졸업작품 중간보고서

Query by Humming



팀원

현경우 200111948

이상일 200212025

안세원 200511336

담당조교

남현우 조교

담당교수

박능수 교수님

제출일

2008년 6월 24일

1. 개요

QBH 소개

2. 팀원소개

현경우

이상일

안세원

3. 관련 연구 조사 (기존 방법)

Query by Example

Audio Fingerprinting

정확도 98%

2004년 KTF 서치뮤직, 2007년 Verizon SongID

단점: 실제 소리여야 함. 목소리로 검색 안됨.

Query by Humming

wav2midi

energy

pitch

voice recognition

UDR + SLR

정확도 떨어짐

2007년 벅스뮤직 그만둠

Fingerprinting Approach

음변화율, 박자 MAX, MIN값 4개씩 추출, 비교

wav to midi

energy

picth

voice recognition

4. 작업내용

client:

flash로 작업하다가, wav로 바뀌지 않아서 실패.

* java applet으로 바꿈. JMF 이용.

server:

aubio 사용

as-is 기존 프로그램

to-be 새 프로그램

기존 105곡, 음원추가 네덜란드 대학 FTP에서 155곡 -> 그러나 피아노곡이 많아서 어려움

서버 새로 추가한 부분 (소스코드)

클라이언트 (소스코드)

기존 프로그램에 똑 같은 wav를 넣었을 경우 결과

5. 향후계획

UDR 알고리듬 -> fingerprinting

미디파일 확보 musedata.org

대량 입출력을 위한 통계/평가 도구. 출력은 csv 파일로.

시간에 따른 음 변화율, 토막내기 알고리듬

GUI

6. 일정

3월 6일 팀 결성

3월 7일 신청서 제출, 교수님 면담

* 리서치. 논문 연구
* flash 기술 조사

4월 11일 중간보고

* 서버 통합
* 클라이언트 자바 애플릿
* 미디파일 추출, 논문 연구

5월 7일 중간보고

* 알고리듬 확인 -> 결과 나쁨
* fingerprinting 접근 연구

6월 24일 중간보고서

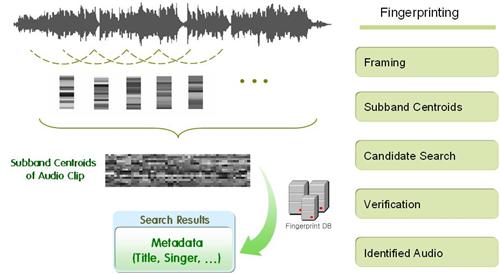
향후계획

1. 기존 검색방법

음악을 검색하는 방법에는 크게 4가지가 있다.

|  |  |
| --- | --- |
| Query by Example (QbE) | 실제 음악으로 질의 (song-to-song) |
| Query by Humming (QbH) | 사람의 허밍으로 질의 (voice-to-song) |
| Query by Music Note (QbMN) | 악보로 질의 (note-to-song) |
| Query by Contour (QbC) | 음의 윤곽선으로 질의 (graph-to-song) |

이 중에서 현재까지 가장 높은 신뢰도를 보이는 방법은 Query by Example이다. 특히 Audio Fingerprinting을 사용할 경우 97%의 매우 높은 정확도를 보인다.[[1]](#footnote-2)



KAIST, Multimedia Processing Lab에서 발췌[[2]](#footnote-3)

음원에서 특징적인 부분들(fingerprints)를 뽑아낸다. (보통 3초당 256개의 지문, 노래가 3분이라면 약 1만 5천개 정도의 지문이 나온다.) 그리고 입력받은 음악에서도 fingerprints를 뽑아내서, 이들끼리 비교하여 가장 유사한 곡을 찾는다.

|  |  |
| --- | --- |
| 장점 | 음악의 어떤 부분이던 15~20초의 질의로 거의 완벽하게 찾아낼 수 있다.  노이즈, 왜곡에 강하다. |
| 단점 | 실제 음악으로만 검색이 가능하다. (적용범위 한정) |

응용의 예: 길을 걷는데 어느 의류 매장에서 마음에 드는 음악을 틀어놓고 있었다. 가게 점원에게 물어봤지만 무슨 노랜지 몰랐다. 이럴 때 재빨리 핸드폰으로 음악을 들려줘서 무슨 음악인지 찾는다.

적용사례

|  |  |
| --- | --- |
| 모바일 질의 | 2004년 한국 KTF의 서치뮤직  2007년 미국 Verizon의 SongID  KTF 서치뮤직에서는 서태지의 미공개곡을 광고로 틀고, 이를 휴대폰으로 들려줘서 구매하도록 유도했음.[[3]](#footnote-4) |
| 인터넷 방송 질의 | 2007년, 인터넷 라디오 방송 커뮤니티 Last.fm에서 모르는 음악에 자동으로 meta data를 붙여주는 서비스[[4]](#footnote-5) |

결론: Query by Example은 매우 정확하게 원하는 음악을 찾아주지만, 적용 범위가 좁다. 그러므로 다양한 분야에 적용하기 위해서는 Query by Humming 등의 다른 접근방법이 필요하다.

Query by Humming: 사람의 허밍으로 찾아주는 방법이다.

동의어: humming search, voice search

응용의 예: 노래방에 갔는데 노래 제목을 모르겠다. 멜로디는 확실하게 기억이 나는데 제목, 가수, 일부 가사도 떠오르지 않는다. 이럴 때 마이크로 노래를 “라라라~” 흥얼거리면 원하는 음악을 찾아준다.

문제점: 정확도가 매우 떨어진다. 원하는 음악이 상위 5위 안에 있을 확률이, 현재까지 가장 좋은 방법으로도 60~70% 이하이다.

무엇이 문제인가?

프로세싱 절차

1. abstraction: 음성에서 의미있는 부분만을 추출한다. (data to meta-data)

2. query: 음악 DB에 질의해서 찾는다.

검색이 안 되는 원인

1. 사람이 노래를 잘못 불렀다. (사람의 왜곡)

2. 음성 녹음 과정에서 잡음 등의 왜곡이 있었다. (기계의 왜곡)

3. 음성을 미디로 잘못 변환했다. (wave-to-midi)

4. 음원 DB가 부실했다. (music DB)

5. 검색하는 알고리즘이 좋지 않았다. (searching way)

이 원인들이 영향을 미치는 범위

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | human distort | machine distort | wave-to-midi | music DB | searching way |
| abstraction | O | O | O | X | X |
| query | O | X | X | O | O |

O: affected

X: not affected

이 중에 이번 졸업작품에서 집중할 부분은 어느 부분인가? 이를 찾기 위해서는

1. 먼저 기존의 연구들이 얼마나 진척되었는가를 조사하고

2. 연구가 이미 잘 된 부분은 제외하고, 미비한 부분을 골라서 집중한다.

기존의 연구

1. wave-to-midi abstraction

3 metrics

energy

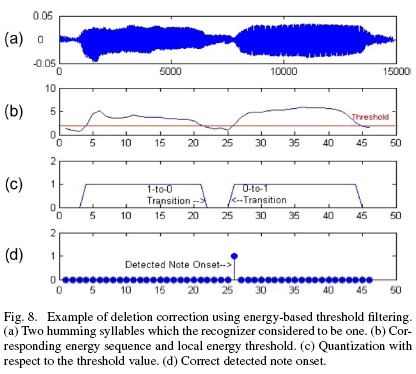
음량이 급격하게 변화하는 부분을 잘라냄

pitch

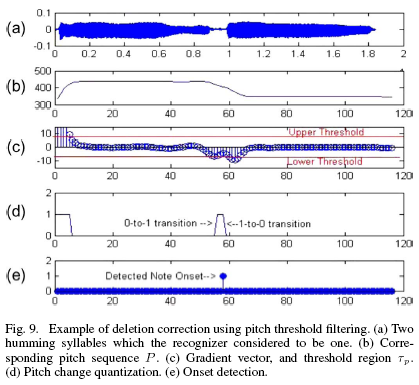
음정이 급격하게 변화하는 부분을 잘라냄

voice recognition

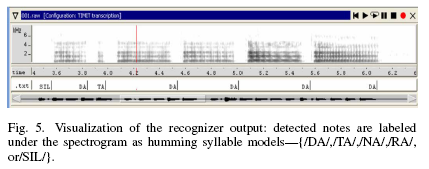
“라라라~”하고 부를 때, “라”라고 인식되는 부분을 잘라냄



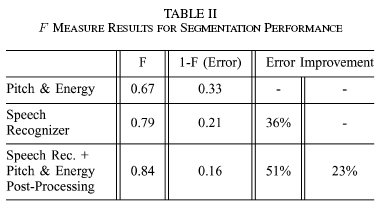
energy-based threshold filtering



pitch threshold filtering

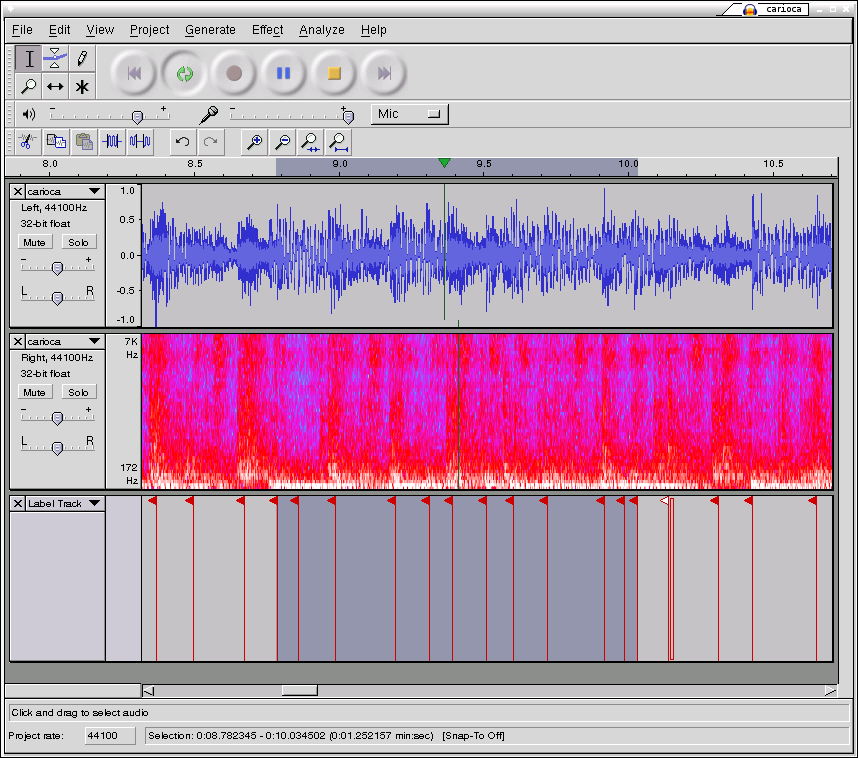


voice recognition

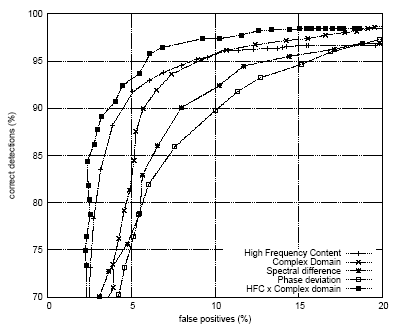


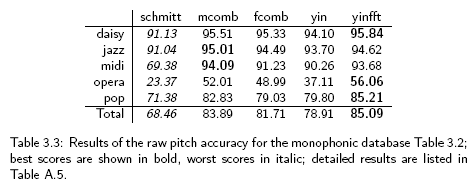
세가지 방법을 합칠 경우 84%까지 정확도가 높아짐

Aubio의 경우 이 중에서 energy, pitch를 이용한다.



Aubio 실행결과를 Audacity로 본 모습





Aubio의 경우 단음계에서 최고 85%까지 정확도를 보이므로, 이 부분은 충분히 연구가 진행된 것으로 보아 이번 졸업작품에는 포함하지 않는다.

2. searching ways

음 표현법

1. 음 변화를 상대적인 크기로 나타내는 방법

절대치로 나타낼 경우, 사람이 똑 같은 노래를 낮춰 부르거나 높여 부르면, 또는 빠르게 부르거나 느리게 부르면 찾을 수 없다. 그러므로 상대치로 찾는다.

음정이 1 올라갈 경우: +1

음정이 2 올라갈 경우: +2

음정이 1 내려갈 경우: -1

음정이 2 내려갈 경우: -2

음정이 되풀이될 경우: 0

(단위: 반음semitone)

음 길이가 0.5 길어질 경우: +0.5

음 길이가 0.5 짧아질 경우: -0.5

음 길이가 되풀이될 경우: 0

(단위: 박tick)

2. UDR/SLR

사람이 정확하게 부르기보다는 다소 오류를 포함하여 틀리게 부를 수 있는데, 이런 오류를 허용하기 위해서는 다소 차이가 나는 값도 받아들여야 한다. 이런 오류 허용을 가장 최대로 한 방법이 UDR/SLR이다.

음정이 올라갈 경우: Up

음정이 내려갈 경우: Down

음정이 되풀이될 경우: Repeat

음 길이가 짧아질 경우: Shorter

음 길이가 길어질 경우: Longer

음 길이가 같을 경우: Repeat

예를 들어 “Happy Birthday”는 다음과 같은 UDR/SLR String으로 나타낼 수 있다.

RUDUDDRUDUDDRUDDDDURDDUD

SLRRRRSLRRRRSLRRRLSSLRRL

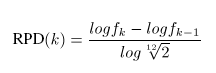
문제점: UDR만 적용할때보다 UDR+SLR 적용시 확률이 떨어진다.

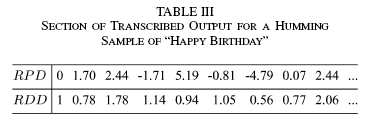
왜? 1. 사람은 박자보다 음정에 민감하기 때문에 (가설)

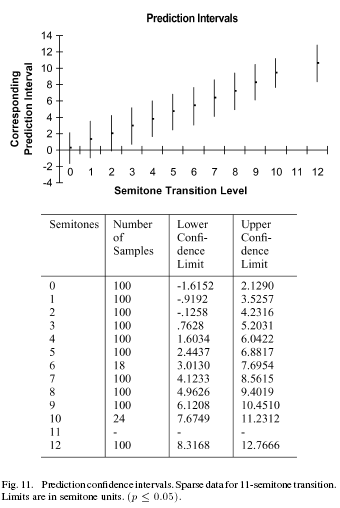
2. 대부분의 음악이 비슷한 박자를 가지고 있기 때문에 (가설)

3. 로그 단위 logarithmic scale

인간은 변화율을 로그 단위로 인지하는 경향이 있으므로, 변화율에 로그를 취해서 오류를 허용하는 방법이 좋을 것이다.







실험자들에게 노래를 들려주고 똑같이 부르게 할 경우, 얼마나 음정을 다르게 부르는지 통계를 낸 것. 대체로 -2~+2 정도의 차이를 보인다.

그러므로 단순히 UDR/SLR로 크게 자르는 것보다, 통계적인 범위 내에서만 오류를 허용하는 방식이 좀 더 정확할 것이다.

계산법

1. edit distance

2개의 String을 비교하는 단순한 방법이다.

UDR을 이용한 방법을 예로 들면

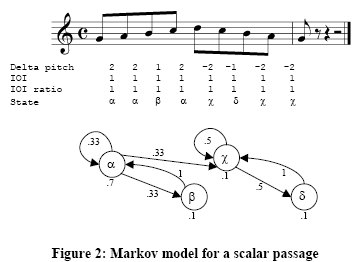
1. 음원 DB에도 많은 UDR String들이 들어있다.

2. 사람의 음성을 입력받아서 UDR String “UDDRRRDD”로 변환

3. 서로 비교하여 비슷한 순서로 정렬하여 결과를 돌려준다.

2. HMM (Hidden Markov Model)

통계적인 방법으로 검색 정확도를 높이는 방법이다.



위 노래에서는 총 4개의 상태(state)가 나온다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 상태 알파 | 상태 베타 | 상태 감마 | 상태 델타 |
| 음정 | +2 | +1 | -2 | -1 |
| 음 길이 | 1 | 1 | 1 | 1 |

이 4개의 상태가 어떤 확률로 서로 변하는지를 통계적인 정보로 내 보면, 예를 들어

상태 알파에서

상태 알파로 갈 확률: 33%

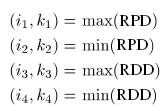
상태 베타로 갈 확률: 33%

상태 감마로 갈 확률: 33%

이런 확률을 토대로 비슷한 노래를 찾으면서, 수천~수십만개의 질의를 다시 통계를 내서 이 확률값을 수정하면 정확도를 높일 수 있을 것이다. 이 방법을 실제로 적용한 사이트로 midomi.com 이 있다.

3. fingerprinting

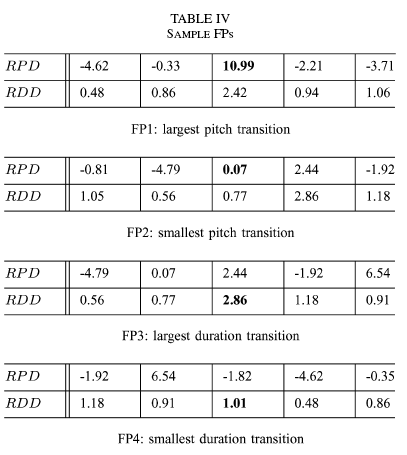
전체 노래를 다 비교하는 것이 아니라, 특징적인 부분만 추출해서 비교하는 방법.



음정의 변화(RPD)가 가장 큰 곳과 가장 작은 곳,

음 길이의 변화(RDD)가 가장 큰 곳과 가장 작은 곳.

이렇게 4곳을 선정한다.



선정한 네 지점에서 반경(Radius) 2 (+2 ~ -2)만큼 가져오면, 길이가 5인 fingerprints 4개가 나온다. (반경 값은 실험으로 최적의 값을 선정)

마찬가지로 입력받은 음성에서도 fingerprints를 추출해서, 이들을 서로 얼마나 차이가 나는지 계산한다.

결론: 지금까지 알아본 방법 중에서, 이번 졸업작품에서는 다음 조합을 사용한다.

음 표현법: 로그 단위

계산법: fingerprint

작업내용: 똑 같은 음을 입력할 경우

완전히 같은 멜로디를 집어넣을 경우 어떤 결과가 나올까? “과수원길” 앞에서부터 4마디를 집어넣을 경우, 105개 노래 중 45위에 나왔다. 아기공룡 둘리도 똑 같은 방법으로 했더니 검색결과에 아예 나오지 않았다.

다른 미디파일도 검색해보았다.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **미디 파일** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
| 가제트 | 1 | 9 | 20 | 25 | 5 | 0 | 0 |  |
| 개구리왕눈이 | 11 | 39 | 0 | 0 |  |  |  |  |
| 고기잡이 | 0 | 0 | 0 | 10 | 0 | 38 | 2 | 2 |
| 고드름 | 0 | 8 | 10 | 15 | 1 | 3 | 2 |  |
| 과수원길 | 0 | 0 | 0 | 17 | 1 | 1 | 46 | 43 |
| 꼬마자동차붕붕 | 26 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 따오기 | 30 | 29 | 4 | 6 | 11 | 0 | 1 | 17 |
| 마징가Z | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

앞에서부터 2마디, 4마디, 6마디… 검색한 결과.

0: 검색 결과에 나오지 않음

특히 개구리 왕눈이의 경우, 어떤 부분을 입력하느냐에 따라 결과가 달라졌다.

|  |  |
| --- | --- |
| “개구리소년 밤바밤” | 11위 |
| “개구리소년 밤바밤 개구리소년 밤바밤” | 39위 |
| “삘릴리 개굴개굴 삘릴릴리” | 없음 |
| “삘릴리 개굴개굴 삘릴릴리 삘릴리 개굴개굴 삘릴릴리” | 없음 |

여기서, 맨 마지막 2개를 보면, 단순히 같은 멜로디를 한번 반복, 2번 반복하느냐 차이인데도 검색결과가 크게 다르다.

|  |  |
| --- | --- |
| “삘릴리~” 1번 | “삘릴리~” 2번 |
| [./강산애\_넌할수있어.qbh] = 2 | [./홍경민\_흔들린우정.qbh] = 10 |
| [./김광석\_일어나.qbh] = 2 | [./\_Midi103.qbh] = 12 |
| [./루돌프사슴코.qbh] = 2 | [./김광석\_일어나.qbh] = 12 |
| [./은하철도999.qbh] = 2 | [./21.qbh] = 12 |
| [./젓가락행진곡.qbh] = 2 | [./피구왕통키.qbh] = 13 |
| [./47.qbh] = 3 | [./나디아\_오프닝.qbh] = 13 |
| [./ookinahurudokei.qbh] = 3 | [./마법의성.qbh] = 14 |
| [./0193.qbh] = 3 | [./꼬마자동차붕붕.qbh] = 14 |
| [./김광석\_사랑했지만.qbh] = 3 | [./핑계.qbh] = 14 |
| [./37.qbh] = 3 | [./\_Midi141.qbh] = 14 |
| [./노사연\_만남.qbh] = 3 | [./13.qbh] = 16 |
| [./38.qbh] = 3 | [./젓가락행진곡.qbh] = 16 |
| [./소양강처녀.qbh] = 3 | [./클래식\_마법의성.qbh] = 16 |
| [./아기공룡둘리.qbh] = 3 | [./마징가Z.qbh] = 17 |
| [./어린이행진곡.qbh] = 3 | [./12.qbh] = 18 |
| [./오빠생각.qbh] = 3 | [./47.qbh] = 18 |
| [./41.qbh] = 3 | [./편승엽\_찬찬찬.qbh] = 18 |
| [./46.qbh] = 3 | [./남자는여자를귀찮게해.qbh] = 18 |
| [./클래식\_마법의성.qbh] = 3 | [./makkanaaki.qbh] = 18 |
| [./편승엽\_찬찬찬.qbh] = 3 | [./한영애\_누구없소.qbh] = 18 |
| [./핑계.qbh] = 3 | [./8.qbh] = 18 |
| [./10.qbh] = 4 | [./ookinahurudokei.qbh] = 19 |
| [./돌아와요부산항에.qbh] = 4 | [./꿍따리사바라.qbh] = 19 |
| [./\_Midi103.qbh] = 4 | [./우리모두다함께.qbh] = 19 |
| [./\_Midi141.qbh] = 4 | [./강산애\_넌할수있어.qbh] = 19 |
| [./16.qbh] = 4 | [./chiisaiakimituketa.qbh] = 19 |
| [./가제트.qbh] = 4 | [./아리랑.qbh] = 20 |
| [./40.qbh] = 4 | [./46.qbh] = 20 |
| [./21.qbh] = 4 | [./태권v.qbh] = 20 |
| [./24.qbh] = 4 | [./은하철도999.qbh] = 20 |
| [./꼬마자동차붕붕.qbh] = 4 | [./아기공룡둘리.qbh] = 21 |
| [./태권v.qbh] = 4 | [./22.qbh] = 21 |
| [./남행열차.qbh] = 4 | [./34.qbh] = 21 |
| [./피구왕통키.qbh] = 4 | [./7.qbh] = 21 |
| [./노란샤쓰의사나이.qbh] = 4 | [./42.qbh] = 21 |
| [./한영애\_누구없소.qbh] = 4 | [./44.qbh] = 21 |
| [./홍경민\_흔들린우정.qbh] = 4 | [./남행열차.qbh] = 21 |
| [./권진원\_살다보면.qbh] = 5 | [./노사연\_만남.qbh] = 21 |
| [./스승의 은혜.qbh] = 5 | [./돌아와요부산항에.qbh] = 21 |
| [./8.qbh] = 5 | [./군밤타령.qbh] = 21 |
| [./아파트.qbh] = 5 | [./김경호\_겨울비.qbh] = 21 |
| [./양희은\_아침이슬.qbh] = 5 | [./소양강처녀.qbh] = 21 |
| [./33.qbh] = 5 | [./33.qbh] = 22 |
| [./36.qbh] = 5 | [./인디안인형처럼.qbh] = 22 |
| [./우리모두다함께.qbh] = 5 | [./24.qbh] = 22 |
| [./날아라슈퍼보드.qbh] = 5 | [./37.qbh] = 22 |
| [./인디안인형처럼.qbh] = 5 | [./38.qbh] = 22 |
| [./남자는여자를귀찮게해.qbh] = 5 | [./siawasenaratewotatakou.qbh] = 23 |
| [./13.qbh] = 5 | [./날아라슈퍼보드.qbh] = 23 |
| [./널사랑하겠어.qbh] = 5 | [./권진원\_살다보면.qbh] = 23 |

결론  
1. 곡의 어떤 부분을 잘라서 넣느냐에 따라 정확도가 크게 달라진다.  
2. 입력이 길어질수록 순위가 떨어진다. (여러 레파토리가 섞이므로)  
3. 같은 레파토리라도 한번 반복하느냐, 2번 반복하느냐에도 다르게 반응한다.

그러므로 곡의 특징적인 부분만 짧게 추출하여 비교하는 것이 좋을 것이다.

그렇다면 특징적인 부분을 어떻게 추출할 것인가?

1. 사람의 판단으로 잘라낸다

2. 자동으로 잘라낸다 (fingerprinting approach)

1. <http://www.esprockets.com/papers/Baluja_Covell_ICASSP2007_1888.pdf> [↑](#footnote-ref-2)
2. <http://fft1.kaist.ac.kr/ra_fingerprinting.html> [↑](#footnote-ref-3)
3. <http://www.muncle.com/media/mobile_news_view.jsp?seq=1396> [↑](#footnote-ref-4)
4. <http://blog.last.fm/2007/08/29/audio-fingerprinting-for-clean-metadata> [↑](#footnote-ref-5)