말소리와 음성과학 제4권 제1호 (2012)
pp. 77~87

오디오 Fingerprint를 이용한 음악인식 연구 동향

Music Recognition Using Audio Fingerprint: A Survey

이 동 현[1] · 임 민 규[2] · 김 지 환[3]
Lee, Donghyun · Lim, Minkyu · Kim, Ji-Hwan

ABSTRACT

Interest in music recognition has been growing dramatically after NHN and Daum released their mobile applications for music recognition in 2010. Methods in music recognition based on audio analysis fall into two categories: music recognition using audio fingerprint and Query-by-Singing/Humming (QBSH). While music recognition using audio fingerprint receives music as its input, QBSH involves taking a user-hummed melody. In this paper, research trends are described for music recognition using audio fingerprint, focusing on two methods: one based on fingerprint generation using energy difference between consecutive bands and the other based on hash key generation between peak points. Details presented in the representative papers of each method are introduced.

Keywords: Music recognition, audio fingerprint, band energy, peak point, hash key

1. 서론

최근 스마트폰의 보급 증대에 힘입어, 음악을 쿼리(query)로 이용하는 음악 검색이 각광을 받고 있다. 기존의 음악검색은 주로 곡목/가수명 등의 메타 데이터를 이용하여 검색하는 방법이었다. 이 방법은 사용자가 검색을 원하는 곡의 곡명 및 가수 명을 모르는 경우 검색을 할 수 없고, 수동으로 메타 데이터를 생성해야 하는 단점이 있다. 이에 반해, 음악을 쿼리로 이용하는 음악 검색은 마이크로부터 입력된 곡의 일부 또는 사용자의 singing이나 humming만으로 검색이 가능한 장점이 있다.

음악을 쿼리로 이용하는 음악검색 방법은 1999년에 Gracenote Inc.의 Disc Recognition Service(DRS) [1]와 2005년 Melodis Inc.(현 Soundhound Inc.)의 Midomi [2]가 PC를 기반으로 상용화 되었다. Gracenote의 DRS는 사용자의 CD-ROM 드라이브에 있는 오디오 CD의 오디오 정보를 인터넷을 통해 Gracenote의 서버에 전송하여, 서버 내의 음악에서 검색된 오디오 정보에 대한 노래곡명을 알려주는 서비스이다. Melodis의 Midomi는 사용자의 singing이나 humming을 서버로 전송하여 singing/humming 데이터베이스를 구축하고, 이 데이터베이스에 기반한 singing/humming 검색 서비스이다. 이 서비스들은 PC상에서 사용하는 것을 가정하였기 때문에, 이동성에 제약이 많았고, PC에 마이크가 없거나, 녹음 프로그램을 구비하지 못한 사용자의 경우 사용이 힘든 단점이 있었다. 휴대폰 상에서의 음악검색 서비스로는 2002년에 Shazam Entertain-ment Ltd.가 특정번호로 전화연결 시, 이동전화망을 통해 사용자가 찾으려는 음악을 들려주면, SMS로 검색 결과의 음악명을 알려주는 2580 서비스(영국 내 접속번호가 2580이었음)를 선보였다 [3]. 하지만 2G망을 사용한 이 당시에는 검색결과 및 추천곡의 다운로드시 비용과 시간이 너무 많이 들었고, 또한 스마트폰이 본격적으로 보급되기 이전이어서 휴대폰 단말기 제조사별로 플랫폼이 달라 사용자에게 편리한 애플리케이션 UI를 제공하지 못했다 (전화상의 자동응답 시스템이었음). 이로 인해 비즈니스 모델이 검색 결과에 대해 전화 요금을 부과하는 방식에 국한될 수밖에 없었다 (약 850원/건). 따라서 사용자들이 음악검색 비용에 부담을 느끼게 되어 폭넓게

1) 서강대학교, redizard@sogang.ac.kr, 제1저자
2) 서강대학교, lmkhi@sogang.ac.kr, 제2저자
3) 서강대학교, kimjihwan@sogang.ac.kr, 교신저자

접수일자: 2012년 1월 25일
수정일자: 2012년 3월 20일
게재결정: 2012년 3월 22일

그림 1. 스마트폰에서 상용화된 음악 검색 앱들의 실행 화면
(a) MusicID (Gracenote Inc.) (b) SoundHound (Soundhound Inc.) (c) Shazam (Shazam Entertainment Ltd.)
(d) Daum 음악검색 앱 (e) 네이버 음악검색 앱
Figure 1. Screenshots of commercialized music recognition apps. in smartphone
(a) MusicID (Gracenote Inc.) (b) SoundHound (Soundhound Inc.) (c) Shazam (Shazam Entertainment Ltd.)
(d) Daum music recognition app. (e) Naver music recognition app.

이용되지 못했다.

스마트폰이 본격적으로 보급된 2008년경부터 표준화된 플랫폼을 제공하는 안드로이드 OS나 iOS를 통해서 애플리케이션 제작이 쉬워졌다. 또한 3G망을 이용한 데이터 전송이 가능해 져서 검색된 음악을 바로 다운로드 받거나 추천 음악에 대한 다운로드가 가능해졌다. 이로 인해 음악 검색에 대해서는 요금을 부과하지 않고 검색된 음악 또는 추천곡에 대해 다운로드하는 경우에만 요금을 부과하는 비즈니스 모델의 구현이 가능해졌다. 현재 이러한 비즈니스 모델을 기반으로 음악검색 서비스를 제공하는 대표적인 애플리케이션으로 외국에서는 2008년에 Shazam Entertainment Ltd.가 Shazam 앱을 출시했고 [4], Gracenote Inc.가 MusicID 앱을 출시했으며 [5], 2009년에 Soundhound Inc.가 SoundHound 앱을 선보였다 [6]. 국내에서는 2010년에 Daum [7]과 2011년에 NHN [8]에서 음악검색 앱이 출시된 바 있다.

음악 검색은 입력 음악 세그먼트와 이에 대응되는 데이터베이스에 저장된 음악과의 음향학적 유사도에 따라 fingerprinting(유사도 높음)에서 시작하여 remix/sampling detection, cover song detection/QBSH(유사도 상대적 낮음)의 순으로 분류된다 [9]. 본 논문에서는 fingerprinting 방법에 기반한 음악 검색에 대하여 대표 논문을 중심으로 연구 방법을 기술한다.

<그림 1>에서 제시한 다섯 가지 대표적인 음악검색 앱 중, Daum과 네이버의 앱은 아직까지 적용한 기술에 대해서 구체적인 내용이 검색되지 않고 있다. SoundHound 앱은 singing/humming을 입력 받는 QBSH 시스템이어서, 본 논문에서 관심을 가지고 있는 fingerprinting에 기반한 방법은 아니다. 문헌조사 결과, Shazam과 MusicID는 오디오 fingerprint를 이용하여 구현되어 있음을 알 수 있다. Shazam의 경우, fingerprint 생성 및 검색에 관한 내용은 [14]-[16]에 정리되어 있다. MusicID의 경우, Gracenote Inc.가 Philips Electronics N.V.로부터 '강인한 오디오 식별 및 fingerprinting 기술' (Robust audio identification and fingerprinting technology)에 관한 특허들을 양수받고, 이를 기반으로 MusicID가 개발되었다 [10]. 이들 특허들 중에서 fingerprint 생성 및 검색에 관한 내용은 [11]에 기술 되어 있고, 특허와 동일한 기술 내용이 논문으로 [12], [13]에 기술되어 있다. 본 논문에서는 이들 두 시스템에 대한 대표 논문을 각각 2장과 3장에서 자세히 소개한다.

본 논문의 구성은 다음과 같다. 2장에서는 [12]의 내용을 중심으로, 밴드 간 에너지 차를 이용한 fingerprint 생성에 기반한 방법에 대해서 세부적인 구현 방법을 기술한다. 3장에서는 [15]의 내용을 위주로, peak point들 간의 해쉬 키 생성에 기반한 방법을 설명한다. 4장에서는 결론을 맺는다.

2. 밴드 간 에너지 차를 이용한 fingerprint 생성에 기반한 방법

본 장에서는 [12]의 내용을 중심으로 밴드 간 에너지 차를 이용한 fingerprint 생성에 기반한 방법에 대해 기술한다.

2.1 Fingerprint 추출

전체 오디오 신호를 일정한 길이 l의 프레임들로 나누어 fingerprint를 추출한다 ([12]에서 l=0.37초). 주파수 분석을 하기 위하여, 각각의 프레임들에 대해 윈도우 함수 ([12]에서는 해닝 윈도우를 적용)를 적용하고, 푸리에 변환 (Fourier transform)을 거친다. 프레임 별 생성되는 fingerprint 값의 급격한 변화를 방지하기 위하여 프레임 간 중복을 두고 길이 s만큼의 프레임 쉬프트를 한다 ([12]에서 중복 비율 (overlap factor)은 31/32이고, 프레임 쉬프트는 s=11.6ms(=0.37sec *1/32)임). 주파수 분석 결과를 b개의 주파수 밴드 (frequency

band)로 나누어 각 밴드 별 에너지를 추출한다 ([12]에서는 Bark scale에 따라 b=33으로 설정). n번째 프레임의 m번째 밴드에 대한 에너지가 $E(n,m)$이라 할 때, (n, m)에 대한 sub-fingerprint의 비트 값인 $F(n,m)$은 [12]에서 식 (1)과 같이 결정 가능하다.

$$F(n,m)=\begin{cases}1 & \text{if } E(n,m)-E(n,m+1)\\ & -(E(n-1,m)-E(n-1,m+1))>0\\ 0 & \text{if } E(n,m)-E(n,m+1)\\ & -(E(n-1,m)-E(n-1,m+1))\le 0\end{cases} \quad (1)$$

식 (1)을 적용하는 경우, $F(n,m)$을 구하기 위해 $E(n,m+1)$과 $E(n-1,m+1)$가 필요하므로 m의 범위는 $1 \le m \le b-1$가 된다. [12]의 경우, 생성되는 sub-fingerprint는 32-bit가 된다.

<그림 3>은 입력 음악 세그먼트에 대해서 생성된 fingerprint block을 도식화한 것이다. Fingerprint block은 프레임 단위로 생성된 sub-fingerprint의 열이다. 하얀색 픽셀은 $F(n,m)$이 1인 경우이고, 검은색 픽셀은 $F(n,m)$이 0인 경우를 나타낸다. 예를 들어, 입력 음악 세그먼트의 길이가 3초이고, s=11.6ms, b=33라 할 때, fingerprint block을 구성하는 sub-fingerprint의 개수는 256 (≒3sec/11.6ms)이 되어 256×(32bit)로 구성된 fingerprint block이 생성된다.

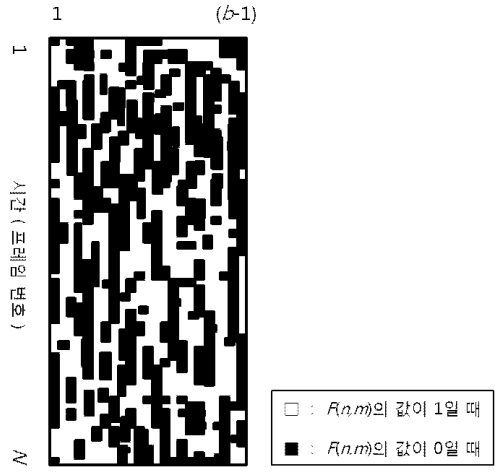


그림 3. 입력 음악 세그먼트에서 추출된 fingerprint block의 예
Figure 3. Example of extracted fingerprint block from input music segment

2.2 음악 데이터베이스 구성

<그림 4>는 총 N개 음악의 fingerprint 정보로 이루어진 fingerprint 데이터베이스의 구성도이다. Fingerprint 데이터베이스는 2.1장에서 기술한 방법에 따라 총 N개 음악에 대하여 추출된 fingerprint를 저장하는 부분과, 동일한 sub-fingerprint들을 연결한 링크드 리스트의 헤더위치를 저장하는 Look-Up Table (LUT)로 구성되어 있다. LUT의 index는 (b-1) bit의 sub-fingerprint로 표현될 수 있는 0~$2^{(b-1)}$-1의 범위를 가진다. 동일한 sub-fingerprint들이 링크드 리스트 형태로 연결된다. LUT의 각 element는 그 element의 index와 같은 값을 가지는 sub-fingerprint에 해당되는 링크드 리스트의 헤더 위치를 저장한다. <그림 4>에서 음악 1과 음악 N은 각각 0x21353115의 동일한 값의 sub-fingerprint를 가지고 있다. 이들 sub-fingerprint들은 링크드 리스트로 연결되어 있다. 이 링크드 리스트의 헤더 노드를 LUT의 index가 0x21353115인 element가 가리키고 있다.

2.3 검색 알고리즘

입력 음악(음악 세그먼트)에 대해 2.1장에서 기술된 방법에 따라 fingerprint block을 추출한다. Fingerprint block의 첫 번째 프레임에서 생성한 sub-fingerprint를 index로 하는 LUT의 링크드 리스트를 방문한다. 입력 음악 세그먼트에서 f개의 프레임을 추출한다. LUT에서 첫 번째 프레임의 sub-fingerprint를 index로 가지는 링크드 리스트를 검색한다. 검색된 링크드 리스트의 각 노드가 가리키고 있는 sub-fingerprint를 기점으로, 길이 f의 fingerprint block과 입력 음악 세그먼트에서 추출한 프레임들을 fingerprint block과의 비트 오류 비율(BER: Bit Error Rate)로써 계산한다. BER이 임계값 T보다 작은 경우 이에 해당하는 fingerprint block이 포함된 곡을 검색 결과로 선택하고, T보다 큰 경우 링크드 리스트의 그 다음 노드에 해당하는 sub-fingerprint부터 시작되는 fingerprint block과의 BER을 계산한다. 연결된 링크드 리스트의 모든 sub-fingerprint에서 검색결과가 나오지 않으면 입력음악 세그먼트의 fingerprint block에서 다음번 sub-fingerprint에 대해서 위와 동일한 작업을 수행한다. 입력음악 세그먼트의 fingerprint block내의 모든 sub-fingerprint에 대해서 검색 결과가 나오지 않을 경우 인식 거부(rejection)을 발생시킨다. <그림 4>에서는 약 3초길이 (f=256)의 음악 세그먼트가 질의되었다. 입력 음악 세그먼트의 첫 번째 sub-fingerprint인 0xA11929EB를 index로 하는 LUT에 연결된 링크드 리스트의 각 노드를 방문한다. 방문한 노드들이 가리키고 있는 sub-fingerprint를 기점으로 한 길이 256의 fingerprint block과 질의된 음악 세그먼트에서 추출된 fingerprint block과의 BER을 계산한다. <그림 4>에서 음악 1

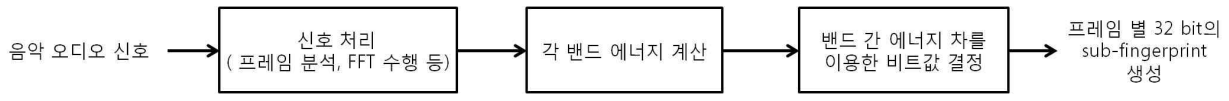


그림 2. [12]에서의 fingerprint 추출의 개요
Figure 2. Overview of fingerprint extraction in [12]

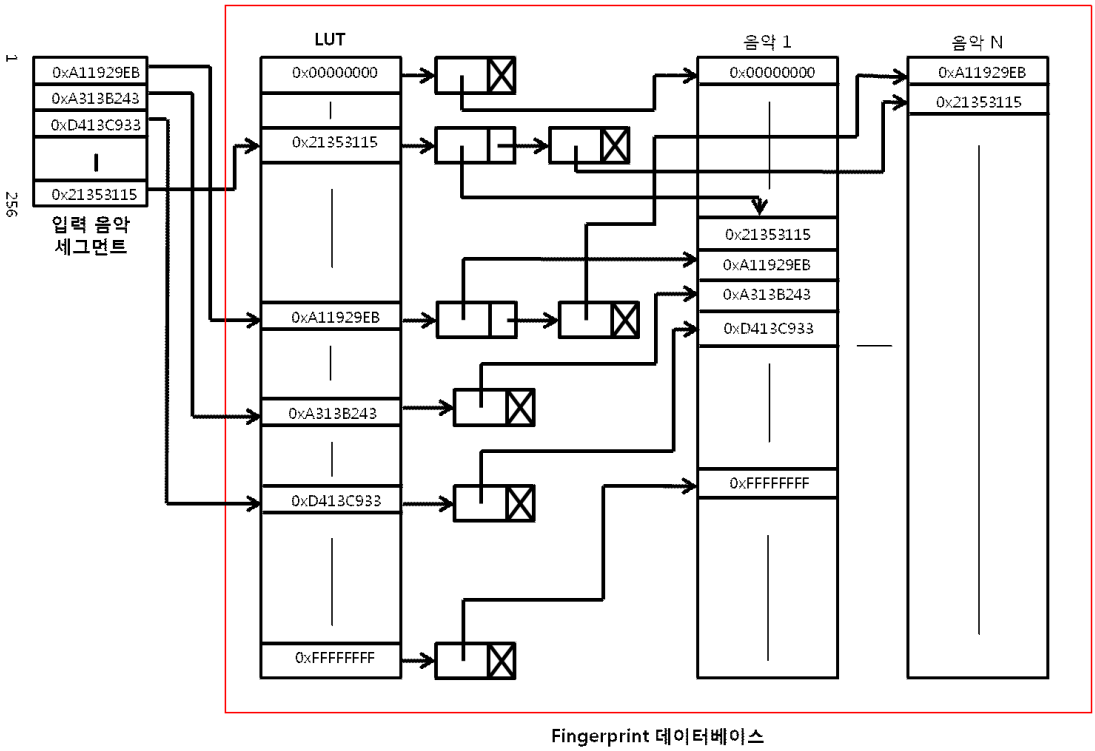


그림 4. Fingerprint 데이터베이스 구성도
Figure 4. Conceptual diagram of fingerprint database

과 음악 N 모두 0xA11929EB인 sub-fingerprint가 존재하지만, 음악 1과의 BER이 임계값 T 보다 작은 경우 음악 1을 검색 결과로 출력하고 검색을 중단한다. <그림 4>의 경우, 음악 1이 검색 결과로 선택된다.

<그림 1>에서 소개된 앱들은 주변에서 들려오는 음악을 휴대폰의 마이크를 통해 녹음하고, 녹음된 음악 세그먼트를 서버에 질의한다. 따라서 채널 잡음 및 주변 잡음에 의해 sub-fingerprint들의 bit가 바뀔 수 있다. 이 경우 위에서 기술한 검색 알고리즘은 잡음 등의 영향으로 입력된 음악 세그먼트의 fingerprint block에서 첫 번째 sub-fingerprint의 bit가 손상되게 되어, 의도한 링크드 리스트의 노드가 아닌 다른 링크드 리스트의 노드들을 방문하게 되는 문제점이 있다. 이 경우 찾아야 하는 fingerprint block이 BER 비교 대상으로 검토되지도 않는 문제가 발생한다.

이 문제를 해결하기 위해서는 손상된 sub-fingerprint의 특정 bit들을 toggle하여 sub-fingerprint 후보를 생성하고, 이들 후보에 대해서도 LUT를 탐색한다. 이 경우 최대 b-1 bit의 toggle을 허용하게 되면 후보 sub-fingerprint가 2^{b-1} 개가 생성되어 검색속도가 매우 느려지게 된다. 따라서 손상될 가능성이 높은 bit들을 선정하는 방법이 필요하다. [12]에서는 sub-fingerprint의 32개 bit들 중에서 어떤 bit를 toggle 시킬지 결정하기 위해 2.1장의 식 (1)의 밴드 에너지 차를 고려한다. 식(1)에서 $E(n,m)$-$E(n,m+1)$-($E(n-1,m)$-$E(n-1,m+1)$) = $D(n,m)$ 라 하자. 원곡의 sub-fingerprint의 bit를 결정지을 때, $|D(n,m)|$가 작을수록 추후 잡음 하에서 동일한 sub-fingerprint를 추출 시, $D(n,m)$의 부호가 바뀌게 되어 해당 bit값의 변화가 있을 확률이 크고, $|D(n,m)|$의 값이 클수록 bit 값이 변경될 확률이 작다고 가정한다. 이에 따라 $|D(n,m)|$값이 0과 가까운 순서대로 32개의 bit들을 정렬하여 상위 M개 bit들에 대해서만 toggle을 수행한다([12]에서는 M=10). 상위 M개 bit들에 대해 toggle을 수행한 결과로 총 2^M개의 sub-fingerprint 후보가 생성된다. 생성된 2^M개의 sub-fingerprint 후보를 index로 하는 LUT에 연결된 링크드 리스트의 노드들을 차례대로 방문한다. 2^M개의 sub-fingerprint 후보를 index로 하는 링크드 리스트를 모두 탐색하여도 검색결과가 나오지 않을 경우, fingerprint block의 다음 sub-fingerprint로 검색을 수행한다.

3. Peak point들 간의 해쉬 키 생성에 기반한 방법

본 장에서는 peak point들 간의 해쉬 키 생성에 기반한 방법에서 적용된 알고리즘을 기술한다. 이 방법에서는 스펙트럼 상에서의 각 (시간, 주파수)로 이루어진 point들에 대해서, 특정 범위 내에서 가장 높은 에너지를 가지는 peak point들을 선정한다. 이후 이들 peak point들 간의 해쉬 키를 생성한다. 주변 잡음 수준이 peak point 선정에 영향을 줄 정도로 잡음이

크지 않다면, 잡음 환경 하에서 녹음한 음악으로부터 추출되는 해쉬 키가 원곡에서 추출한 해쉬 키와 동일하기 때문에, 이 방법이 잡음환경에 대해서 보다 강인하다. 음악에 대한 fingerprint는 해쉬키의 열로 구성되고, 각 해쉬 키는 두 peak point의 주파수 정보, 시간차 및 해당 곡의 곡 번호, 곡 시작으로부터의 위치정보로 구성된다. 음악 검색 시스템의 메모리 요구량 및 탐색 시간은 해쉬 키의 수에 따라 결정된다. 메모리 요구량과 탐색시간을 줄이기 위해 peak point들 중 일부를 anchor point로 선정한다. 각각의 선정된 anchor point로부터 peak point와의 해쉬 키를 생성하여 전체 해쉬 키의 개수를 줄인다. 3.1장에서는 peak point 생성 방식에 대해 기술한다. 3.2장에서는 생성된 peak point로부터 해쉬 키를 생성하는 방법에 대해 기술 한다. 3.3장에서는 해쉬키의 수를 줄이기 위하여 peak point 중 anchor point를 선정하는 방법과 target zone선정 방법에 대해 기술한다. 3.4장에서는 해쉬 테이블 생성 방법에 대하여 기술한다. 3.5장에서는 입력 음악에 대한 검색과정에 대해 기술한다.

3.1 Peak point 생성

Peak point를 생성하는 과정은 다음과 같다. 먼저 푸리에 변환을 이용하여, 입력된 음악 파일을 시간 축과 주파수 축으로 이루어진 2차원 평면 공간상의 스펙트로그램 (spectrogram)으로 변환한다. 스펙트로그램 상의 모든 점들은 (프레임 번호, 주파수 bin index)의 pair로 이루어진다. Frame 번호는 시간 축상에서 index이다. 예를 들어 프레임 한 개의 길이가 20ms, 프레임 쉬프트가 10ms인 경우, 입력 음악에서 0.5초에서 0.52초의 구간은 51번째 프레임이 되고, 이 프레임에 대해서 스펙트로그램상의 주파수 축에 대해 bin이 512개가 있는 경우, 대역 주파수가 작은 bin부터 큰 bin의 순서로 1~512의 index가 부여된다. 초당 sampling 수가 16,000이고, FFT point가 1024인 경우, 스펙트로그램의 전체 대역은 8,000Hz가 되며 bin의 총 개수는 512가 된다. 스펙트로그램상의 모든 점들에 대해서 주위의 다른 점들 보다 높은 에너지를 가지는 점을 peak point로 선택한다. 이때, 비교 대상이 되는 주위의 크기는 구현에 따라 다르다.

생성된 peak point들을 시간 축과 주파수 축으로 이루어진 2차원 평면 공간상에 표시하는 경우, <그림 5>과 같은 결과를 얻을 수 있다. <그림 5>의 그림을 peak point 맵(peak point map)으로 명명한다. peak point 맵은 시간-주파수 공간상에서 선택된 peak point의 위치정보를 보여준다. peak point 맵 상의 peak point들은 각 point를 기준으로 주위보다 높은 에너지를 가지는 점들이기 때문에, 잡음이 peak point의 위치에 영향을 줄 정도가 아니라면, 같은 음악에 대해 동일한 별자리표가 생성된다. 따라서 이 peak point들을 이용하여 검색을 수행하면, 잡음에 강인한 음악 검색의 구현이 가능하다.

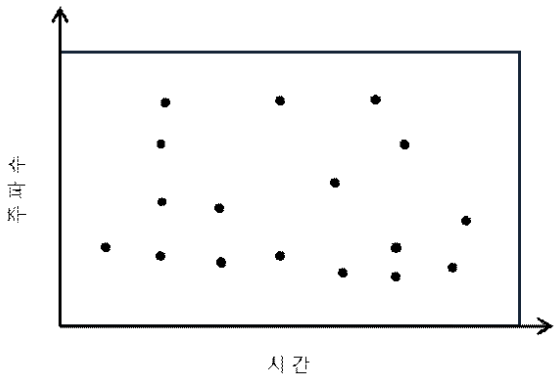


그림 5. 시간-주파수 공간 상 peak point 맵
Figure 5. A peak point map in time-frequency domain

3.2 해쉬 키 생성

본 장에서는 peak point들 간에 해쉬 키를 생성하는 방법에 대해서 기술한다. <그림 6>과 같이 두 개의 peak point (t_1, f_1), (t_2, f_2)가 있다고 가정하자. 여기서 t_1, t_2는 3.1장에서 기술한 시간 축 상의 index 값이다. f_1, f_2는 주파수 bin의 index 값이다. 이들 두 개의 peak point로부터 (f_1, f_2, t_2-t_1)으로 구성되는 해쉬 키를 생성한다. 예를 들어, t_1 = 4, f_1 = 200, t_2 = 52, f_2 = 327 이면 생성되는 해쉬 키는 [200, 327, 48]이다. 해쉬 키의 포맷 결정시, 각 필드별 할당되는 비트의 수는 bin의 개수와 t_2-t_1의 가능한 범위에 따라 다르다. 예를 들어 bin의 개수가 최대 512개라면, f_1을 저장하기 위해 9bit가 할당되고, 마찬가지로 f_2의 저장을 위해 9bit가 필요하다. Peak point들 간의 pair 생성을 1초 이내의 범위로 국한한다면, $t_2 - t_1 < 100$이 되어, 7bit를 할당하면 된다. 따라서 해쉬 키 저장을 위해 최소한 25bit (9 bit + 9 bit + 7 bit)이 필요하다.

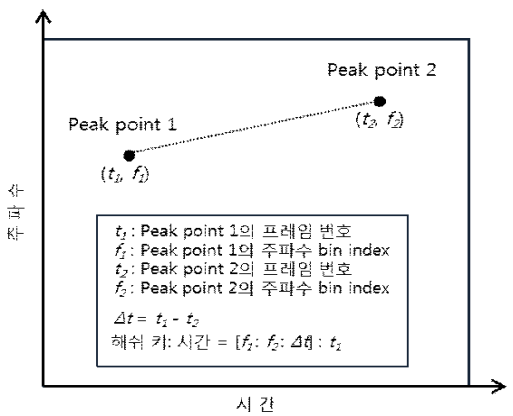


그림 6. 두 peak point들의 주파수와 두 peak point간의 시간 차이를 이용한 해쉬 키 생성 방법
Figure 6. Hash key generation using two peak points and the time difference between these points

3.3 Anchor point 결정 및 target zone 설정

전체 peak point의 수가 n 개라면, 모든 peak point들이 pair를 형성할 때 생성되는 해쉬 키의 수는 $O(n^2)$이 된다. 생성되는 해쉬 키의 수가 많으면 음악을 보다 정밀하게 표현 할 수 있다는 장점이 있지만, 많은 양의 메모리를 사용하고 검색에 많은 시간을 소모하는 단점이 있다. Peak point 중 특정 조건을 만족하는 peak point들을 anchor point로 선택하고, 이들 anchor point들과 peak point들과의 pair로 해쉬키를 생성하면 해쉬 키의 숫자를 줄일 수 있다. 만족해야 하는 특정 조건에 대한 세부적인 방법은 [15]에 제시되어 있지 않다. 따라서 실제 시스템 구현 시 anchor point 선정 방법을 결정해야 한다. Anchor point 선정의 두 가지 예는 다음과 같다. 하나의 방법은 시간 축 상에서의 일정 구역마다 발생한 peak point 중에서 주파수 에너지 값을 기준으로 상위 peak point들을 anchor point로 선정하는 것이다. 또 다른 방법은, 시간 축 상에서의 일정 구간마다 주파수 대역을 나누고 각 주파수 대역마다 주파수 에너지 값을 기준으로 상위 peak point를 anchor point로 선정하는 것이다. 이 외에도 anchor point를 선정하는 방법이 여러 가지가 있다. Anchor point 선정 방법에 따라 성능지표(메모리요구량, 반응속도, 인식률 등)가 영향을 받기 때문에 anchor point 선정 방법의 결정은 많은 주의가 요구된다.

해쉬 키의 수를 줄이는 다른 방법은 target zone을 설정하여 anchor point와 target zone 내의 peak point들의 pair로만 해쉬 키를 생성하는 것이다. <그림 7>은 하나의 anchor point에 대해 target zone을 설정하고, 이 anchor point와 target zone 내의 peak point들이 pair를 생성하는 것을 보여준다. Target zone은 시간/주파수의 2차원 평면에서 가로의 길이가 TL_t, 세로의 길이가 FL_t인 구역으로 정의한다. TD_t를 anchor point와 target zone 사이의 거리라 정의하면, 하나의 anchor point (t_a, f_a)에 대해 시간 축 상으로 t_a+TD_t에서 $t_a+TD_t+TL_t$, 주파수 축으로 $f_a-FL_t/2$에서 $f_a+FL_t/2$의 구역으로 정의된다. TD_t의 값과 TL_t의 값, FL_t의 값의 결정에 따라 성능지표에 영향을 받기 때문에 이 값 결정에 많은 주의가 요구된다.

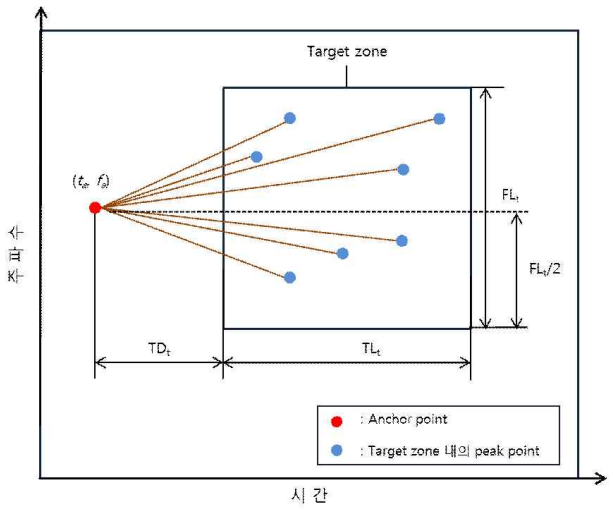


그림 7. Anchor point에 대한 target zone 설정 및 anchor point와 target zone 내의 peak point들과의 pair 생성

Figure 7. Determination of target zone for an anchor point and pair generation between the anchor point and peak points in its target zone

3.4 해쉬테이블 생성

해쉬테이블을 구성하기 위해 모든 음악 트랙(track)에 대해서 3.2절에서 기술한 방식에 따라 해쉬 키를 생성한다. 음악 트랙 ID별로 해쉬 키를 생성하고, 각 해쉬 키마다 해당 트랙 ID와 곡 시작으로부터 해쉬 키 간의 time offset을 부여하여 (해쉬 키, 트랙 ID, time offset) pair들을 해쉬 파일에 저장한다. 해쉬테이블은 각 음악 트랙별로 생성된 모든 해쉬 파일로부터 구성한다.

해쉬테이블 구성 시, 값(value)으로 (트랙 ID, time offset)이 이용된다. 해쉬테이블의 버킷의 개수는 생성가능한 모든 해쉬 키의 가짓수가 된다. 예를 들어, FFT point를 1,024로 하여 주파수 bin의 수가 512개이고 FL_t가 64이며, anchor point와 peak point의 pair 생성을 프레임 수 64개 이내의 범위로 국한한다면, 해쉬 키는 21bit (9bit + 6bit + 6bit)로 구성되며 총 버킷의 개수는 221이다. 해쉬 키가 해쉬테이블의 인덱스가 되도록 한다. 각 버킷에 해당되는 값인 (trackID, time offset)이 슬롯을 구성한다. 슬롯의 열이 variable length array로 저장되고, 이 array의 시작주소와 array의 크기가 버킷에 저장된다. 따라서 버킷의 개수가 221개이며, array의 시작주소 저장에 4 바이트, array 크기 저장에 4 바이트 할당할 경우 해쉬테이블에 필요한 메모리는 총 16M 바이트가 된다.

슬롯의 저장에 필요한 메모리요구량은 전체 해쉬의 수 N_{hash}와 (트랙 ID, time offset)을 저장하기위한 메모리 공간의 크기에 비례한다. 전체 400만곡, 각 곡 당 최대 길이를 40분으로 가정 할 경우, 각 슬롯 당 필요한 메모리는 trackID 저장에 22 비트, time offset 저장에 18 비트가 필요하므로 5 바이트가 필요하다. 또한 한 곡의 길이가 평균 5분이며 1초당 anchor point의 개수가 8개, anchor point당 target zone내의 peak point가 평균 8개일 경우 곡당 약 19,200개의 해쉬 키가 생성된다. 따라서 400만곡에서 생성되는 모든 해쉬 키의 (트랙 ID, time offset)를 저장하기 위해서는 약 384GB의 메모리 공간이 필요하다.

<그림 8>은 트랙 ID가 ID_j와 ID_k인 음악으로부터 해쉬테이블이 구성되는 과정을 보여준다. 각 음악에서 추출된 해쉬 키로부터 트랙 ID 및 time offset이 부여된 해쉬 pair들이 생성되어 파일로 저장되고, 생성된 해쉬 파일로부터 각 pair들을 읽어 들여 해쉬테이블이 구성된다. <그림 8>에서는 ID_j의 anchor point a_1과 peak point a_2로부터 생성된 해쉬 키와 ID_k의

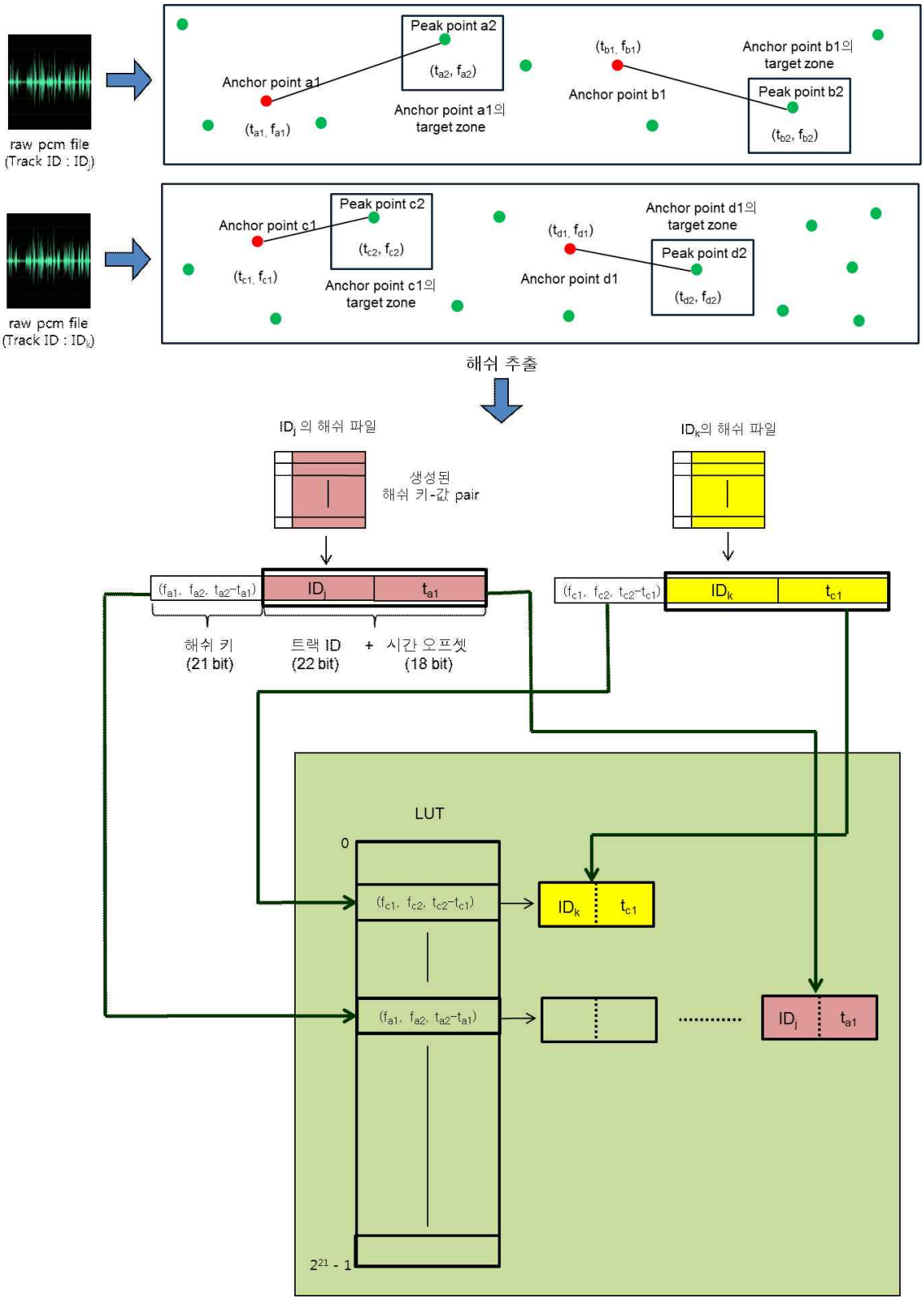
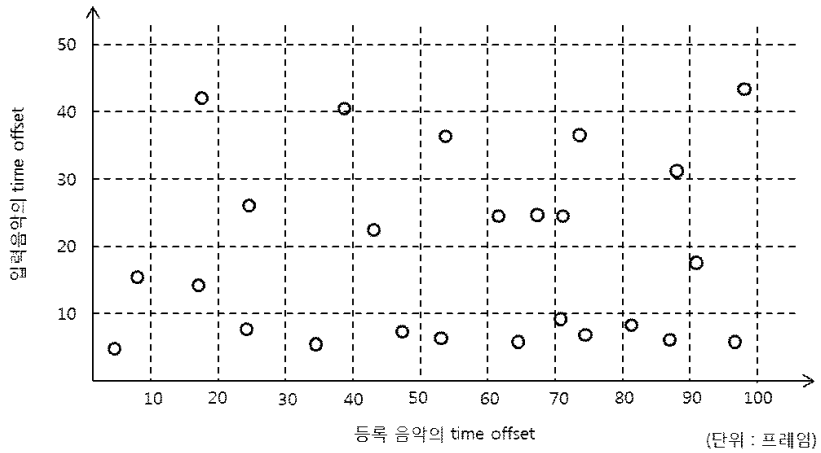
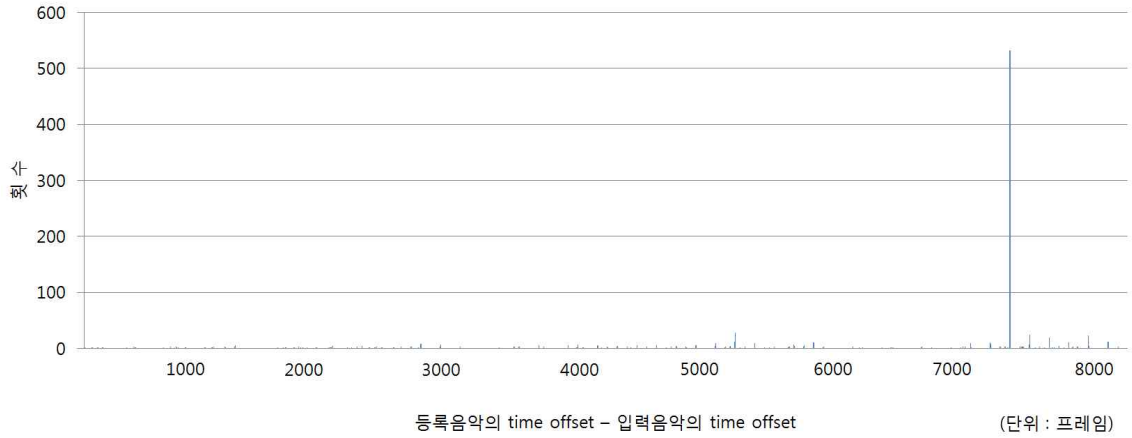


그림 8. 해쉬 테이블 생성 과정

Figure 8. Hash table generation process



(a)



(b)

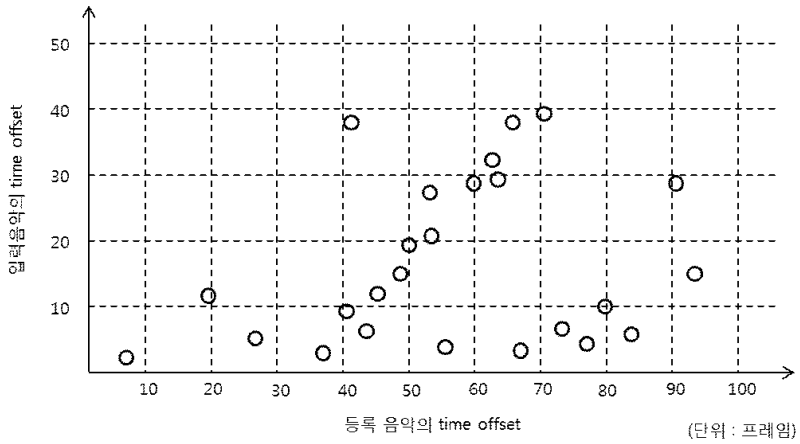
그림 9. 등록음악과 입력받은 음악이 잘 정합되는 경우
(a) 등록음악의 time offset과 입력음악의 time offset간의 scatterplot
(b) (등록음악의 time offset - 입력음악의 time offset)의 히스토그램
Figure 9. In case of well‐matching to the input query.
(a) Scatterplot of track time offset versus input query time offset
(b) Histogram of (track time offset - input query time offset)

anchor point c_1과 peak point c_2로부터 생성된 해쉬 키가 해쉬 테이블에 위치하는 것을 보여주고 있다. ID_j의 경우 해쉬 키가 (f_{a1}, f_{a2}, t_{a2}-t_{a1})이므로 해쉬테이블에서 이와 동일한 인덱스에 위치하는 버킷이 가리키는 배열의 메모리를 추가 할당하여 가장 마지막 슬롯에 (ID_j, t_{a1})을 저장한다. ID_k의 경우 해쉬가 (f_{c1}, f_{c2}, t_{c2}-t_{c1})이므로 이와 동일한 인덱스에 위치하는 버킷이 가리키는 배열에 슬롯이 하나도 없으므로 하나의 슬롯만큼의 메모리를 할당하여 해당 위치에 (ID_k, t_{c1})을 저장한다.

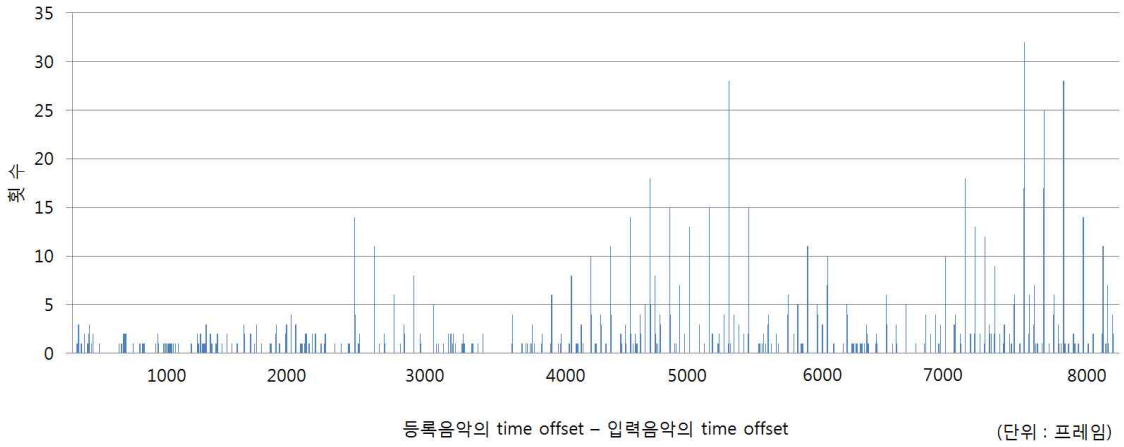
3.5 검색 과정

입력으로 들어온 쿼리에 대해서 3.2 장에서 언급한 방식과 동일한 방법으로 해쉬 키를 추출한다. 입력된 쿼리에 대해서는 (해쉬 키, 트랙 ID, time offset) pair가 아닌 (해쉬 키, time offset) pair를 추출한다. 추출된 각 pair들의 해쉬 키를 인덱스로 하여 해쉬테이블을 탐색한다. 해당 버킷에 연결된 모든 슬롯들을 방문하여 (트랙 ID, time offset) 정보들을 추출한다. 추출된 정보로부터 각 트랙 ID 별로 배열을 생성한다. 각 슬롯에서 추출된 time offset과 입력된 쿼리의 시작으로부터의 time offset 정보를 pair로 해서 해당 트랙 ID의 배열에 저장한다. 위 과정을 입력받은 쿼리에서 얻은 모든 해쉬 키에 대해 수행한다.

위 과정이 완료되면 생성된 모든 배열들을 검사하여, 각 배열에 저장된 time offset pair(곡의 시작으로부터의 time offset, 쿼리의 시작으로부터의 time offset) 정보로부터, 모든 해쉬 키에 대해서 (등록음악의 time offset, 입력음악의 time offset)을 점으로 나타낸 scatterplot을 그린다. <그림 9-(a)>는 등록음악



(a)



(b)

그림 10. 등록음악과 입력받은 음악이 정합되지 않는 경우
(a) 등록음악의 time offset과 입력음악의 time offset간의 scatterplot
(b) (등록음악의 time offset - 입력음악의 time offset)의 히스토그램
Figure 10. In case of poorly – matching to the input query.
(a) Scatterplot of track time offset versus input query time offset
(b) Histogram of (track time offset - input query time offset)

과 입력받은 음악이 잘 정합될 때의 scatterplot이며, <그림 10-(a)>는 등록음악과 입력받은 음악이 정합되지 않는 경우의 scatterplot이다. <그림 9-(a)>에서는 대각선 모양으로 일치하는 패턴이 존재한다. 이는 입력받은 쿼리에서 얻은 해쉬 키들의 time offset과 이들과 일치하는 음악으로부터 생성된 해쉬 키들의 time offset과의 차가 모두 동일하기 때문이다. 따라서 scatterplot 상에서 나타나는 점들이 나타내는 대각선의 기울기가 1이 된다. <그림 10-(a)>에서는 대각선 모양으로 일치하는 패턴이 존재하지 않는다. 이는 입력받은 쿼리와 일치하지 않는 곡에 대해서는 등록음악에서의 해쉬 키와 입력음악의 해쉬 키에 대한 시간오프셋의 차이가 일정치 않기 때문이다.

Scatterplot 상의 모든 점들에 대해 각 점들의 x-좌표 (등록음악에서 추출한 해쉬 키의 시간 오프셋)와 y-좌표 (입력 음악에서 추출한 해쉬 키의 시간 오프셋) 값의 차를 계산한다. 이 값을 모두 계산한 후, 이들을 오름차순으로 정렬한다. 정렬시킨 값들에 대해서 몇 번 나타났는지 개수를 세어본다. 그 결과를 히스토그램으로 나타낸다. 히스토그램에서 x축은 (등록음악에서 추출한 해쉬 키의 시간 오프셋 - 입력음악에서 추출한 해쉬 키의 시간 오프셋)이며 y축은 x축 상의 값들이 나타난 횟수를 나타낸다. <그림 9-(b)>는 등록음악과 입력받은 음악이 잘 정합될 때의 히스토그램이며, <그림 10-(b)>는 등록음악과 입력받은 음악이 정합되지 않는 경우의 히스토그램을 보여주고 있다. <그림 9-(b)>는 x축 상에서 어느 특정한 값에 대한 횟수가 다른 값들에 대한 횟수보다 더 많이 나타난다. 이는 입력음악과 동일한 등록음악에 대해서는 등록음악의 해쉬 키와 입력음악의 해쉬 키에 대한 시간오프셋의 차이가 일정하

기 때문이다. 반면 <그림 10-(b)>는 x축 상의 특정 값의 출현 횟수가 집중되어 있지 않다. 이는 입력받은 쿼리와 일치하지 않는 곡에 대해서는 등록음악에서의 해쉬 키와 입력음악의 해쉬 키에 대한 시간오프셋의 차이가 일정치 않기 때문이다. 입력음악으로부터 추출한 모든 해쉬 키를 이용하여 생성된 모든 히스토그램을 탐색하여 어느 특정 (등록음악에서 추출한 해쉬 키의 시간 오프셋 - 입력음악에서 추출한 해쉬 키의 시간 오프셋)에서 횟수가 가장 높은 음악을 검색 결과로 반환한다.

4. 결 론

본 논문에서는 상용화된 음악 검색 서비스에서 활용되는 음악 식별 기술 중에서 fingerprint를 이용한 방식에 대해 대표적으로 밴드 간 에너지 차를 이용한 fingerprint 생성에 기반한 방법과 peak point들 간의 해쉬 키 생성에 기반한 방법을 중심으로 지금까지의 연구 동향을 정리해보았다. 밴드 간 에너지 차를 이용한 fingerprint 생성에 기반한 방법은 모든 주파수 밴드의 에너지 값을 이용하여 인접한 밴드 에너지들의 차를 통해 fingerprint를 추출하고, 추출한 fingerprint를 인덱스로 사용하여 LUT를 통해 검색이 이루어진다. Peak point들 간의 해쉬 키 생성에 기반한 방법은, 특정 범위 내에서 가장 높은 에너지를 가지는 peak point들을 추출하여 이들을 포함한 해쉬 키를 이용해 각 해쉬 키의 시간 offset을 통해서 검색이 이루어진다.

위의 두 가지 방법의 성능을 1만곡을 대상으로 비교해 보았다. 음악 데이터베이스는 벅스(www.bugs.co.kr)에서 구입한 가요, POP, 광고음악, OST, 클래식, 재즈로 구성된 10,000 개의 음악 mp3 파일을 16,000 Hz로 샘플링하여 구성하였다.

이들 1 만곡 중 임의로 선정한 200곡에 대하여 시작 지점으로부터 곡의 길이의 25%되는 지점에서 3초 길이의 음악 클립을 추출하였다. 커피 전문점에서 iPhone을 이용하여 배경 잡음을 수집하고, 위의 200개 클립에 대하여 SNR15에 맞춰 배경 잡음을 원음에 첨가하여 실험 데이터를 생성하였다. 실험은 두 가지 방법으로 구현한 시스템들에 200개의 테스트 데이터를 인식하여 진행하였다. 밴드 간 에너지 차를 이용한 fingerprint 생성에 기반한 방법의 인식률은 97.5%로 측정되었으며, 이 때 메모리는 약 6GB를 사용하였다. Peak point들 간의 해쉬 키 생성에 기반한 방법으로 구현된 시스템은 인식률은 100%로 측정되었고, 메모리는 약 4GB를 사용하였다.

참고문헌

[1] Gracenote Inc. (1999). CDDB, World's Largest Online CD Music Database, Debuts New Web Site, http://www.gracenote.com/company_info/press/1999/199904270.

[2] Fong, C. (2008). Tracking the Internet of Music, http://www.cnn.com/2008/TECH/11/16/digitalbiz.musicid/index.html?iref=allsearch.

[3] Seo, G. (2002). U.S. DEMO Mobile Conference: Breaking IT Recession Mobile Communication Technology Draws Attention, http://www.etnews.com/news/detail.html?id=200209290035.
(서기선. (2002). 美 데모모바일 전시회 IT불황 날려버릴 이통 기술 관심 집중, http://www.etnews.com/news/detail.html?id=200209290035.)

[4] Shazam Entertainment Ltd. (2008). Shazam Announces Application for iPhone, http://www.shazam.com/music/web/pressrelease.html?nid=NEWS20080710151044.

[5] Gracenote Inc. (2008). Gracenote and MetroLyrics Extend Authorized Lyrics Service to Mobile Devices and iPhone Applications, http://www.gracenote.com/company_info/pres/11/12/2008.

[6] SoundHound Inc. (2010). SoundHound Launches Free Music Search and Discovery App for iPhone, iPod touch & iPad. http://www.soundhound.com/index.php?action=s.press_release&pr=14.

[7] Lee, H. (2011). What is this Song? Daum Launches Mobile Music Recognition. http://bntnews.hankyung.com/apps/news?popup=0&&nid=05&c1=05&c2=05&c3=00&nkey=201101082031223&mode=sub_view.
(이현아. (2011). 이 노래 뭘까? 다음, 모바일 음악검색 서비스 오픈. http://bntnews.hankyung.com/apps/news?popup=0&nid=05&c2=05&c3=00&nkey=201101082031223&mode=sub_view.)

[8] Lee, Y. (2011). Building Music Feature Database for People Who Forgot the Music Title, Search Within 10 Seconds. http://biz.chosun.com/site/data/html_dir/2011/07/15/2011071501138.html.
(이윤식. (2011). 노래 제목 깜빡한 분을 위해 곡의 특징 DB화, 10초면 찾아내. http://biz.chosun.com/site/data/html_dir/2011/07/15/2011071501138.html.)

[9] Casey, M., Veltkamp, R., Goto, M., Leman, M., Rhodes, C. and Slaney, M. (2008). Content-Based Music Information Retrieval: Current Directions and Future Challenges, *Proceedings of the IEEE*, Vol. 96, No. 4, 668-696.

[10] Gracenote Inc. (2005). Gracenote News, http://www.gracenote.com/company_info/press/2005/2005083000.

[11] Haitsma, J., Kalker, A., Baggen, C. and Oostveen, J. (2002). Generating and Matching Hashes of Multimedia Content, United States Patent Application, US 2002/0178410 A1.

[12] Haitsma, J. & Kalker. T. (2002). A Highly Robust Audio

Fingerprinting System, *Proceedings of International Symposium on Music Information Retrieval (ISMIR)*, 144-148.

[13] Haitsma, J., Kalker, T. and Oostveen, J. (2001). Robust Audio Hashing for Content Identification, *Proceedings of Content-Based Multimedia Indexing*, 19-21.

[14] Wang, A. (2006). The Shazam Music Recognition Service, *Com. ACM*, Vol. 49, No. 8, 44-48.

[15] Wang, A. (2003). An Industrial Strength Audio Search Algorithm, *Proceedings of International Symposium on Music Information Retrieval (ISMIR)*, 7-13.

[16] Wang, A. & Smith, J. (2002). *Method for Search in an Audio Database*, WIPO publication, WO 02/11123A.

- **이동현(Donghyun Lee), 제1저자**
  서강대학교 컴퓨터공학과
  서울특별시 마포구 신수동 1번지
  Email: redizard@sogang.ac.kr
  관심분야: Speech Recognition, Music Information Retrieval
  현재 컴퓨터공학과 학사과정

- **임민규(Minkyu Lim), 제2저자**
  서강대학교 컴퓨터공학과
  서울특별시 마포구 신수동 1번지
  Email: lmkhi@sogang.ac.kr
  관심분야: Speech Recognition, Music Information Retrieval
  현재 컴퓨터공학과 대학원 박사과정

- **김지환(Ji-Hwan Kim), 교신저자**
  서강대학교 컴퓨터공학과 부교수
  서울특별시 마포구 신수동 1번지
  Email: kimjihwan@sogang.ac.kr
  관심분야: Spoken Multimedia Content Search, Speech Recognition using Cloud Computing and Dialogue Understanding