디를 인덱싱하는 방법이다. 이는 사용자가 자주 질의한 음악은 빠르게 검색해 주는 장점을 가지고 있지만 만약 자주 질의하지 않는 멜로디가 입력이 되면 데이타베이스에 직접 접근해야 하는 단점이 있다. 또한 음의 박자를 고려하지 않고 음 높이의 곡선만을 이용한 'UDS' 문자열 방법을 사용하였기 때문에 정확도 측면에서 낮은 성능을 나타낸다.

3. 허밍 질의 처리 시스템 구축

이 장에서는 그림 1과 같이 허밍 질의 처리 시스템을 구축하는 과정을 설명한다. 허밍 질의 처리 시스템을 구축하는 과정은 다음과 같다.

(1) 미디(MIDI)형태의 음악 파일을 입력받은 후 미디 음표(Note)의 높이(Pitch)와 길이(Duration), 쉼표(Rest)의 길이, 마디(Bar)의 경계를 추출한다.
(2) 추출한 음표들과 쉼표들의 높이와 박자정보를 사용하여 특징 데이타(Feature Data)를 추출한다. 본 논문에서 지정한 특징 데이타는 멜로디를 표현할 수 있는 연속된 음표와 쉼표들의 변화 수치이다.
(3) 생성한 특징 데이타를 특징 데이타베이스(Feature Database)에 저장한다.
(4) 인덱스를 생성하기 위해 특징 데이타를 문자 형태로 변환한다.
(5) 문자 형태로 변환한 후 의미 있는 멜로디를 추출한다. 의미 있는 멜로디란 사용자가 질의할 가능성이 높은 멜로디를 말한다.
(6) 추출한 멜로디를 서픽스 트리(Suffix Tree)에 저장하여 3개의 인덱스를 구축한다.

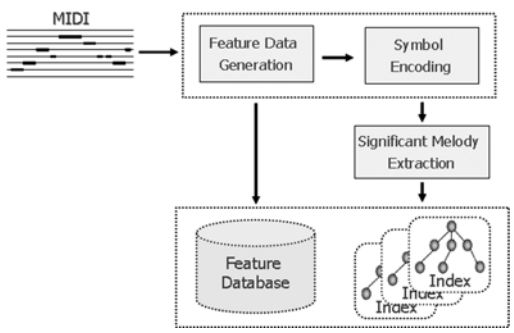

그림 1 허밍 질의 처리 시스템 구축 과정

3장은 다음과 같이 이루어져 있다. 3.1절에서는 특징 데이타 추출 과정과 저장 방법을 기술하고 3.2절에서는 특징 데이타를 문자 형태로 변환하는 과정에 대해 설명한다. 3.3절에서는 의미 있는 멜로디인 빈번 멜로디와 쉼표 단위 멜로디를 추출하는 과정에 대해 설명하고 3.4절에서는 인덱싱 과정에 대해 설명한다.

3.1 특징 데이타 추출과 저장

본 논문이 구축할 허밍 질의 처리 시스템은 데이타베이스에 저장된 멜로디와 사용자의 질의 멜로디를 서로 비교하여 사용자가 원하는 음악을 검색해주는 시스템이므로 멜로디를 추출하는 과정이 가장 우선적으로 진행되어야 한다. 이를 위해서 우선 미디 형식의 음악 파일들을 입력 받은 후 음표의 높이와 길이, 쉼표의 길이, 마디 경계를 추출해야 한다. 이들을 추출하는 이유는 연속된 음표와 쉼표들의 높이 변화량과 길이의 변화율이 멜로디를 표현하기에 가장 적절한 데이타이기 때문이다[6,17].

음표의 높이 정보를 얻기 위하여 미디에서 음의 높이를 미리 정의해 놓은 값으로써 옥타브(Octave)를 사용한다. 옥타브의 표기는 알파벳 형태의 음계와 숫자로 이루어져 있고 값의 범위는 0부터 127까지 이다. 예를 들어 옥타브가 'F5'인 음표는 그 높이 값이 '65'이다. 본 논문은 이러한 숫자 값을 사용하여 음표의 높이를 표현한다. 다음으로 음표와 쉼표의 길이를 추출한다. 미디 음악에서 음표의 길이는 4분 음표 즉, 한 박자의 길이(Time Base)를 기준으로 계산된다. 그러나 미디 음악마다 한 박자의 길이가 다를 수 있기 때문에 같은 음표라도 그 음의 길이는 음악마다 다른 값이 될 수 있다. 그러므로 모든 음악의 한 박자 길이를 동일한 값으로 변환해 주는 작업이 필요하다. 이와 같은 방법을 통해 모든 음표의 높이와 길이를 추출한다. 또한 이 단계에서 마디 경계를 추출하는데 이를 추출하는 이유는 한 음악에서 빈번하게 발생하는 빈번 멜로디를 발견할 때 일정한 길이를 가진 단위로서 마디를 사용하기 위해서이다. 마디의 경계는 입력된 미디 음악의 전체 박자 즉, 몇 분의 몇 박자 정보(Time Signature)와 한 박자의 길이,

그리고 음표와 쉼표의 길이를 통해 계산이 가능하다.

사용자가 입력한 허밍은 음표의 정확한 높이와 길이가 아닌 부정확한 멜로디이다. 다시 말해 사용자는 원 음악과 다르게 전체 음조를 높게 부르거나 낮게 부를 수 있고 원 음악보다 빠르게 부르거나 느리게 부를 수도 있다. 이러한 특징을 고려하여 정확도가 높은 음악 검색 시스템을 구축하기 위해서는 음표의 높이 변화량과 길이 변화율을 특징 데이타로 사용하는 것이 바람직하다. 음표의 높이 변화량이란 인접한 두 음표 사이에서 뒤의 음표의 높이와 앞의 음표의 높이의 차를 의미한다. 즉, 음표의 높이가 얼마만큼 증가, 감소했는지 또는 동일한지를 정확한 수치로 표현하는 것이다. 길이의 변화율은 인접한 두 음표 사이에서 뒤 음표의 길이를 앞 음표의 길이로 나누어 계산한다. 그림 2와 같이 두 개의 인접한 음표 사이에 2차원 벡터 형태의 특징 데이타를 계산하고 이 값을 앞 음표에 대한 정보로서 저장한다. 만약 인접한 두 음표를 N_i와 N_{i+1}이라면 N_i의 특징 데이타를 추출하기 위한 수식은 아래와 같다.

만일 연속된 음표 사이에 쉼표가 있다면 그림 3과 같이 처리한다. 많은 허밍 질의 처리 연구에서는 쉼표를 무시한 경우가 대부분이지만 본 논문에서는 사용자가 쉼표를 포함하여 불렀을 경우를 고려하여 쉼표 정보를 특징 데이타에 포함시킨다. 또한 일정한 길이 이상의 쉼표 후에 시작되는 멜로디를 질의할 가능성이 높다는 가정을 바탕으로 하였기 때문에 쉼표의 위치를 중요하게 취급한다. 쉼표는 음표와는 달리 길이 정보만 가지고 있다. 그렇기 때문에 쉼표의 높이를 널 값 (Null)으로 저장하고 그림 3과 같이 '#'으로 표현한다. 다음으로 쉼표의 특징 데이타를 추출하기 위해 높이의 변화량과 길이의 변화율을 계산한다. 이때 쉼표의 위치에 따라 두 가지의 경우로 나누어 처리할 수 있다. 첫 번째 경우는 음표 뒤에 쉼표가 나오는 경우이다. 이 때 계산되어야 할 특징 데이타는 앞 음표의 데이타가 되며 그 값은 그림 3과 같이 높이의 변화량은 변화가 없음을 의미하는 '0.00'으로 저장하고 길이의 변화율을 음표의 길이와 쉼표의 길이를 사용하여 '0.5'의 변화율로 계산함으로써 쉼표의 정보를 앞 음표의 정보에 포함시킨다. 이때 만일 쉼표 대신에 길이와 박자가 각각 67과 30인 음표가 이어진다면 위의 경우와 마찬가지로 (0.00, 0.5)라는 특징 데이타 값이 계산될 것이다. 이는 쉼표의 정보가 신뢰성이 없다는 문제점이 될 수 있지만 사용자가 허밍을 부를 경우 쉼표를 정확히 지키지 못하고 음표로 인식하여 음을 계속 이어서 부를 경우가 발생할 수 있기 때문에 위와 같은 두 가지 경우를 분리하지 않고 생각한다. 다음으로 그림 3과 같이 쉼표 뒤에 음표가 나온다면 이 때 계산되어야 할 특징 데이타는 쉼표의 데이타가 되는데 이는 무시한다. 그 이유는 무시된 길이 변화율은 멜로디를 표현하지 못하는 무음의 멜로디이기 때문이다. 지금까지 설명한 방법을 통해 하나의 미디 음악에서 연속된 2차원 벡터 형태의 특징 데이타들을 모두 생성한 후 그림 2와 같이 이들을 특징 데이타베이스에 저장한다.

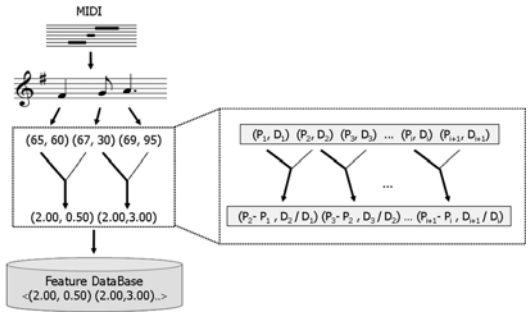

그림 2 특징 데이타 추출 과정

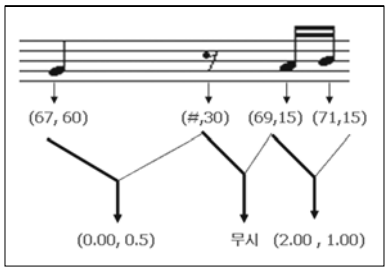

그림 3 쉼표 처리 과정

$$N_i = (P_i, D_i)$$
$$N_{i+1} = (P_{i+1}, D_{i+1})$$

$$Pitch\ val = P_{i+1} - P_i$$
$$Duration\ Ratio = D_{i+1} / D_i$$

3.2 문자 변환

2차원 벡터 형태로 특징 데이타가 저장된 데이타베이스를 검색할 경우 모든 음악을 순차적으로 비교하여 검색해야 한다. 이 때 데이타베이스에 저장된 2차원 벡터는 숫자 값으로 되어있고 검색을 위한 질의와의 비교 방법은 일반적으로 유클리디안 거리(Euclidean Distance) 계산 방법을 사용한다. 그러므로 이러한 검색 방법은 데이타베이스의 크기가 증가될수록 검색 속도가 느려지게 된다. 이를 보완하기 위해 특징 데이타를 문자열 형태로 변환하여 저장해 놓은 인덱스의 필요성이 대두되었다. 즉, 특징 데이타를 문자열로 변환하는 이유는 숫자 형태로 이루어진 2차원 벡터를 비교하는 것보다 문자로 바꾼 후 문자열 매칭(String Matching)알고리즘을 사용하는 것이 계산 복잡도를 줄일 수 있는 방법이기 때문이다.

2차원 벡터 형태의 숫자 값을 문자 값으로 변환하기 위해 그림 4와 같이 2차원의 공간을 35개의 구간으로

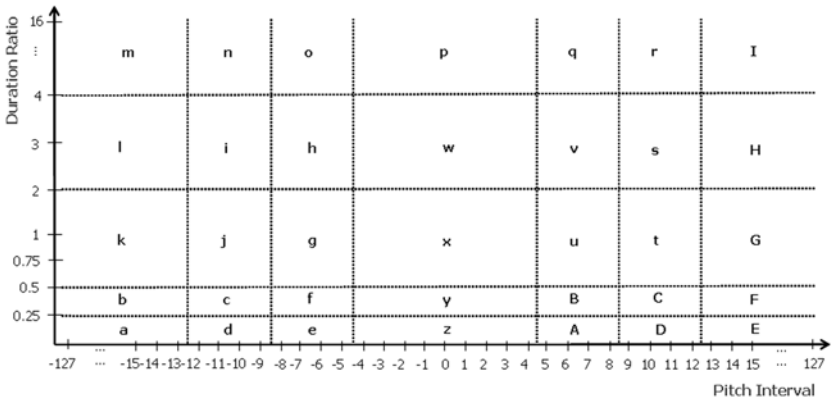

그림 4 문자 변환 기준의 예

나누고 각 구간을 하나의 문자로 할당하는 문자 변환 기준을 적용한다. 따라서 비슷한 2차원 벡터 값은 같은 문자로 치환되게 된다. 이는 허밍이 부정확하다는 특징을 반영한 것으로서 변환된 문자는 부정확한 허밍이 가지고 있는 에러를 허용한 값이 된다.

그림 4에서 X축은 높이의 변화량을 의미하고 Y축은 길이의 변화율을 의미한다. X축에서 0값을 기준으로 오른쪽은 높이의 변화량이 양수 값을 보인다. 이는 인접한 두 음표 사이에서 뒤 음표의 높이가 앞 음표의 높이보다 높은 경우를 의미하고 오른 쪽으로 갈수록 그 차이는 점점 더 커짐을 의미한다. 반대로 X축의 0값을 기준으로 왼쪽은 높이의 변화량이 음수 값을 보인다. 이는 뒤 음표의 높이가 앞 음표의 높이보다 낮음을 의미하고 왼쪽으로 갈수록 그 차이는 점점 커짐을 의미한다. 마찬가지로 Y축에서 1값을 기준으로 위쪽은 뒤 음표의 길이가 앞 음표의 길이보다 긴 경우이고 그 길이의 비율 차이는 위쪽으로 갈수록 점점 더 커짐을 의미한다. 반면 Y축의 1값을 기준으로 아래쪽은 뒤 음표의 길이가 앞 음표의 길이보다 짧은 경우이고 그 길이의 차이는 아래로 갈수록 점점 더 커짐을 의미한다. 위의 기준에서 -4와 +4 사이의 높이 변화량은 같은 문자로 변환되는데 이는 사용자가 실제 음악의 한 음을 부를 때 실제 음의 높이보다 4만큼 높게 부르거나 낮게 불러도 그만큼의 오차는 허용됨을 뜻한다. 마찬가지로 박자의 변화율에서 0.5와 2사이는 같은 문자로 변환된다. 이는 사용자가 실제 음악의 한 음을 부를 때 실제 길이보다 2배의 길이로 부르거나 0.5배의 길이로 불러도 그만큼의 오차를 허용한다는 의미이다. 그림 5는 위의 문자 변환 기준을 사용하여 연속된 2차원 벡터 값들을 문자로 변환한 결과이다.

문자 변환 기준은 2차원 구간의 크기에 따라 정확도가 달라진다. 그러므로 최적의 성능을 내는 문자 변환

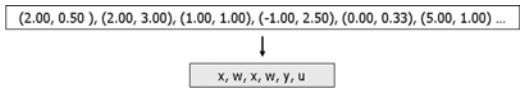

그림 5 문자 변환 결과

기준을 지정하여야 한다. 그러므로 본 논문은 5.5절에서 여러 가지 문자 변환 기준을 생성하여 실험을 통해 그 중 최적의 성능을 보이는 기준을 선택하였다.

3.3 의미 있는 멜로디 추출

한 음악은 여러 개의 멜로디로 이루어져 있다. 멜로디란 음악적인 표현과 인간의 감정을 가장 잘 나타내는 요소로서 다양한 음높이와 길이 정보를 담고 있는 음표와 쉼표의 결합을 의미한다. 이와 같이 한 음악을 이루는 여러 멜로디 중 그 음악을 대표할 수 있는 멜로디는 큰 의미를 갖게 된다. 본 논문에서는 의미 있는 멜로디를 2가지로 나누었다. 이들은 사용자가 자주 질의할 가능성이 높은 멜로디로서 인덱스를 생성하기 위해 사용된다.

3.3.1 의미 있는 멜로디 정의

우선 첫 번째 의미를 갖는 멜로디는 한 음악에서 여러 번 반복되어 나타나는 임의의 길이를 가진 멜로디이다. 이는 사용자가 음악을 들었을 경우 자주 반복하여 나타난 멜로디를 기억하기 쉽다는 가정을 토대로 하여 정의한 것이다. 즉, 사용자가 원하는 음악을 검색하기 위해서 그 음악의 멜로디들 중에 쉽게 기억할 수 있는 멜로디를 흥얼거릴 경우가 많다는 가정 하에 본 논문에서는 빈번하게 발생한 멜로디를 의미 있는 멜로디로 지정한다.

두 번째 의미를 갖는 멜로디는 일정 길이 이상의 쉼표 사이에 있는 멜로디이다. 일정 길이 이상의 긴 쉼표는 멜로디를 나누는 경계로 볼 수 있다. 실제 음악 데이타를 분석해본 결과 멜로디의 흐름이 긴 쉼표를 경계로 나뉘어진다는 것을 알 수 있었다. 대부분의 음악에서 긴

쉼표는 시작 부분 전, 후렴 부분 전, 2절이 시작되기 전 등에 많이 나타난다. 즉, 음악의 후렴 부분과 처음 부분을 사용자가 기억하기 쉬울 것이라는 가정 하에 쉼표의 의미를 적용함으로써 일정 길이 이상의 쉼표로 나누어지는 멜로디를 의미 있는 멜로디로 지정한다. 이와 같은 가정은 성능 평가 장에서 실제 사용자들의 허밍 질의를 대상으로 검증하였다.

3.3.2 빈번 멜로디 추출 과정

한 음악에서 여러 번 반복되는 멜로디를 찾기 위해 우선 마디 단위로 문자열 패턴 분석을 수행한다. 마디를 사용하여 빈번 멜로디를 추출하는 이유는 마디란 동일한 박자를 가지고 있는 물리적이고 객관적인 단위이므로 이를 기준으로 빈번 멜로디를 추출하면 정확하고 간결한 빈번 멜로디를 얻을 수 있기 때문이다. 문자열 패턴 분석을 수행하기 위해 문자열 형태인 하나의 음악을 그림 6과 같이 마디 경계인 '/ ' 로 나눈다. 마디 단위로 나눈 문자열들을 문자열 패턴 매칭 알고리즘을 통해 일치되는 문자열들을 분석하여 마디 안의 문자열, 마디 위치정보*(BarId)*, 일치되는 마디들의 개수*(C)* 그리고 *C*를 내림차순으로 정렬한 순위*(R)*를 얻는다. 예를 들어 '그 후로 오랫동안'이란 제목을 가진 음악을 입력하여 문자열을 마디 단위로 나눈 후 마디 단위 문자열 패턴 분석을 수행하면 표 1과 같은 결과를 얻을 수 있다. 표 1을 보면 문자열 <j, x, k, j, i, x>는 18, 22, 35, 39번째 마디에서 나타났고 발생 횟수 *C*는 4임을 확인할 수 있다. 또한 이 문자열은 위의 음악에서 가장 많이 발생한 것이므로 순위 *R*은 1이 된다.

마디 단위 문자열 패턴 분석을 모두 마친 후 다음 단계로 빈번 멜로디를 생성한다. 알고리즘 1은 한 음악에서 빈번 멜로디를 생성하는 과정을 나타낸다. 빈번 멜로디 생성의 첫 단계는 그룹을 생성하는 것이다 (단계 1). 빈번 멜로디 생성 알고리즘의 입력 데이타는 표 1과 같이 4개의 속성으로 구성된 리스트의 집합이다. 즉, 마디 단위로 문자열을 저장한 리스트, 마디 위치를 저장한 리스트, *C*를 저장한 리스트, *R*을 저장한 리스트를 모두 입력받은 후 *R*값이 1인 마디의 *C*값만큼 그룹을 생성한다. 그리고 *C*개의 *BarId*를 각 그룹의 대표 마디*(Key Bar)*로 선정한다(단계 2). 그림 7은 표 1의 데이타를 이용해 그룹을 생성한 결과이다.

표 1에서 *R*이 1인 마디의 *C*값은 4이므로 그림 7과 같이 4개의 그룹이 생성된다. 그림 7의 수직선은 하나의 음악을 의미하고 각 눈금은 하나의 마디를 의미한다. 또한 검은색이 칠해져 있는 눈금은 빈번 멜로디의 시작 마디인 각 그룹의 대표 마디를 의미한다. 그림 7과 같이 첫 번째 그룹의 대표 마디는 18번째 마디인 Bar18, 두 번째 그룹의 대표 마디는 Bar22, 세 번째 그룹의 대표 마디는 Bar35, 마지막으로 네 번째 그룹의 대표 마디는 Bar39이다. 이들 대표 마디를 기준으로 첫 번째 그룹부터 마지막 그룹까지 양 옆의 마디 중 *C*값이 높은 마디를 합쳐 나가면서 빈번 멜로디를 생성해 나간다.

본 논문에서는 빈번 멜로디를 생성하기 위해 2개의 변수가 필요하다. 첫 번째 변수는 어떤 마디를 빈번 멜

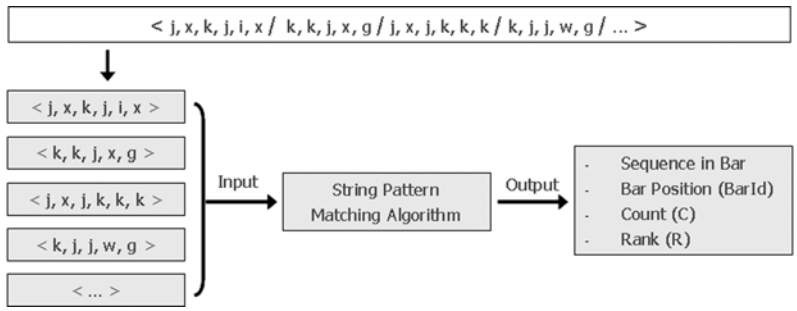

그림 6 마디 단위 문자열 패턴 분석

표 1 음악 '그 후로 오랫동안'의 마디 단위 문자열 패턴 분석 결과

Sequence in Bar	Bar Position (BarId)	Count (C)	Rank (R)
<j, x, k, j, i, x>	18, 22, 35, 39	4	1
<k, k, j, x, g>	19, 36	2	2
<j, x, j, k, k, k>	20, 37	2	2
<k, j, j, w, g>	21, 38	2	2
...	...	...	...
<j, x, g, k, E, x, i, x>	17, 49	1	3

알고리즘 1. 빈번 멜로디 생성 알고리즘

Input:	마디단위 문자열 리스트 *(Seq_list)*, 마디 위치를 저장한 리스트 *(BarId_list)*, 일치되는 마디 개수 리스트 *(C_list)*, 순위 리스트 *(R_list)*, 마디선택 임계값 *(BST)*, 빈번 멜로디 길이 임계값 *(FMLT)*
Output:	문자열 형태의 빈번 멜로디

1; *R*값이 1인 마디의 *C*값만큼 그룹을 생성한다. (그룹의 개수 = *C*)

2: *R*값이 1인 마디들을 각 그룹의 대표 마디 *(Key Bar)*로 선정한다.

3: 1부터 *C*까지의 각 I에 대하여 단계 4부터 단계 8까지를 차례로 실행한다.

4: 그룹 안의 대표 마디를 *KB*라 지정한다. 그리고 대표 마디의 양쪽 마디들을 각각 KB_{right}, KB_{left}이라 지정하고 현재 마디라 부른다. 또한 마디합*(BSum)*의 초기 값을 0으로 할당한다.

5: *BSum*이 빈번 멜로디 길이 임계값 *FMLT*이하이면 계속 진행하고, 초과하면 단계 3으로 돌아간다.

6: 현재 마디인 KB_{right}와 KB_{left}의 각 *C*값과 이미 정의된 *BST*값을 비교하여 *BST*값 이상의 *C*값을 가진 마디를 선택한 후 아래의 경우 중 한 가지를 수행한다. 그러나 두 개의 *C*값이 모두 *BST*값 보다 작으면 단계 3으로 돌아간다.
- 경우 1 : 현재 마디 중 하나의 마디가 선택되었다면 선택된 현재 마디의 문자열을 *KB*마디의 문자열과 합친다.
- 경우 2 : 두 개의 현재 마디가 모두 선택되었다면 각 *C*값을 서로 비교하여 큰 값을 가진 현재 마디의 문자열을 *KB*마디의 문자열과 합친다.
- 경우 3 : 두 개의 현재 마디가 모두 선택되었고 그 마디들의 *C*값이 같으면 두 개의 현재 마디 문자열을 *KB*마디의 문자열과 합친다.

7: 합쳐진 현재 마디의 *R*값과 *KB*마디의 *R*값의 차를 기존의 *BSum*에 더한다.

8: 현재 마디 중 오른쪽 마디가 합쳐졌다면 오른쪽의 다음 마디인 $KB_{right+1}$을 현재 마디로 지정하고 왼쪽 마디가 합쳐졌다면 왼쪽의 다음 마디인 KB_{left-1}을 현재 마디로 지정한 후 단계 5로 돌아가 계속 수행한다.

9: 그룹 개수만큼의 빈번 멜로디가 생성된 후, 겹쳐지거나 이웃하는 빈번 멜로디들이 있으면 이들을 하나고 합친다.

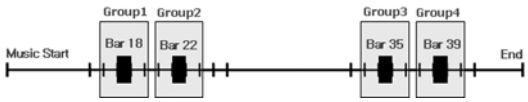

그림 7 그룹 생성

로디에 포함시킬지 결정하기 위한 선택 임계값 *BST* (Bar Selection Threshold)이다. 만일 *BST*값이 2라면 *C*값이 2이상인 마디들만 사용하여 빈번 멜로디를 생성함을 의미한다. 그러나 *BST*값이 작을수록 빈번 멜로디의 길이는 길어질 수 있다는 단점이 있기 때문에 두 번째 변수로서 빈번 멜로디의 길이를 결정하는 빈번 멜로디 길이 임계값 *FMLT*(Frequent melody Length Threshold)을 사용한다. 이는 빈번 멜로디를 생성하면서 마디가 추가될 때 마다 계산되는 마디 합(*BSum*)의 크기를 결정해 준다. *BSum*의 계산 방법은 대표 마디의 *R*값과 이전에 합쳐진 마디의 *R*값의 차를 계산하여 이전까지 계산된 *BSum*에 더해가는 방식이다. 이 때 *BSum*의 상한 값을 결정하기 위해 사용하는 것이 *FMLT*로서 *BSum*이 *FMLT*이하일 때까지 반복하여 빈번 멜로디를 생성해 나간다.

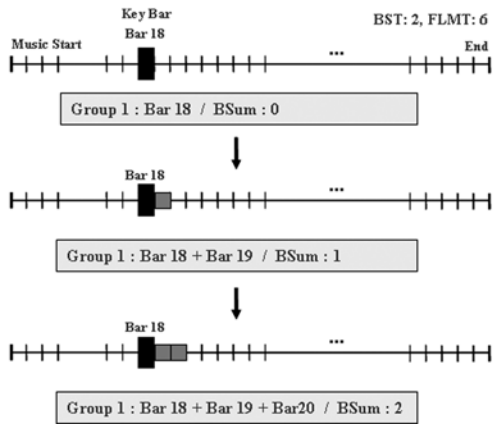

그림 8 첫 번째 그룹에서의 멜로디 추출 초기 단계

그림 8은 *BST*값과 *FMLT*값이 각각 2와 6으로 주어진 상황에서 표 1을 기반으로 각 그룹 안에서 빈번 멜로디를 생성하는 초기 단계를 그림으로 표현한 것이다. 우선 첫 번째 그룹을 보면 대표 마디인 *KB*가 Bar18로 지정되어 있다. 대표 마디의 양쪽 마디인 Bar17과 Bar19

는 각각 *KBleft* 와 *KBright*로 할당되어진다. Bar17의 *C*값은 1로서 *BST*값보다 작기 때문에 조건을 만족시키지 못하므로 무시한다. 다시 말해 Bar17은 충분히 여러 번 반복되어 나타난 마디가 아니기 때문에 대표 마디인 Bar18과 합쳐질 수 없다. 반면 Bar19의 *C*값은 2로서 *BST*값 이상이므로 Bar18과 합쳐진다. 마디를 합쳤다면 그 다음으로 그 그룹의 *BSum*을 계산한다. 합쳐진 Bar19의 *R*값은 2로서 대표 마디인 Bar18의 *R*값과의 차를 구하면 1이 된다. 이 값을 기존의 *BSum*값에 더하여 다시 *BSum*에 저장한다. 다음으로 Bar20의 *C*값을 *BST*값과 비교함으로써 위의 과정을 반복한다. 이때 대표 마디를 기준으로 왼쪽 방향은 Bar17의 *C*값이 *BST* 값 이상이 되지 못하였으므로 빈번 멜로디를 생성하는데 무시된다. 오른쪽 방향으로 마디를 증가시켜가며 *BSum*이 *FMLT*를 넘지 않을 때까지 또는 *C*값이 *BST* 값 보다 작은 마디가 나타날 때까지 빈번 멜로디를 생성하게 된다.

*BSum*을 계산하기 위해 마디 개수인 *C*값이 아닌 순위 *R*을 사용하는 이유는 다음과 같다. 표 2와 같은 정렬순서를 가진 두 개의 음악이 있고 *BSum*을 계산하기 위해 *R*값 대신 *C*값을 사용한다고 가정한 후 각각 빈번 멜로디 생성 알고리즘을 적용해 보자. 만약 *BST*값은 2 그리고 *FMLT*값은 6으로 지정해 놓는다면 음악 1의 첫 번째 그룹은 6번째 마디부터 13번째 마디까지 8개의 마디로 이루어진 빈번 멜로디를 생성하게 된다. 이는 음악 1에서 빈번하게 발생한 마디들을 최대한 고려하여 *BSum*이 *FMLT*를 초과하지 않을 때까지 빈번 마디를 합쳐나감으로써 빈번 멜로디를 생성한 것이다. 또한 이는 순위 *R*값을 사용하여 *BSum*을 계산하여도 같은 빈번 멜로디가 생성됨을 알 수 있다. 반면, *BSum*을 계산하기 위해 *C*값을 사용하여 음악 2에서 첫 번째 그룹의 빈번 멜로디를 생성해보면 4번째 마디부터 6번째 마디까지 3개의 마디로 이루어진 빈번 멜로디를 얻을 수 있다. 그러나 7번째 마디 또한 음악 2에서는 빈번하게 나타난 마디임에도 불구하고 빈번 멜로디로 취급되지 않았다. 즉, 빈번하게 발생한 마디를 최대한 고려하지 못하기 때문에 부정확한 빈번 멜로디를 생성하게 된다. 반면 *R*값을 사용하여 빈번 멜로디를 생성하면 3번째 마디부터 10번째 마디까지 8개의 마디로 이루어진 빈번 멜로디를 생성할 수 있다. 이는 음악 2에서 빈번하게 발생한 마디들을 최대한 고려하여 빈번 멜로디를 생성한 결과라 할 수 있다. 이들 두 음악은 각각 다른 특성을 갖는 음악이므로 각 음악에 나타난 빈번 마디의 횟수가 다를 수 있다. 음악 1에서 가장 빈번한 마디 개수가 3인 반면 음악 2에서는 6이다. 그러나 이들 마디 개수의 순위 *R*은 1로 동일하다. 즉, 각각 다른 마디 개수를 가진 음악들의 빈번 멜로디를 생성하기 위해서는 그들의 마디 개수를 객관적인 수치로 표현하기 위해 순위를 적용하였고, 이 값을 사용하여 *BSum*을 계산함으로써 빈번 멜로디를 생성해 나가는 것이다.

모든 그룹에서 빈번 멜로디 생성이 완료되면 각 그룹 안에는 빈번하게 발생한 마디들로 이루어진 빈번 멜로디가 생성되어 있을 것이다. 하지만 그림 9와 같이 두 개의 그룹이 겹쳐지는 상황이 발생할 수 있다. 각 그룹 안에 있는 빈번 멜로디를 각각 따로 저장한다면 중복된 데이타가 많이 발생할 것이다. 그러므로 이를 막기 위해 겹쳐지는 그룹들을 하나의 그룹으로 합침으로써 중복된 멜로디를 하나의 멜로디로 취급한다. 그림 9는 표 1을 사용하여 빈번 멜로디를 생성하는 마지막 단계를 보여준다. 첫 번째 그룹과 두 번째 그룹은 상당한 부분이 중복되어 저장되어 있다. 이렇듯 두 개의 그룹 안에 있는 데이타가 서로 중복 되거나 이웃하면 하나의 그룹으로 합친다. 그림 9의 마지막 단계를 보면 첫 번째 그룹과 두 번째 그룹이 하나의 그룹으로 합쳐진 것을 볼 수 있다. 이렇게 합쳐진 그룹안의 멜로디를 하나의 빈번 멜로디로 취급한다.

표 1의 예제를 사용하여 빈번 멜로디 생성을 모두 마치면 그림 10과 같이 2개의 빈번 멜로디를 얻을 수 있다. 빈번 멜로디의 문자열은 다음절에서 설명할 빈번 멜로디 인덱스에 저장되고 사용자가 이들 멜로디의 일부

표 2 두 개 음악의 마디 개수와 순위 비교

Music 1				Music 2			
Sequence in Bar	BarId	C	R	Sequence in Bar	BarId	C	R
c, c, a, d, d	6, 9, 20	3	1	a, e, e, s, d	4, 9, 15, 27, 30, 34	6	1
c, d, a, a, d	7, 21	2	2	c, e, s, a	5, 16, 28	3	2
a, d	8, 22	2	2	a, g, c, d	6, 10, 31	3	2
a, a, c, s	10, 23	2	2	e, s, c, c, d, s	3, 7, 11	3	2
b, e, h, z	11, 30	2	2	c, e, a	8, 12	3	2
b, b, k, u	12, 31	2	2	c, c, d	13, 14	3	2
c, c, a	13, 32	2	2	a, d, b, b	29, 31	3	2
...	...	...	...	...	...	...	...

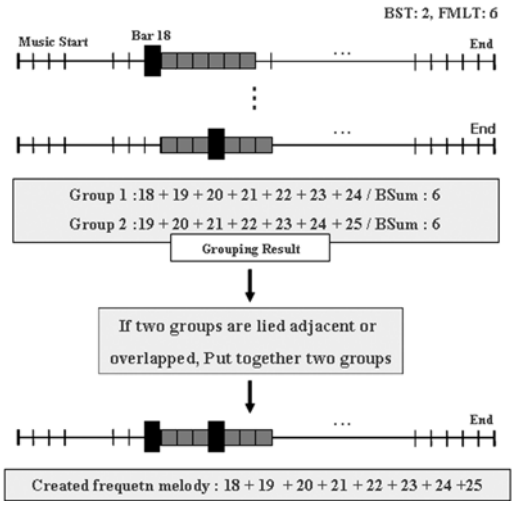

그림 9 중복된 멜로디 처리

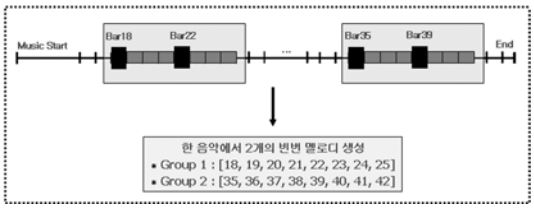

그림 10 빈번 멜로디 생성 결과

분을 질의하였다면 인덱스를 통해 빠른 시간 안에 검색이 수행될 수 있다.

3.3.3 쉼표 단위 멜로디 추출 과정

본 논문은 하나의 음악을 일정한 길이 이상의 쉼표로 나눔으로써 생성된 각각의 멜로디를 빈번 멜로디와 마찬가지로 의미 있는 멜로디로 정의한다. 허밍 질의 처리 시스템의 구축을 위한 초기 단계에서 쉼표의 길이 정보를 추출할 때 일정 길이 이상의 긴 쉼표의 위치 정보를 따로 저장해 놓았다. 그 위치를 경계로 하여 하나의 음악을 나눔으로써 여러 개의 쉼표 단위 멜로디를 생성한다. 여기서 일정 길이 이상의 쉼표란 멜로디의 연속된 흐름이 끊어지는 부분으로서 일정 길이를 어떤 값으로 선택하느냐에 따라 쉼표 단위 멜로디의 길이가 달라질 수 있다. 예를 들어 한 박자 즉, 4분 쉼표를 기준으로 멜로디를 나눈다면, 한 박자 이상의 길이를 갖는 쉼표가 나타난 곳을 경계로 멜로디를 나눈다. 그림 11은 짧은 멜로디를 한 박자 이상의 쉼표로 나누어본 예제이다. 이때 한 박자의 쉼표는 앞 음표의 특징 데이타를 생성하는데 사용되고, 쉼표의 특징 데이타는 무시되면서 단지 그 위치 정보만 저장한다. 그리고 문자 형태로 변환한 후 저장해 놓은 쉼표 위치를 경계로 멜로디를 나눈다.

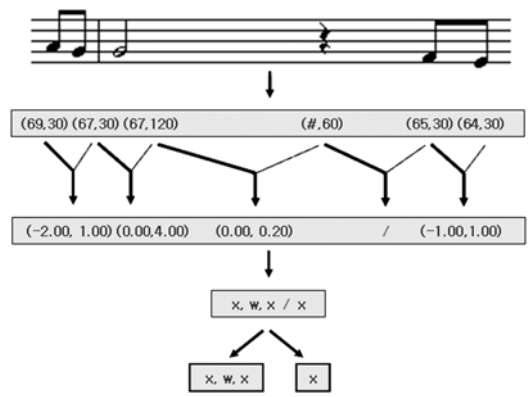

그림 11 멜로디를 한 박자 이상의 쉼표로 나누는 과정

3.4 인덱스

본 논문은 그림 12와 같이 서픽스 트리 구조를 갖는 3개의 인덱스를 생성한다. 여기서 서픽스 트리를 사용하는 이유는 사용자가 트리에 저장된 멜로디안의 임의의 부분부터 질의하여도 검색이 가능하도록 하기 위해서이다. 우선 모든 음악을 문자 형태로 변환한 후 음표단위로 서픽스 트리에 저장하는데 이를 1차 인덱스라 부른다. 1차 인덱스를 생성하는 이유는 사용자가 빈번 멜로디 부분을 질의하지 않거나 긴 쉼표 후에 시작되는 멜로디를 질의하지 않는다면 빈번 멜로디 인덱스 또는 쉼표 단위 인덱스에서 검색이 수행되지 못하고 데이타베이스로 접근해야 하는데 이를 최소화하기 위해서 이다.

다음으로 3.3절에서 추출한 빈번 멜로디를 음표 단위로 서픽스 트리에 저장하는데 이를 2차 인덱스 또는 빈번 멜로디 인덱스라고 한다. 이는 음표 단위로 트리에 저장함으로써 빈번 멜로디의 부분 멜로디를 질의하여도 검색이 가능할 수 있도록 하였다.

마지막으로 세 번째 인덱스인 쉼표 단위 인덱스는 어느 일정한 길이 이상의 쉼표를 경계로 하여 멜로디를 자른 후 쉼표 단위로 서픽스 트리에 저장한 것을 의미하고 이를 3차 인덱스라 한다. 그림 12에서 세 번째 문자열을 보면 일정 길이 이상의 긴 쉼표인 '𝄽'를 경계로 멜로디 들이 나뉘어져 있다. 모든 인덱스의 단말 노드에는 음악 아이디와 마디 위치, 그리고 음표 위치를 저장해 둔다.

이 서픽스 트리의 구성에 필요한 비용은 각 인덱스마다 저장할 멜로디의 길이 즉, 시퀀스의 길이가 m이고 알파벳의 개수가 Σ일 경우 트리를 생성하는데 O(mlog|Σ|)이 소요된다. 또한 저장 공간은 O(m)이 필요하다. 또한 인덱스에 저장할 멜로디의 전체 길이가 m이고 n길이의 멜로디를 삽입, 삭제할 때 유지비용은 각각 O(nlog(n+m)), O(nlogm)이 소요된다.

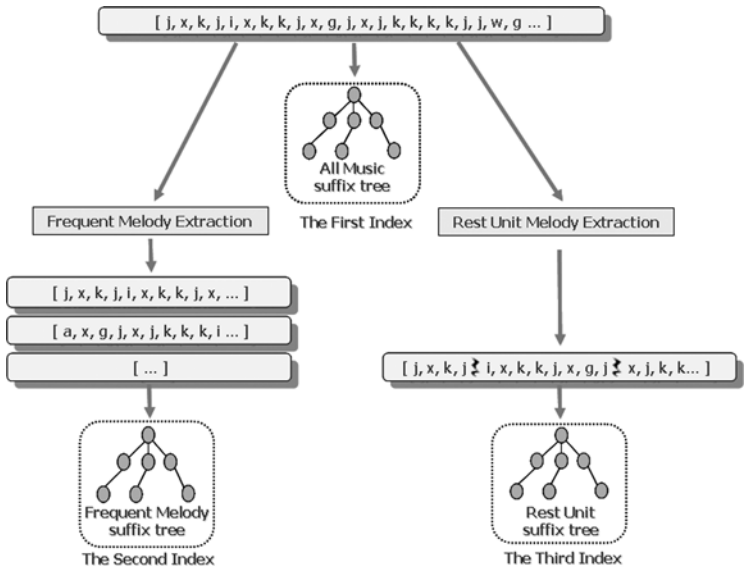

그림 12 인덱스 생성

4. 질의 처리

이 절에서는 사용자의 허밍 질의를 처리하는 과정에 대해 설명한다. 질의 처리는 크게 두 단계로 나뉘어진다. 첫 번째 단계는 질의 전처리 단계로서 입력된 질의를 데이타베이스와 인덱스에 저장된 데이타 형태와 같도록 변경해준다. 다음으로 두 번째 단계는 검색 단계로서 원하는 음악이 검색될 때까지 미리 생성한 인덱스를 차례로 검색해 나간다. 그림 13은 질의 처리 과정을 보여준다.

우선 데이타베이스 검색의 첫 번째 단계인 질의 전처리 과정에 대해서 설명한다. 입력 데이타인 질의는 사용자가 직접 마이크를 통해 흥얼거린 허밍이다. 이는 데이타베이스를 생성할 때 음악 데이타의 입력 형태인 미디와 다른 웨이브(Wave)형태를 지니고 있다. 그러므로 웨이브 형태의 음악에서 미디 형태와 같이 음의 높이와 길이 그리고 쉼표의 길이를 추출해야 한다. 추출한 이 정보를 통해 데이타베이스에 저장된 형태와 마찬가지로 특징 데이타를 계산한다. 즉, 높이의 변화량과 길이의 변화율을 계산하는 것이다. 다음으로 인덱스를 검색하기

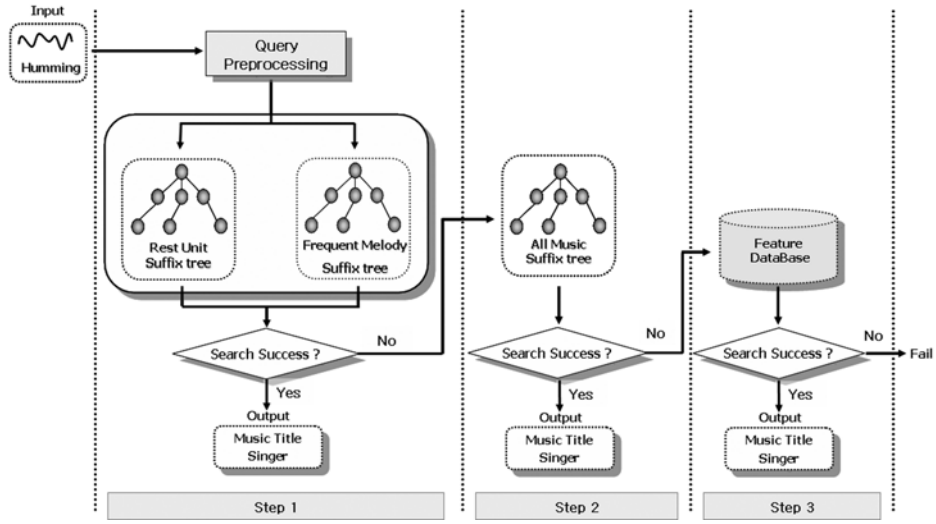

그림 13 질의 처리 과정

표 3 단계별 검색 방법

단계	검색 방법
1단계 검색	2차 인덱스와 3차 인덱스 병렬 검색
2단계 검색	1차 인덱스 검색
3단계 검색	특징 데이타베이스 검색

위해 특징 데이타를 문자 형태로 변환한다. 이는 인덱스에 입력된 문자열들과 같은 문자 변환 기준을 사용하여야 한다. 이러한 질의 전처리 과정이 끝나면 다음으로 검색을 진행한다.

검색 단계는 표 3과 같이 3단계로 나누어진다. 우선 질의 전처리의 마지막 단계에서 문자열 형태로 만든 질의를 입력하여 빈번 멜로디 인덱스인 2차 인덱스와 쉼표 단위 멜로디인 3차 인덱스를 병행하여 검색하는 1단계 검색이 있다. 이는 사용자가 빈번 멜로디를 흥얼거렸거나 일정 길이 이상의 쉼표 다음에 시작하는 멜로디를 흥얼거렸다면 이 단계에서 검색이 완료되어 사용자에게 가장 빠른 시간 내에 결과를 전달할 수 있다. 이 때 비교 방법으로 문자열의 정확 매칭(String Exact Matching)알고리즘을 사용한다. 이 단계에서 사용자가 원하는 검색 결과를 얻었다면 검색을 종료시킨다. 그러나 1단계에서 검색이 완료되지 못하고 검색 결과를 얻지 못할 경우 2단계 검색으로 들어간다. 2단계 검색의 목적은 모든 음악을 저장한 인덱스를 사용함으로써 사용자가 빈번 멜로디 또는 쉼표 다음에 시작하는 멜로디를 질의하지 않았을 경우에 검색이 완료될 수 있도록 하려는 것이다. 이러한 검색 2단계에서 사용자가 원하는 음악을 검색하였다면 사용자는 원하는 음악을 전달받고 검색을 종료시키면 된다. 하지만 2단계 검색에서도 검색이 완료되지 못할 경우 특징 데이타베이스를 접근하는 3단계 검색을 통해 검색을 수행한다. 이 때 비교 방법으로 유클리디안 거리 함수를 사용하여 가까운 거리의 음악을 얻어낸다. 본 논문이 제안하는 시스템은 2단계 검색과 3단계 검색을 최소화하고 1단계 검색에서 정확한 검색 결과를 얻어낼 수 있기 때문에 검색 속도가 상대적으로 우수하다.

5. 성능 평가

본 장에서는 제안하는 허밍 질의 처리 시스템의 효율성과 정확도 등의 실험 결과를 기술한다. 5.1절에서는 데이타 및 질의 생성 방법 등을 설명하고 5.2절에서는 실제 사용자의 질의를 입력받아 사용자들이 한 음악에서 어떤 부분을 자주 질의하는지 분석하였다. 5.3절에서는 빈번 멜로디를 생성하기 위해 필요한 두 가지 변수인 *BST*와 *FMLT* 중 *BST*값을 변화시켜 가며 검색 시간과 정확도를 분석하여 가장 최적의 *BST*값을 결정하였고 5.4절에서는 *FMLT*값을 변화시켜가며 성능을 분석하여 최적의 *FMLT*값을 결정하였다. 또한 5.5절에는 문자 변환 기준인 2차원 공간의 범위를 변화시켜 가며 정확도 분석을 통해 가장 적절한 기준을 결정하였다. 5.6절에서는 데이타베이스의 크기와 질의의 길이에 따른 검색 속도를 검증하였고 5.7절에는 질의의 부정확한 정도에 따른 검색 결과의 정확도를 검증하였다.

5.1 데이타 및 질의 생성 방법

본 논문은 실험에 사용할 데이타로서 데이타베이스와 인덱스에 저장하기 위한 미디 형식의 음악과 검색을 위한 웨이브 형식의 허밍 질의가 필요하다. 우선 데이타베이스와 인덱스에 저장할 음악 데이타로서 한국 가요 3000곡을 수집한 후 이들의 특징 데이타를 추출하여 특징 데이타베이스를 구축하였고 이들을 문자 형태로 변환하여 3개의 인덱스를 생성하였다. 또한 성능 평가를 위해 사용할 질의는 표 4, 5와 같이 생성하였다. 우선 표 4는 사용자 질의와 실제 멜로디 사이의 거리를 증가시켜 가며 각 30개씩 생성한 것이다. 예를 들어 실제 멜로디와의 거리가 2인 질의란 사용자가 입력한 허밍 질의와 실제 멜로디와의 특징 데이타를 사용하여 유클리디안 거리를 계산한 결과 값이 2인 경우를 의미한다. 표 5는 질의 시간을 5초 단위로 증가시켜 가며 길이가 다른 여러 종류의 질의들을 생성한 것이다. 그리고 각 질의 시간당 30번씩 질의를 함으로써 질의 시간동안 발생한 음표수를 평균화 하였다.

표 4 실제 멜로디와의 거리 정도에 따른 질의

Type of Query (The distance of between query and target melody)	count
2	30
4	30
6	30
8	30
10	30
12	30
14	30
16	30

표 5 허밍 질의 시간에 대응되는 평균 음표 수

Query time (sec.)	Average value of the number of notes	Count
5	7	30
10	13	30
15	20	30
20	26	30
25	30	30
30	33	30

5.2 사용자 질의 패턴 분석

본 절에서는 실제 사용자에게 직접 허밍을 부르게 하여 사용자가 한 음악의 어떤 부분을 자주 질의하는지 분석해 보았다. 본 실험에서 사용한 음악 수는 현재 데이타베이스에 저장된 3000곡이고 참여한 사용자 수는 30명이며 총 질의 수는 중복을 허락하여 약 700곡이다. 본 실험은 각 음악마다 음악의 시작부분, 빈번 멜로디 부분, 한 박자 이상의 쉼표 후에 시작되는 멜로디 부분 그리고 그 외의 부분으로 나누어 그림 14와 같이 해당하는 부분의 질의 횟수를 그래프로 표현해 보았다.

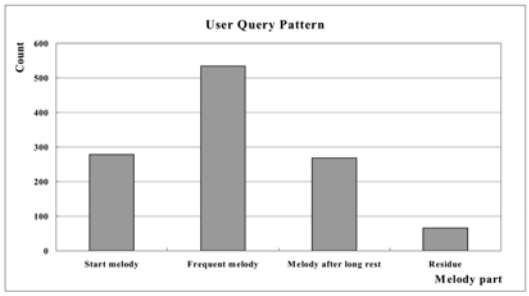

그림 14 사용자 질의 패턴

그림 14에서 사용자는 전체 질의 중 빈번 멜로디 부분을 약 76%정도 질의하였음을 알 수 있다. 이는 사용자가 빈번 멜로디를 가장 자주 질의한다는 가정을 뒷받침해주는 증거가 된다. 그리고 시작 멜로디와 긴 쉼표 이후에 나타나는 멜로디 또한 많은 수가 질의되었음을 알 수 있다. 음악의 시작 멜로디를 질의한 경우는 전체 음악의 약 40%를 차지하고 시작 멜로디를 질의한 경우의 35%가 빈번 멜로디와 일치한다. 또한 일정 길이 이상의 쉼표 후에 시작되는 멜로디를 질의한 경우는 전체 음악의 약 38%를 차지하고 이들 질의의 약 49%가 빈번 멜로디와 일치한다. 반면 그 외 나머지 부분의 멜로디는 전체 질의 중 약 6%만이 질의되었다.

5.3 최적의 *BST* 값 결정

빈번 멜로디를 생성하기 위해서 필요한 2가지 변수인 *BST*와 *FMLT*는 빈번 멜로디의 길이를 결정하는데 큰 역할을 한다. 본 절에서는 두 개의 변수 중 마디 선택 임계값인 *BST*의 값을 변화시켜가며 빈번 멜로디의 크기와 검색 시간 그리고 정확도를 분석하여 가장 최적의 성능을 보여주는 값을 결정하였다. 그림 15는 *BST*값을 1부터 4까지 1씩 증가해가며 빈번 멜로디 인덱스에 저장된 문자수를 비교한 것이다. 이때 특징 데이타베이스에 저장된 음악의 수는 3000곡으로 지정하였고 질의는 15초의 길이를 갖는 빈번 멜로디 부분으로 표 4과 같이 실제 멜로디와의 거리 당 30개씩 생성하였다. 또한 *FMLT*값은 15초인 질의의 길이보다 긴 빈번 멜로디를 만들어 낼 수 있는 값으로 검증된 6으로 지정하였다. 이에 대한 검증은 5.4절에서 보여줄 것이다.

빈번 멜로디의 크기를 분석해본 결과 *BST*값이 1인 경우 다른 값들에 비해 빈번 멜로디 인덱스 크기가 가장 크다는 것을 확인할 수 있었다. 즉, 상대적으로 길이가 긴 빈번 멜로디가 생성된다는 의미이다. 이는 한 음악에서 1번 이상 발생한 마디들로 이루어진 빈번 멜로디가 생성되었기 때문이다. 반면 *BST*값이 커질수록 빈번 멜로디의 인덱스 크기가 감소함을 확인할 수 있었다. 이는 대부분의 음악에서 짧은 길이의 빈번 멜로디가 생성될 뿐만 아니라 빈번 멜로디가 생성되지 못하는 음악들도 발생하기 때문이다. 예를 들어 *BST*값이 3인 경우 입력된 3000곡의 음악 중에 빈번 멜로디를 생성하지 못하는 음악이 332곡이고, 4인 경우는 1369곡임을 확인할 수 있었다. 이는 *C*의 값이 3이상인 마디를 포함하지 않는 음악이 332곡이고 4이상인 마디를 포함하지 않는 음악이 1369곡이라는 의미이다. 또한 3 또는 4이상의 마디를 가지고 있는 음악이라도 그 마디의 개수가 작기 때문에 길이가 짧은 빈번 멜로디가 생성되는 것이다.

그림 16은 *BST*의 값에 따라 빈번 멜로디 인덱스의 검색 시간을 비교해 본 것이다. 이때 입력된 음악수를

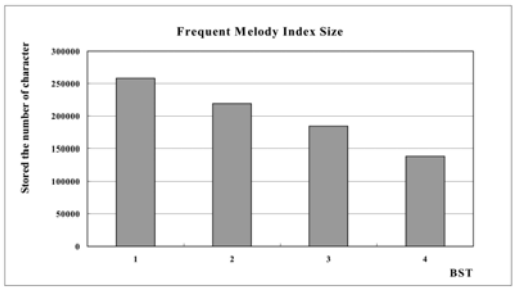

그림 15 *BST*의 값에 따른 빈번 멜로디 인덱스 크기

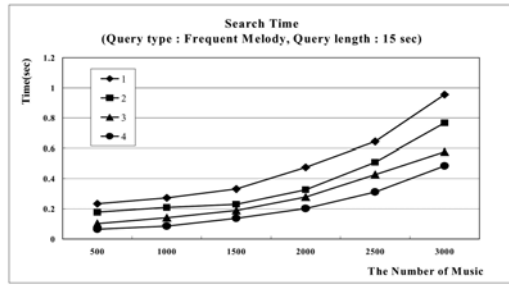

그림 16 *BST*의 값에 따른 빈번 멜로디 인덱스의 검색 시간

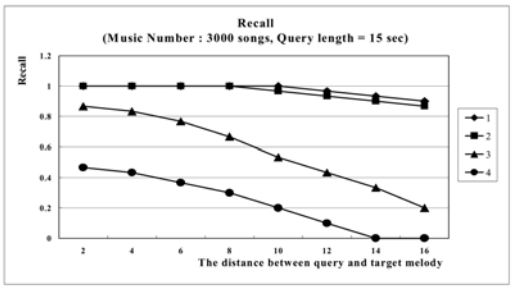

그림 17 *BST*값에 따른 재현율 비교

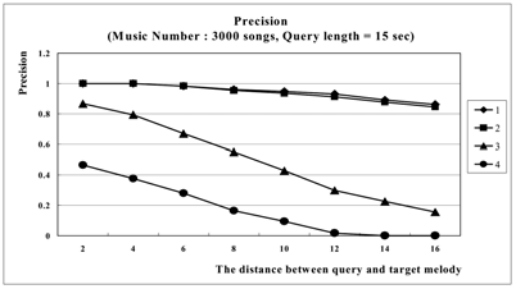

그림 18 *BST*값에 따른 정밀도 비교

500곡에서 3000곡까지 500곡 단위로 늘려가며 실험하였다. 그 결과 입력된 음악 수에 상관없이 *BST*의 값이 작을수록 검색 시간이 느려짐을 확인할 수 있다. 이는 *BST*값이 작을수록 빈번 멜로디 인덱스의 크기가 증가하기 때문이다.

다음으로 *BST*값에 따른 정확도를 측정하여 성능을 분석해 본 결과 그림 17, 18과 같은 결과를 얻었다. 두 그래프를 보면 재현율과 정밀도는 *BST*의 값이 작을수록 높게 측정됨을 확인할 수 있다. 그리고 *BST*값이 1과 2인 경우의 재현율과 정밀도는 비슷한 수치를 보였지만 3과 4인 경우는 그 수치가 상당히 떨어짐을 보여준다. 이러한 결과에 대한 이유는 2가지로 나누어 볼 수 있다. 첫 번째 이유는 *BST*가 3과 4일 때에 생성된 빈번 멜로디의 길이가 질의의 길이보다 짧은 경우가 많이 발생하기 때문이다. 본 실험은 질의의 길이를 15초로 고정시켰는데 이에 해당하는 평균 문자수는 표 5과 같이 20개이고 *BST*의 값에 따라 생성되는 하나의 빈번 멜로디 평균 문자수는 표 6과 같다. 즉, *BST*값이 3과 4일 때 생성되는 대부분의 빈번 멜로디의 길이가 질의의 길이보다 짧기 때문에 검색 결과를 얻지 못하는 경우가 발생한다. 두 번째 이유는 *BST*값이 커질수록 빈번하게 반복된 마디들을 포함하지 못하여 부정확한 빈번 멜로디가 생성되기 때문이다. 예를 들어 *BST*값이 4인 경우 빈번 마디값이 4이상인 마디들로만 이루어진 빈번 멜로디가 생성되는데 이는 사용자가 한 음악에서 3번 반복된 부분을 질의하였다면 검색결과를 얻지 못하게 된다.

위의 실험들을 통해 최적의 성능을 보이는 *BST*값은 2임을 확인할 수 있었다. 그 이유는 *BST*값이 2인 경우가 3과 4인 경우보다 빈번 멜로디 인덱스의 크기가 크고 검색 시간이 느리지만 정확도 측면에서 향상된 성능을 보였고 *BST*값이 1인 경우보다 검색 시간이 빠르고 인덱스 크기가 작기 때문이다.

표 6 *BST*값에 따른 하나의 빈번 멜로디의 평균 길이

BST	빈번 멜로디의 평균 길이 (문자 수)
1	25.5
2	22.6
3	14.7
4	8.7

5.3 최적의 *FMLT* 값 결정

본 절에서는 빈번 멜로디 길이의 임계값인 *FMLT*를 변경해가며 빈번 멜로디 인덱스 크기와 검색 시간, 그리고 정확도를 분석하여 최적의 *FMLT*값을 결정한다. *FMLT*는 빈번 멜로디의 길이를 결정하기 위해 마디의 순위(*R*)를 사용하여 계산된 BSum의 임계값을 의미한다. 그러므로 *FMLT*값에 따라 빈번 멜로디의 길이와 빈번 멜로디 인덱스의 크기가 달라진다. 본 실험은 5.3절과 마찬가지로 3000곡의 음악을 사용하였고 질의는 15초의 길이를 갖는 빈번 멜로디 부분을 입력하였다. 그리고 빈번 멜로디 인덱스를 생성하기위한 또 다른 변수 *BST*는 2값을 사용하였다. 그림 19는 *FMLT*를 2부터 10까지 2단위로 증가시켜가며 빈번 멜로디 인덱스를 생성한 후 그 크기를 분석한 그래프이다. 또한 그림 20은 *FMLT*값에 따른 검색 시간을 비교한 그래프이다. 이는 *FMLT*값이 증가할수록 빈번 멜로디의 길이가 길어지기 때문에 인덱스 크기가 점점 커지고 이로 인하여 검색 시간이 증가하는 것이다.

다음으로 *FMLT*값 마다 질의의 부정확한 정도에 따른 정확도를 분석하였다. 그림 21과 22는 각각 재현율과 정밀도를 분석한 그래프이다. 두 개의 그래프는 *FMLT*값이 증가할수록 정확도가 높게 나타나고 이에 따른 각

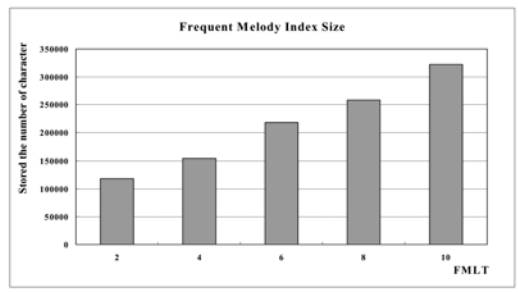

그림 19 *FMLT*값에 따른 빈번 멜로디 인덱스 크기

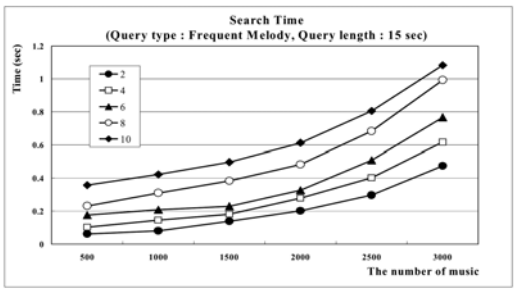

그림 20 *FMLT*의 값에 따른 빈번 멜로디 인덱스의 검색 시간

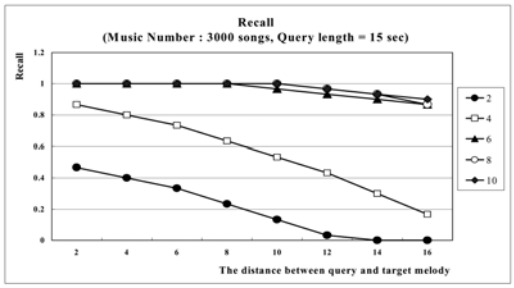

그림 21 *FMLT*값에 따른 재현율 비교

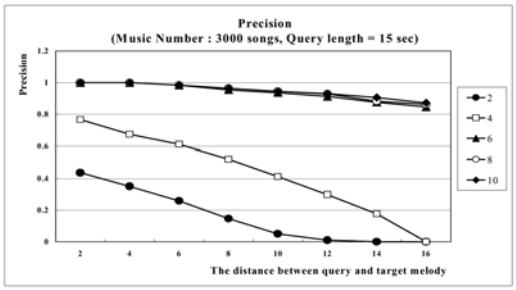

그림 22 *FMLT*값에 따른 정밀도 비교

정확도는 질의의 부정확한 정도가 증가할수록 그 수치가 감소함을 보여준다. 이는 *FMLT*값이 작을수록 빈번 멜로디의 길이가 짧아지므로 15초의 질의를 입력하였을 경우 검색되지 못하는 경우가 증가하기 때문이다. 표 7은 각 *FMLT*값에 따라 생성되는 빈번 멜로디의 평균 문자수를 보여준다. *FMLT*값이 2와 4일 경우에 생성되는 빈번 멜로디의 평균 길이는 질의의 길이보다 짧음을 확인하였다. 그리고 *FMLT*값이 6, 8, 10인 경우는 비슷한 정확도를 보였다. 이는 질의를 포함한 신뢰성 높은 빈번 멜로디가 생성되었기 때문에 만족스런 검색결과를 얻게 된 것이다.

위의 실험을 통해 가장 최적의 성능을 보여주는 *FMLT*의 값은 6임을 확인할 수 있었다. 이는 높은 정확도를

표 7 *FMLT*값에 따른 하나의 빈번 멜로디의 평균 길이

FMLT	빈번 멜로디의 평균 길이 (문자 수)
2	8.4
4	14.2
6	23.6
8	30.1
10	37.3

보여준 값인 6, 8, 10중에 가장 검색 속도가 빠르고 인덱스의 크기가 작기 때문이다. 그리고 고정된 질의의 길이보다 긴 빈번 멜로디를 생성하여야 좋은 성능을 보여줌을 확인할 수 있었다. 그러므로 이는 질의의 길이에 따라 적절한 *FMLT*값을 사용하여야 함을 알려준다. 또한 적절한 *BST*값과 *FMLT*값을 사용하여야 신뢰성 있는 빈번 멜로디가 생성된다는 것을 증명해 준다.

5.5 정확도 분석을 통한 적절한 문자 변환 기준 결정

본 절에서는 2차원 공간의 범위를 변경해가며 정확도를 분석해 봄으로써 가장 정확도가 높은 최적의 기준을 결정한다. 본 실험에서는 그림 23과 같이 4가지 형태의 문자 변환 기준을 각각 적용하여 정확도를 분석해 보았다. 이때 사용되어진 질의는 표 4와 같이 질의의 부정확한 정도를 변화시켜가며 생성한 멜로디들이다. 본 실험은 원하는 음악 한곡을 찾으면 검색을 종료한다. 실험에 앞서 파라메타인 *FMLT*값은 6으로, *BST*값은 2로 지정하여 허밍 질의 처리 시스템을 구축해 둔다. 또한 3000곡이 저장되어 있는 특징 데이타베이스를 사용하였고 허밍 질의의 길이를 15초로 고정시켜 놓았다. 그림 24와 25는 실제 멜로디와의 거리 당 30개의 질의를 입력하여 얻어낸 정확도이다.

그림 24는 4가지의 문자변환 기준을 적용한 결과의 재현율을 비교 분석한 그래프이다. 그림과 같이 재현율은 모든 범위에서 질의가 부정확 할수록 그 값이 감소되어짐을 알 수 있다. 그러나 네 개의 범위 중 Range 4가 재현율이 가장 높게 나타남을 확인할 수 있는데 이는 하나의 문자열로 할당되는 구간의 크기가 다른 범위보다 크기 때문에 부정확한 질의라도 실제 멜로디와 같은 문자열로 변환됨으로써 원하는 음악이 검색되어지는 경우가 많이 발생하기 때문이다. 반면 Range 1은 가장 재현율이 낮게 나타났는데 이는 하나의 문자열로 할당되는 구간의 크기가 다른 범위에 비해 상대적으로 작기 때문에 사용자가 허밍을 부정확하게 부를수록 사용자가 찾고자 하는 음악과 다른 문자열로 변환되는 경우가 많이 발생하므로 원하는 음악이 검색되어지는 경우가 줄어들게 된다. 다음으로 그림 25는 질의의 부정확한 정도를 증가 시키며 4가지 종류의 문자 변환 기준을 적용한 결과의 정밀도를 측정하였다. 이때 질의가 부정확 할수록 모든 문자 변환 기준에서 정밀도가 감소되어지는 것

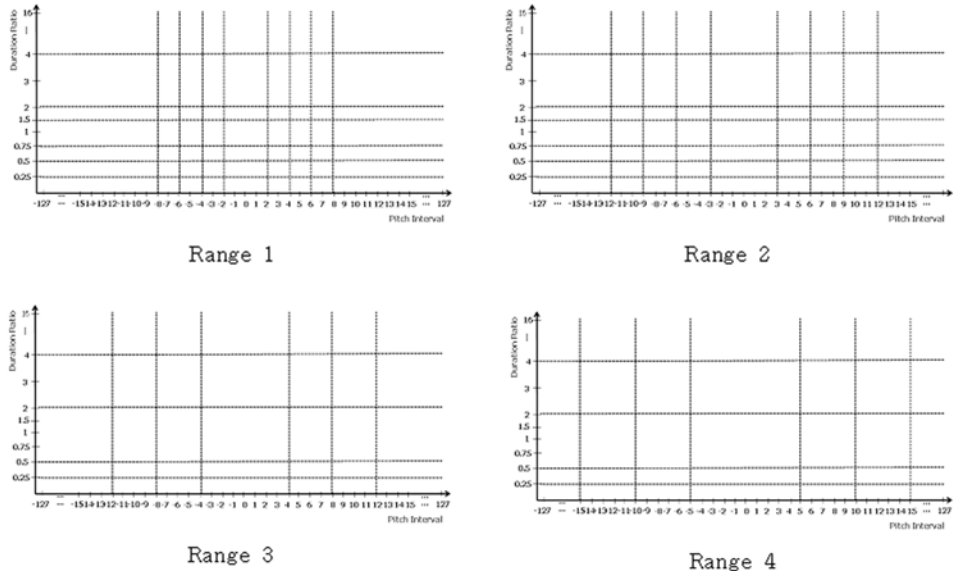

그림 23 문자 변환 기준 범위 종류

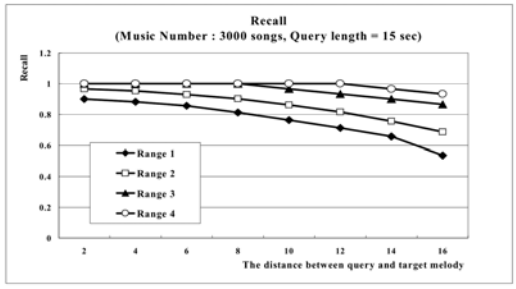

그림 24 문자 변환 기준에 따른 재현율

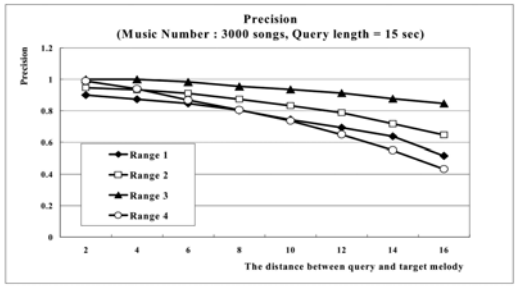

그림 25 문자 변환 기준에 따른 정밀도

을 확인할 수 있었다. 이는 질의의 부정확한 정도가 커질수록 원하지 않은 음악들의 검색 횟수가 증가하기 때문이다. 특히 문자 변환 기준의 2차원 공간상의 범위가 가장 큰 Range 4의 경사가 가장 급함을 알 수 있다. 이는 다른 범위보다 질의가 부정확 할수록 원하지 않는 음악들이 더 많이 검색됨을 알려준다. 위의 실험을 통해 정밀도와 재현율 모두가 상대적으로 가장 최적의 값을 나타낸 경우가 Range 3임을 발견할 수 있었다.

5.6 검색 시간 비교

본 절에서는 특징 데이타베이스에 저장된 음악 수를 증가시키면서 검색 시간을 비교한다. 또한 질의의 길이를 증가시키며 같은 방법으로 검색 시간을 비교한다. 비교 대상과 그에 해당하는 표현법은 표 8과 같이 정하였다.

우선 첫 번째 검색 방법은 오직 특징 데이타베이스만을 순차적으로 검색하는 검색 방법이다. 이는 유클리디안 거리 함수를 사용하여 음악의 처음부터 원하는 음악이 검색 될 때까지 거리 계산을 수행한다. 두 번째 검색 방법으로 모든 음악을 문자 형태로 변환하여 저장한 1차 인덱스를 추가한 검색 방법이다. 이 검색 방법은 1차 인덱스를 검색한 후 검색이 성공하지 못하면 특징 데이타베이스를 접근하는 방식이다. 이는 사용자가 빈번 멜로디 또는 긴 쉼표 후에 나오는 멜로디 질의에 전혀 상관없이 검색을 수행한다. 세 번째 검색 방법으로는 본 논문이 제안하는 방법으로 빈번 멜로디 인덱스인 2차 인덱스와 쉼표 단위 인덱스인 3차 인덱스를 추가한 검색 방법이다. 마지막으로 비교할 검색 방법으로는 N-gram방법이다.

표 8 비교 대상 검색 방법 종류와 표현법

Compared search method	Expression
1. 특징 데이타베이스 검색	Naïve DB scan
2. 1차 인덱스 검색 추가	2 step search method
3. 2차 인덱스와 3차 인덱스 병렬 검색 추가	Our search method (3 step search method)
4. N-gram	N-gram

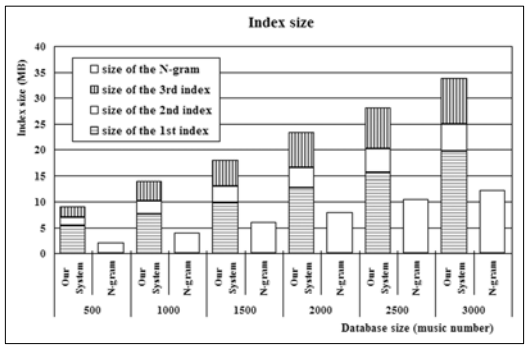

그림 26 데이타베이스 크기와 비교 시스템 별 인덱스 크기

그림 26은 특징 데이타베이스에 저장한 음악의 수에 따라 생성되는 인덱스의 크기를 비교한 그래프이다. 이는 본 논문이 제안하는 시스템의 각 인덱스 크기를 하나의 막대로 표현하였고 N-gram의 인덱스 크기를 다른 막대로 표현하였다. 본 논문이 생성한 3개의 인덱스의 전체 크기는 N-gram의 크기보다 약 2.5배 크지만 검색 속도의 향상을 위하여 인덱스의 크기 증가에 대한 비용을 감수하였다.

다음으로 그림 26은 특징 데이타베이스에 저장된 음악 수에 따른 검색 시간을 비교한 그래프이다. 데이타베이스 크기는 500곡을 시작으로 3000곡 까지 500곡 단위로 증가시켰다. 이때 사용한 질의의 개수는 30개이고 각 질의의 길이는 15초로 고정시켜 놓았다. 또한 각 질의의 부정확한 정도는 4로 지정하였다. 분자 변환 기준은 Range 3을 사용하였고 기본 면수인 FMLT와 BST는 각각 6과 2로 지정하여 검색을 수행하였다. 그림 28은 질의의 길이를 증가시켜 가며 검색 시간을 비교한 그래프이다. 5초 동안 흥얼거렸을 경우를 시작으로 30초까지 5초 단위로 증가시켰다. 이때 사용한 데이타베이스 크기는 3000곡으로 고정시켜 놓았고 문자 변환 기준과 변수값은 위와 동일하다. 그림 27에서 데이타베이스 길이가 증가할수록 본 논문이 제안하는 검색 방법의 검색 속도가 다른 검색 방법보다 상대적으로 향상됨을 알 수 있다. 이는 1차 검색 단계에서 검색이 완료되는 경우가 많이 발생하기 때문이다. 반면 N-gram 검색 방법이 본 논문의 검색 방법보다 약 2배가량 느림을 알 수 있었다. 그 이유는 N-gram의 인덱스 크기가 본 논문이 제안하는 2차 인덱스와 3차 인덱스 크기보다 크기 때문이고 1차 인덱스 보다는 크기가 작지만 서픽스 트리를 사용하는 본 논문의 검색 방법과 달리 순차적으로 문자열을 비교해가면 일치되는 문자열을 찾아야 하는 검색 방법을 사용하기 때문이다. 마지막으로 데이타베이스를 순차적으로 검색한 방법은 검색 속도가 가장 느림을 보여준다. 이는 데이타베이스에 저장된 모든 음악의 음표들의 특징 데이타를 통해 유클리디안 거리 함수를 사용하여 검색하는 방법으로 많은 계산 량을 필요로 하기 때문이다. 만약 하나의 음악에 m개의 음표가 있다면 한 음표는 높이 변화량과 길이 변화율이라는 두 개의 정보를 가지고 있으므로 $2m$개의 숫자 형태의 특징 데이타가 생성된다. 그리고 질의의 음표 개수가 n개라면 이도 마찬가지로 $2n$개의 숫자 형태의 특징 데이타가 생성된다. 이들은 서로의 거리를 계산하기 위해 음표단위로 질의를 이동시켜 가면 $(m-n+1)*2n$개의 계산 량이 필요하게 된다. 다음으로 그림 28은 질의의 길이를 증가시켜가며 검색 시간을 측정해 보았다. 이때도 위의 실험과 비슷한 형태의 검색 결과를 얻을 수 있었다.

5.7 정확도 비교

본 절에서는 사용자 질의의 부정확한 정도에 따른 정확도를 4개의 검색 방법에 적용하여 비교해 보았다. 그림 29는 재현율을 그림 30은 정밀도를 그래프로 표현하였다. 이때 데이타베이스에는 3000곡을 저장하였고 질의의 길이는 15초를 기준으로 하였다. 또한 위의 실험들과 마찬가지로 문자 변환 기준은 Range 3을 사용하였고 기본 파라메타인 $FMLT$와 BST는 각각 6과 2로 지정하여 검색을 수행하였다.

본 실험은 각 질의의 부정확한 정도마다 데이타베이스에서 원하는 음악이 모두 검색되도록 하였다. 방법은 질

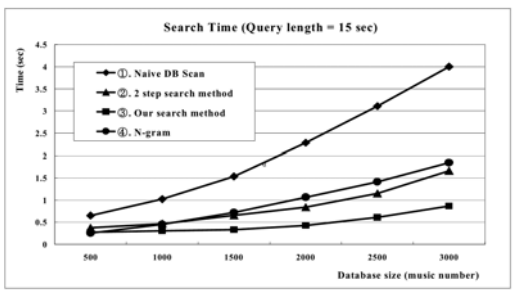

그림 27 데이타베이스 크기에 따른 검색 시간

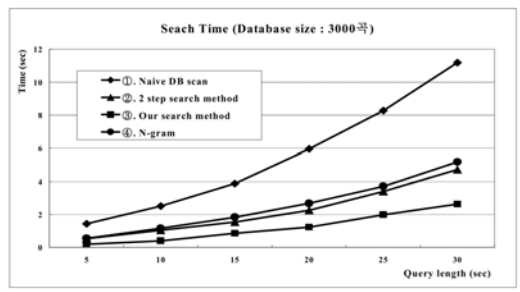

그림 28 질의 길이에 따른 검색시간

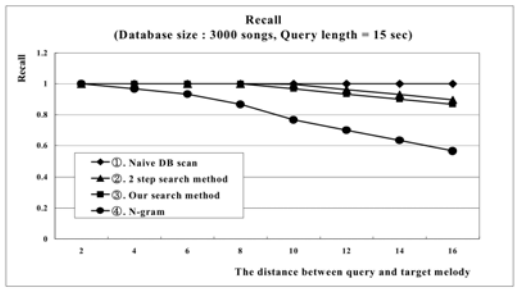

그림 29 질의의 부정확한 정도에 따른 재현율

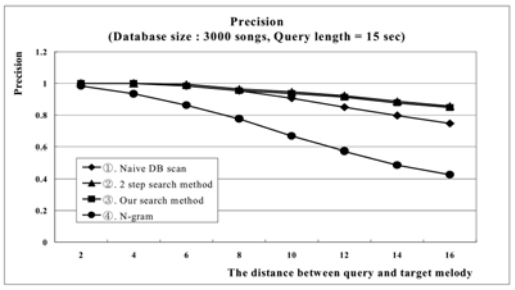

그림 30 질의의 부정확한 정도에 따른 정밀도

의의 부정확한 정도가 2일 때 실제 멜로디와의 거리가 2인 30개의 서로 다른 종류의 질의를 입력하여 검색하였다. 이와 같은 방법을 사용한 이유는 만약 질의와의 거리 차이가 2인 모든 음악을 검색하도록 하였을 때 원하는 음악이 검색되도록 하였다면 이때 인덱스를 사용한 검색 방법인 두 번째 검색방법과 세 번째 검색 방법 그리고 타 검색 방법인 N-gram을 이용한 검색방법의 정확도는 상대적으로 어떠한 결과가 나오는지를 비교한 것이다. 그러므로 질의의 부정확한 정도가 늘려가면서 매번 위와 같은 실험을 통해 서로의 상대적인 정확도를 비교하였다. 두 개의 그래프에서 중점적으로 보아야 할 점은 본 논문이 제안하는 방법이 기존 N-gram 방법보다 질의가 부정확 할수록 원하는 음악이 검색되어지는 경우가 많은 반면 원하지 않는 음악이 보다 적게 된다는 점이다.

5.8 장르별 성능 분석

본 절에서는 음악 데이타를 장르별로 나뉘어 검색 시간과 정확도를 분석하였다. 본 실험에서 사용한 장르는 3가지로 재즈, 락, 클래식이며 이들을 사용한 이유는 각 리듬의 특징이 다른 어떤 장르보다 잘 구분되기 때문이다.

본 실험을 위해 각 장르별로 최적의 성능을 보여주는 파라미터 값을 정하였다. 이때 데이타베이스에 저장할 음악 수는 장르별로 각각 225곡을 사용하였고, 질의는 표 9와 같이 생성하였고 모든 질의의 길이는 5초로 고정하였다. 각 장르별로 데이타베이스의 크기가 증가함에 따른 검색 시간과 질의의 부정확한 정도가 변함에 따른 정확도를 분석한 결과 표 10과 같이 3개의 최적의 파라미터 값이 결정되었다. 재즈와 락은 장르를 구분하지 않은 음악 셋을 입력한 실험과 같은 파라미터 값이 요구되지만 클래식은 BST값이 3, 문자 변환 기준이 Range 2로써 약간의 차이를 보였다. 우선 클래식의 BST값이 다른 장르보다 높게 나타난 이유는 클래식 마디의 반복 횟수가 평균적으로 다른 장르인 재즈와 락에 비해 높기 때문이다. 이는 클래식이 다른 장르에 비해 음악의 길이가 길기 때문인 이유도 있다. 다음으로 클래식의 문자변환 기준이 Range 2로 나타난 이유는 그림 31과 같이

표 9 실제 멜로디와의 거리 정도에 따른 질의

Type of Query (The distance of between query and target melody)	The number
2	15
4	15
6	15
8	15
10	15

표 10 장르별 최적의 파라미터 값

장르	Parameter		
	BST	FMLT	문자변환 범위
Jazz	2	6	Range 3
Rock	2	6	Range 3
Classical	3	6	Range 2

음의 높이 변화량의 절대 값과 길이변화율의 절대 값이 작은 경우가 다른 장르에 비해 많이 나타나기 때문이다, 그러므로 이들을 같은 문자로 변화되는 경우를 줄이기 위하여 Range 3보다 벡터 영역이 좀 더 세밀하게 나눠진 Range 2일 경우 좀 더 높은 정확도를 보이며 Range 1보다 허밍의 부정확한 정도가 커질수록 정확도가 덜 감소한다.

위와 같은 파라미터 값을 입력하여 장르별 검색시간과 정확도를 측정하였다. 그림 32는 데이타베이스 크기를 50곡씩 증가시켜가며 측정한 검색 시간을 보여준다. 이 그림을 통해 세 개의 장르 모두 비슷한 검색 속도를 보여줌을 알 수 있다. 다음으로 그림 33과 34는 정확도를 보여주고 있다. 이 또한 비슷한 성능을 보여줌을 그래프를 통해 확인할 수 있다.

이와 같이 세 개의 장르는 각 다른 특성의 리듬을 가지고 있지만 이에 맞는 적절한 파라미터 값을 시스템에 적용해주면 어떤 장르든 간에 비슷한 성능을 보여줌을 알 수 있다. 그러므로 검색 시 음악의 장르를 미리 파악하고 그에 맞는 적절한 파라미터를 적용하면 좀 더 향상된 성능을 가진 시스템이 구축될 것이다.

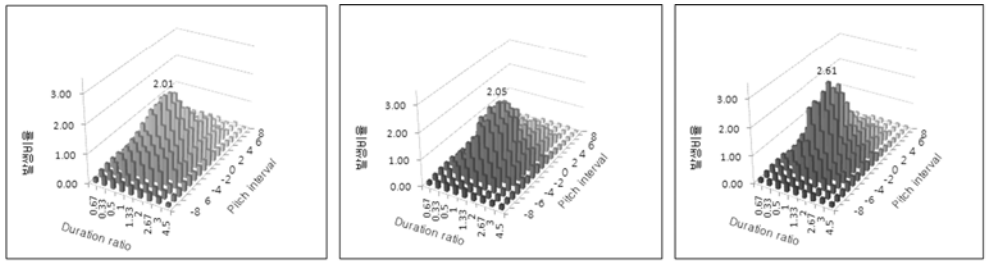

그림 31 장르별 특징 데이타 발생 비율 (a) 재즈, (b) 락, (c) 클래식

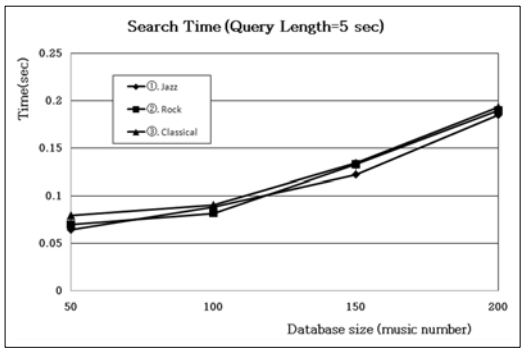

그림 32 장르별 데이타베이스 크기에 따른 검색 시간

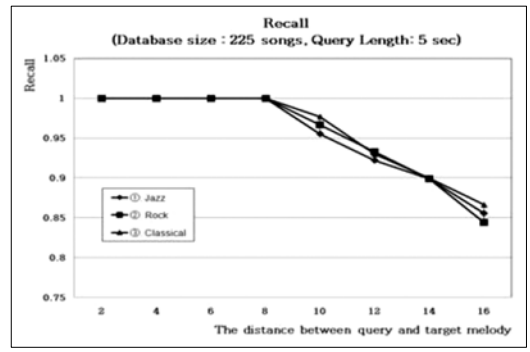

그림 33 질의의 부정확한 정도에 따른 장르별 재현율

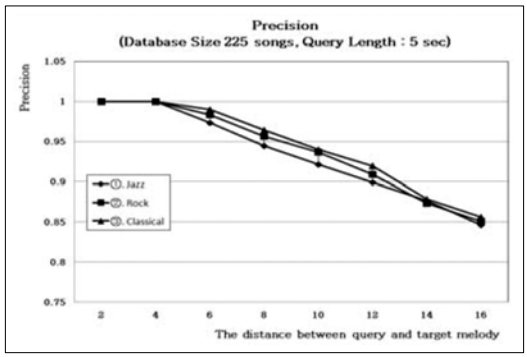

그림 34 질의의 부정확한 정도에 따른 장르별 정밀도

6. 결론 및 향후 연구

본 논문은 방대한 양의 음악 데이타베이스에서 사용자가 원하는 음악을 빠르게 검색하기 위하여 내용 기반 검색 방법을 기반으로 하였다. 즉, 사용자가 음악을 대표하는 키워드가 아닌 멜로디를 질의하여 유사한 멜로디를 갖는 음악을 데이타베이스에서 검색해 주는 검색 방법을 적용한 것이다. 이때 멜로디를 표현하는 특징 데이타로는 음표의 높이의 변화량과 길이의 변화율인 2차원 벡터로서 이들을 특징 데이타베이스에 저장하였고, 질의와의 유사성은 유클리디안 거리 함수를 사용하여 비교하였다. 그러나 이러한 방법은 특징 데이타베이스에 저장된 모든 음악을 순차적으로 비교해야하므로 많은 계산 비용과 검색 시간이 필요하다. 그러므로 이러한 비효율적인 검색방법을 보완하기 위해 효과적인 인덱스를 설계하였다.

본 논문은 사용자가 자주 질의할 가능성이 높은 멜로디를 의미 있는 멜로디로 취급하여 이들을 인덱싱하는 방법을 제안하였다. 이때 의미 있는 멜로디를 두 가지로 나누어 취급하였다. 첫 번째 의미 있는 멜로디란 빈번 멜로디로서 이는 한 음악에서 빈번하게 나타나는 멜로디를 사용자가 자주 질의한다는 점을 기반으로 선정하였다. 본 논문은 빈번 멜로디 추출 알고리즘을 제안하여 신뢰성이 높은 빈번 멜로디를 추출하였다. 두 번째 의미 있는 멜로디란 일정한 길이 이상의 쉼표 후에 시작되는 쉼표 단위 멜로디로써 이는 대체적으로 음악의 시작부분, 간주가 끝나고 다음절이 시작되는 부분, 그리고 후렴의 시작 부분 등에 나타난다. 이러한 멜로디는 사용자가 자주 질의한다는 가정 하에 접근하였고 이 가정을 실험을 통해 검증하였다. 이와 같은 의미 있는 멜로디를 인덱싱하여 이들을 먼저 검색함으로써 사용자가 원하는 음악을 신속히 알려주도록 하는 시스템을 구축하였다.

본 논문은 인덱스에 저장된 의미 있는 멜로디의 형태를 기존의 숫자 형태의 2차원 벡터가 아닌 문자열 형태

로 변환하여 검색 속도를 줄이고자 하였다. 또한 문자 형태로 변환하면서 멜로디의 정보 손실을 최소화하고 검색 결과의 정확도를 높이기 위해 2차원 벡터 공간을 하나의 문자로 치환하는 방법을 사용하였다. 이는 허밍이 가지고 있는 에러를 허용하는 방법으로서 사용자가 부정확한 음악을 입력하여도 원하는 음악이 검색되어질 수 있도록 하였다.

본 논문은 3단계 검색 방법을 사용하였다. 우선 1단계 검색은 빈번 멜로디 인덱스와 쉼표단위인덱스를 병행하여 검색을 수행하는 방법으로서 이때 검색 결과를 얻지 못했을 경우 2단계 검색으로 들어간다. 2단계 검색은 모든 음악을 음표 단위로 저장한 인덱스로서 마찬가지로 이때에도 검색 결과를 얻지 못했을 경우에 특징 데이타베이스를 순차적으로 검색하는 3단계 검색으로 들어간다. 이와 같은 3단계 검색 방법은 1차 검색 단계를 통해 원하는 음악을 찾는 것이 목표이다.

본 논문은 의미 있는 멜로디 인덱싱 방법과 에러를 허용하는 문자 변환 방법 그리고 3단계 검색 방법을 사용함으로써 기존의 방법보다 데이타베이스의 접근을 최소화하였을 뿐만 아니라 검색 시간이 줄었고 정확도가 향상되어 상당한 성능향상을 가져왔다.

본 논문의 공헌은 다음과 같다. 첫째, 사용자가 자주 질의하는 멜로디를 의미 있는 멜로디로 취급하였고 이들을 추출하여 인덱싱 함으로써 기존이 검색 방법보다 검색이 빠르게 수행되도록 하였다. 둘째, 사용자가 허밍이 많은 부정확한 멜로디라는 특징을 반영하여 부정확한 정도가 커져도 정확도가 보장될 수 있는 문자 변환 방법을 적용하였다. 마지막으로 데이타베이스의 접근을 최소화 하는 3단계 검색 방법을 사용하였다. 또한 실험을 통하여 이들을 검증하였다.

향후 연구로는 기존의 사용한 인덱스 구조의 문제점을 분석하고 이를 보완하여 검색 성능 향상을 가져오는 효과적인 인덱스를 생성하려 한다. 또한 3개의 인덱스를 생성함으로써 방대해진 인덱스의 크기를 줄일 수 있는 방안을 강구하려 한다.

참 고 문 헌

[1] A. Uitdenbogard, J. Zobel, "Melodic Matching techniques for Large Music Databases," In Proc. ACM Multimedia, pp. 57-66, 1999.

[2] P. Rolland, G. Raskinis, and J. Ganascia, "Musical Content-Based Retrieval: An Overview of the Melodiscov Approach and System," In Proc. ACM Multimedia, pp. 81-84, 1999.

[3] A. Ghias, J. Logan, and D. Chamberlin, "Query By Humming," In Proc. ACM Multimedia, pp. 231-236, 1995.

[4] S. Pauws, "CubyHum: A Fully Operational Query by Humming System," In Proc. 3rd International Symposium on Music Information Retrieval (ISMIR), pp. 187-196, 2002.

[5] S. Doraisamy, S. Ruger, "Robust polyphonic music retrieval with n-grams," Journal of Intelligent Information Systems, 21:1, pp. 53-70, 2002.

[6] R. L. Kline, E. P. Glinert, "Approximate matching algorithms for music information retrieval using vocal input," In Proc. ACM Multimedia, pp. 130-139, 2003.

[7] R. B. Dannenberg, W. P. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo, "The Musart Testbed or Query-By-Humming Evaluation," In Proc. 4th International Symposium on Music Information Retrieval (ISMIR), pp. 41-50, 2003.

[8] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro, and K. Kushima, "A Practical Query-By-Humming System for a Large Music Database," In Proc. ACM Multimedia, pp. 333-342, 2000.

[9] E. Scheirer, "Tempo and Beat Analysis of Acoustic Musical Signals," Acoustic Society of America, 103:1 January, pp. 588-601, 1998.

[10] A. Pienimaki, "Indexing Music Databases Using Automatic Extraction of Frequent Phrases," In Proc. 3rd International Symposium on Music Information Retrieval (ISMIR), pp. 25-30, 2002.

[11] J. Downie. "Evaluating a Simple Approach to Music Information Retrieval: Conceiving Melodic N-Grams as Text," Thesis, University of Western Ontario, 1999.

[12] J. Reiss, J. Aucouturier and M. Sandler, "Efficient multidimensional searching routines for music information retrieval," In Proc. 2nd International Symposium on Music Information Retrieval (ISMIR), pp. 163-171, 2001.

[13] K. Ioannis, N. Alexandros, Apostolos N. Papadopoulos, M. Yannis, "Audio Indexing for Efficient Music Information Retrieval," International Multimedia Modelling Conference, pp. 22-29, 2005.

[14] K. Andreas, "Themefinder: A Web-based Melodic Search Tool, Melodic Similarity: Concepts, Procedures and Applications," Computing in Musicology, pp. 231-236, 1998.

[15] C. Liu, J. Hsu and A. L. P. Chen, "Efficient Theme and Non-trivial Repeating Pattern Discovering in Music Databases," In Proc. 15th International Conference on Data Engineering (ICDE '99), pp. 14-21, 1999.

[16] 노승민, 박동문, 황인준, "사용자 질의 패턴을 이용한 효율적인 오디오 색인 기법", 한국 정보과학회 논문지, 제31권, 제1호, pp. 143-153, 2004.

[17] R. B. Dannenberg, N. Hu, "Understanding Search Performance in Query-by-humming Systems," In ISMIR, 2004.

유 진 희
2004년 8월 한성대학교 정보시스템공학과 졸업(학사). 2005년 3월~현재 연세대학교 컴퓨터과학과 석사과정. 관심분야는 멀티미디어 데이타베이스, 데이타마이닝

박 상 현
정보과학회논문지 : 데이타베이스
제 34 권 제 1 호 참조