허밍 대수를 이용한 허밍 질의처리 시스템
(A Query by Humming System Using Humming Algebra)

신 제 용 † 한 욱 신 †† 이 종 학 †††
(Je-Yong Shin) (Wook-Shin Han) (Jonghak Lee)

요 약 허밍 질의는 사용자가 가사를 모를 때, 유용하고 직관적으로 사용할 수 있는 질의 방법이다. 허밍 질의 시스템은 사용자 멜로디를 입력으로 받고, 이 멜로디를 음악 데이터베이스의 모든 멜로디와 비교하며, 가장 유사한 k개의 멜로디를 반환한다. 본 논문에서는 허밍 질의 시스템을 위한 허밍 대수를 제안하고, 허밍 대수를 이용하여 실제 허밍 질의처리 시스템인 HummingBase를 설계하고 구현하였다. 기존 유사 검색 방법들을 분석함으로써 10개의 기본 연산자로 구성된 대수를 유도하였다. 제안한 허밍 대수는 허밍 질의 시스템이 확장이 가능하고 모듈화가 되게끔 구현하는데 사용될 수 있다. 본 논문에서는 두 가지 사례 연구를 통해, 제안한 허밍 대수를 이용하면 기존의 허밍 질의처리 시스템을 쉽고 편리하게 표현할 수 있음을 보인다.

키워드 : 허밍 질의, 허밍 대수, 시계열 데이터

Abstract Query by humming is an effective and intuitive querying mechanism when a user wants to find a song without knowing lyrics. The query by humming system takes a user-hummed melody as input, compares it with melodies in a music database, and returns top-k similar melodies to the input. In this paper, we propose a novel algebra for query by humming, and design and implement a real query by humming system called HummingBase by exploiting the algebra. By analyzing existing similarity search techniques, we derive 10 core operators for the algebra. By using the well-defined algebra, we can easily implement such a system in a extensible and modular way. With two case studies, we show that the proposed algebra can easily represent the query processing processes of existing query-by-humming systems.

Key words : Query by humming, Humming algebra, Time-series data

1. 서 론

멀티미디어 데이터에 대한 검색이 점점 늘어나고 중요해 지고 있다. 음악이나 동영상과 같은 멀티미디어 데이터들이 전 세계적으로 급속하게 증가하고 있고, 이에 따라 방대한 양의 멀티미디어 데이터의 활용이 많아지고 있기 때문이다. 그러므로 사용자가 원하는 데이터를 정확하고 빠르게 검색하는 것이 중요하다. 멀티미디어 데이터 중, 음악 데이터에 대한 검색은 주로 가수 이름, 노래 제목, 가사 등의 메타 데이터를 사용하여 이루어진다.

메타 데이터 이외에 음악을 검색할 수 있는 방법에는 사용자가 노래의 멜로디를 부르는 허밍(humming)을 이용한 방법[1-4]이 있다. 허밍을 이용하여 노래를 검색하는 시스템을 허밍 질의처리 시스템(Query by Humming System)이라 부른다. 이것은 주로 미디 파일이나 디지털 음악 악보와 같은 것에서 멜로디를 추출하여 데이터베이스를 구축하고, 허밍으로부터 데이터를 추출하여 구축해 놓은 데이터베이스에 질의하여 검색하는 방식을 취하고 있다.

일반적으로 사용자는 자신이 아는 노래를 최대한 잘 부르려고 노력한다. 하지만 그러한 허밍조차 악보에 적

· 이 논문 또는 저서는 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2007-521-D00399)

† 정 회 원 : 경북대학교 컴퓨터공학과
jesusofskys@nate.com
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
wshan@knu.ac.kr
(Corresponding author임)
††† 종신회원 : 대구카톨릭대학교 컴퓨터공학과 교수
jhlee11@cu.ac.kr

논문접수 : 2008년 12월 30일
심사완료 : 2009년 6월 29일

혀있는 것과 완벽하게 일치하지 않는다. 그러므로 사용자의 허밍과 유사한 노래를 검색해 주는 허밍 질의처리 시스템에 대해 연구가 진행되고 있다.

노래 멜로디와 허밍은 시계열 데이터(time-series data)로 표현이 가능하다. 왜냐하면, 두 데이터 모두 일정한 시간 간격으로 특정 값이 연속적으로 나타나는 시계열 데이터의 특성을 가지고 있기 때문이다. 그러므로 허밍과 유사한 노래를 찾는 문제는 어떤 질의 시퀀스와 유사한 데이터 시퀀스를 찾는 문제로 치환하여 해결할 수 있다.

본 논문에서는 허밍에 의해 유사 멜로디를 검색하는 허밍베이스(HummingBase)의 구현에 대해 설명한다. 이를 위해 허밍 검색과 관련된 정형화된 대수를 제안하고, 이 대수들을 활용하여 허밍 질의가 처리됨을 보인다. 허밍 대수를 활용함으로써 여러 가지 표현식들을 결합하여 보다 복잡하고 흥미로운 질의를 표현할 수도 있다. 사용자에게는 선언적으로 정의된 질의 언어에 기반을 둔 인터페이스를 제공할 수 있다.

허밍베이스에서는 미디 파일과 웨이브 파일로부터 시계열 데이터를 추출한다. 음악 데이터에 대한 시계열 데이터는 음조(pitch), 음의 길이(duration) 쌍의 리스트로 표현한다. 추출된 데이터는 윈도우를 생성하는 기법을 사용하여 하나 이상의 윈도우로 나뉜다. 미디 파일로부터 추출된 시계열 데이터와 윈도우는 데이터베이스와 인덱스를 구축하는데 사용하고, 허밍으로부터 추출된 시계열 데이터와 윈도우는 검색에 사용하여, 허밍과 유사한 결과를 찾는다.

본 논문의 공헌은 다음과 같이 요약된다. 첫째, 허밍 질의를 표현하기 위한 허밍 대수를 제안하였다. 둘째, 제안한 허밍 대수를 처리할 수 있는 허밍 질의처리 시스템을 구현하였다. 셋째, 사례 연구를 통해 본 논문에서 제안하는 대수가 기존 허밍 질의처리 시스템에도 적용하여 표현할 수 있음을 보였다.

본 논문의 구성은 다음과 같다. 제2절에서는 관련 연구에 대해 설명하고, 제3절에서는 본 논문에서 제안하는 허밍 질의처리를 지원하는 허밍 대수에 대해 논의한다. 제4절에서는 논문에서 제안하는 허밍 대수를 처리하는 허밍 질의처리 시스템 허밍베이스에 대해 기술한다. 제5절에서는 사례 연구로서, 기존 허밍 질의처리 시스템을 본 논문에서 제안하는 허밍 대수를 이용하여 표현한다. 마지막으로 제6절에서 결론을 맺는다.

2. 관련 연구

시계열 데이터에 대해 유사 검색을 하는 방법에는 전체 매칭(whole matching)[5]과 서브시퀀스 매칭(subsequence matching)[6,7]이 있다. 전체 매칭은 N개의 데이터 시퀀스 S_1, ..., S_N과 질의 시퀀스 Q와 한계치 ε이 주어졌을 때, Q와 S_i의 유사도가 ε보다 작거나 같은 데이터 시퀀스를 찾는 방법이다. 유사도란 두 시퀀스의 차이 정도를 의미한다. 서브시퀀스 매칭 방법은 N개의 데이터 시퀀스 S_1, ..., S_N과 질의 시퀀스 Q와 한계치 ε이 주어졌을 때, Q와 S_i의 서브시퀀스 사이의 유사도가 ε보다 작거나 같은 데이터 시퀀스의 서브시퀀스를 찾는 방법이다. 다음 절에서부터는 허밍 질의처리 시스템에 유사 검색을 이용한 연구들에 대해 설명한다.

2.1 Keogh의 방법[8]

이 논문은 동적시간워핑(DTW: Dynamic Time Warping)을 이용하여, 데이터 시퀀스와 질의 시퀀스를 비교하기 위한 인덱싱 방법과 질의 방법을 설명하고 있다. 인덱싱 과정과 질의 과정은 다음과 같다.

길이가 N인 데이터 시퀀스를 효율적으로 인덱싱하기 위하여, 일정한 구간의 평균값을 이용하는 구분적 집적근사(PAA: Piecewise aggregation approximation)[9]라는 변환을 이용하여 길이가 N인 데이터 시퀀스를 n(n ≪N)길이의 저차원 데이터로 변화시킨 후, 저차원 데이타들을 다차원 인덱스에 저장하였다.

반면 질의의 경우, 길이가 N인 질의 시퀀스에 엔벨로프(envelope)라 불리는 길이가 N인 두 시퀀스를 생성한다. 엔벨로프는 일정 구간의 최소, 최대값을 이용하는 변형된 구분적 집적근사(본 논문에서는 Keogh_PAA 부름)를 이용하여 n(n≪N)길이의 저차원 데이터로 변환된다. 이 저차원 데이터와 k-최근접 질의(k-Nearest Neighbor search)[10,11]를 이용하여 인덱스를 검색하였다.

2.2 Zhu et al's 시스템[4]

이 시스템은 허밍 질의처리 시스템에 [8]의 방법을 도입하여 전체 매칭 방법으로 허밍을 검색하였다. 미디 파일로부터 멜로디 정보를 추출한 후, 음과 음 길이의 쌍의 시퀀스로 표현된 멜로디의 서브멜로디들로 데이터베이스를 구축하였다. 질의 데이터는 사용자 허밍에서 데이터를 추출하였다. 이것을 이용하여 데이터베이스에 저장되어 있는 미디 데이터와 유사한 멜로디를 검색하였다.

데이터 시퀀스인 미디 데이터를 이용하여 인덱스를 구축하는 방법은 [8]의 방법과 유사하였다. N길이의 데이터 시퀀스를 n길이의 데이터로 저차원 변환한 후 인덱스에 저장하였다. 다만, 음악 데이터의 특성으로 인해 생기는 빠르기와 음정의 차이 등을 고려하여 데이터 값을 변형시킨 후 저차원 변환을 수행하였다.

질의 시퀀스에 대해서는 N길이의 시퀀스에 대한 엔벨로프를 생성하고,생성된 엔벨로프의 평균값을 이용(본 논문에서는 Shasha_PAA 부름)하여 n차원의 저차원 데이터를 만들었다. 데이터 시퀀스와 마찬가지로 데이터 값을 변형시킨 후에 저차원 변환을 수행하였다. 이 데이

터를 인덱스에 ε-범위 질의하여 결과가 될 수 있는 후보 집합을 찾은 후, 그 중에서 실제 질의 시퀀스와 가장 유사한 데이터 시퀀스들을 찾았다.

2.3 SoundCompass[2]

이것은 서브시퀀스 매칭 방법을 사용하는 허밍 질의처리 시스템이다. [4]와 마찬가지로 미디와 허밍에서 데이터를 추출하여 데이터베이스를 구축하고 유사검색을 수행하였다.

미디 데이터는 하나의 윈도우에 들어가는 음표 정보의 개수를 동일하게 하기 위해, 슬라이딩 윈도우 방법을 응용하여 서브/서브서브시퀀스라 불리는 윈도우를 생성하였다. 윈도우 안의 데이터에 대해, 음가에 대한 음의 비율(TPV: Tone-in phonetic value rate)이라 불리는 음의 길이와 음의 값을 변화시킨 윈도우(CSTPV: constant slide-length TPV)를 생성하였다. 또한, 한 윈도우의 첫 번째 슬라이딩 크기 안에서 가장 높은 음을 시작점으로 하는 가변 길이의 보조윈도우(VSTPV: variable slide-length TPV)를 추가로 생성하였다. 이렇게 해서 만들어진 두 종류의 윈도우는 각각 인덱스로 구축되었다.

사용자의 허밍 역시 위와 같은 두 종류의 윈도우를 만든 후, 각각 대응되는 인덱스에 질의하였다. [4]와 달리 서브시퀀스 매칭으로 인해 질의 결과가 동일한 윈도우에 대해 하나 이상 나왔을 경우, 보다 적은 유사도를 그 윈도우에 대한 유사도로 결정하였다.

지금까지 살펴본 두 가지의 허밍 질의 시스템은 데이터베이스 구축 및 인덱스 생성 과정과 허밍 검색 과정이 상당히 유사하다. 이것을 통합하여 정리하면 다음과 같다.

데이터베이스와 인덱스 구축 과정을 살펴보면, 우선 미디 파일로부터 특정 데이터 추출한다. 추출된 데이터를 하나 이상의 윈도우로 나눈다. 나눈 윈도우에 대해 빠르기와 음정의 차이, 음의 길이와 음의 값 등을 고려하여 윈도우 안의 정보를 일반화시키는 정규화(normalization) 과정을 거친다. 정규화 과정을 거친 데이터에 대해 저차원 변환을 거친 후, 인덱스를 생성하고 데이터베이스를 구축한다.

검색 과정은 먼저 허밍 데이터에서 특정 데이터를 추출하여 데이터베이스에 질의할 수 있는 형태의 데이터로 변환한다. 데이터베이스 구축 과정과 마찬가지로 하나 이상의 윈도우로 나누고, 정규화 과정을 거친 후 저차원 변환을 수행한다. 인덱스 질의 시, 특정 거리 함수를 사용하여 유사도를 측정한다.

본 논문에서는 음악 데이터에 대한 허밍 질의처리를 지원하는 대수를 제안한다. 허밍 질의처리의 각 과정들은 연산자로 정의되고, 일반화된다. 또한, 허밍 질의처리와 관련된 과정을 수행하는 질의문을 생성한다. 이것을 파싱하고 컴파일하여 플랜을 생성한 후, 플랜을 실행함으로써 질의문을 처리한다. 다음 장에서 허밍 질의처리를 지원하는 대수와 플랜에 대해 자세하게 논의한다.

3. 허밍 대수

본 절에서는 허밍 질의처리를 지원하는 허밍 대수에 대해 논의한다. 모든 질의문은 순서가 있는 연산자들을 실행함으로써 수행된다. 그러므로 다양한 종류의 질의문을 처리하기 위해 연산자들은 확장성을 고려하여 만들었다. 연산자로 만들 수 있는 질의처리 계획인 플랜에 대해서도 설명한다.

3.1 연산자들(Operators)

본 절에서는 허밍 질의처리를 지원하기 위한 연산자들에 대해 설명한다. 허밍 질의처리를 위해 새롭게 정의한 연산자는 표 1과 같다. 각 연산자는 다음 절에서부터 자세하게 설명한다. 본 논문에서 사용하고 있는 기호의 의미는 표 2와 같다.

3.1.1 유사성(Ω)

(Q) Ω_c (DSs) = Ω_c(Q, DSs) ⇒ Bag(DS)

이 연산자는 데이터시퀀스들 DSs가 주어졌을 때, 질의시퀀스 Q와 유사한 데이터시퀀스들을 찾는 연산자이

표 1 연산자 종류와 역할

연산자 종류	연산자의 역할
유사성(Similarity)	유사 검색 지원
윈도우 나눔(Split window)	시계열데이터를 나누어서 윈도우 생성
저차원 변환(Low dimension transform)	저차원 변환 수행
스케일링(Scaling)	정규화 방법 중 스케일링 수행
시프팅(Shifting)	정규화 방법 중 시프팅 수행
웨이브 파일에서 피치 추출(Wave to pitch)	웨이브 파일로부터 시계열데이터 추출
미디 파일에서 피치 추출(Midi to pitch)	미디 파일로부터 시계열데이터 추출
스트림 데이터에서 피치 추출(Stream to pitch)	스트림 데이터로부터 시계열데이터 추출
중첩 루프 조인(Nested loop join)	조인 수행
최소 그룹핑(Min groupping)	동일한 여러 개의 결과를 하나로 만듦

표 2 기호의 의미

기호	의미
T_N	음악에 대한 N길이의 시계열 데이터 T_N = <(음조$_1$, 음의 길이$_1$),(음조$_2$, 음의 길이$_2$), ..., (음조$_N$, 음의 길이$_N$)>
Q	허밍에서 추출한 시계열 데이터, 질의시퀀스
DS	미디 파일에서 추출한 시계열 데이터, 데이터시퀀스
DSs	데이터시퀀스들 DSs = Set(DS)
S	사용자 인터페이스로부터 넘어온 스트림 데이터
R	사용자에게 반환되는 결과 R = (노래번호, 윈도우번호, 유사도, ..., DS)

다. 왼쪽의 피연산자는 질의시퀀스이고, 오른쪽의 피연산자는 데이터베이스에 저장되어 있는 데이터시퀀스들을 의미한다. 결과가 Bag(DS)로 나타나는 이유는 서브시퀀스 매칭으로 인해 유사도가 다른 동일한 데이터시퀀스 DS가 여러 번 나올 수 있기 때문이다. 유사성 조건 c는 반환할 결과 개수 k와 유사성을 비교할 검색 방법 s로 이루어져 있다.

예를 들어, "아래 표 3과 같은 5개의 데이터시퀀스 DS와 질의시퀀스 Q가 주어졌을 때, k-최근접 검색을 이용하여 질의시퀀스와 유사한 3개의 데이터시퀀스를 찾아라"는 질의문을 연산자로 표현하면

$$(Q)\ \Omega_{3,k\text{-최근접 검색}}(DSs)$$

이고, 그 결과는 표 4와 같다. 위의 식에서 '3'과 'k-최근접 검색'은 질의시퀀스와 유사한 데이터시퀀스를 찾기 위해 필요한 유사성 조건들이다. 3은 검색되는 결과의 개수를 의미하고, k-최근접 검색은 유사 검색을 하는데 이용하는 방법을 의미한다.

표 3 데이터시퀀스 DSs와 질의시퀀스 Q

데이터 종류	시퀀스 데이터
질의시퀀스	<(51,10),(51,10),(51,10),(51,10),(51,10)>
데이터시퀀스 DS_1	<(50,10),(50,10),(50,10),(50,10),(50,10)>
데이터시퀀스 DS_2	<(59,10),(59,10),(59,10),(59,10),(59,10)>
데이터시퀀스 DS_3	<(52,10),(52,10),(53,10),(52,10),(52,10)>
데이터시퀀스 DS_4	<(60,10),(60,10),(60,10),(60,10),(60,10)>
데이터시퀀스 DS_5	<(45,10),(45,10),(45,10),(45,10),(45,10)>

표 4 $(Q)\ \Omega_{3,k\text{-최근접 검색}}(DSs)$

데이터 종류	시퀀스 데이터
데이터시퀀스1	<(50,10),(50,10),(50,10),(50,10),(50,10)>
데이터시퀀스3	<(52,10),(52,10),(53,10),(52,10),(52,10)>
데이터시퀀스5	<(45,10),(45,10),(45,10),(45,10),(45,10)>

3.1.2 윈도우 나눔(Φ)

$$\Phi\ (T_N) \Rightarrow DSs$$

이 연산자는 N길이의 시계열 데이터 T_N를 n($n \le N$) 길이의 하나 이상의 시계열 데이터 DSs로 나누는 연산자이다. 주로 한 곡의 노래로부터 추출한 시계열 데이터를 여러 마디의 시계열 데이터들로 나누거나 허밍을 하나 이상의 윈도우로 나눌 때 사용한다.

예를 들어, "시계열 데이터 T: <(1,10),(2,10),(3,10), ..., (99,10), (100,10)>를 디스조인트 윈도우 방법을 이용하여 나누어라. 단, 윈도우 사이즈는 10이다."라는 질의문이 있으면, 이것은

$$\Phi_{\text{디스조인트},10}(T)$$

로 표현하고, 그 결과는 표 5와 같다. '디스조인트'와 '윈도우 사이즈 10'은 이 연산자를 수행하는데 필요한 조건들로, 디스조인트는 하나 이상의 윈도우로 나누는 방법을 의미하고 윈도우 사이즈 10은 윈도우의 크기를 의미한다. 추가적인 조건으로 슬라이딩 사이즈가 있다.

표 5 $\Phi_{\text{디스조인트},10}(T)$

	시퀀스 데이터
데이터시퀀스1	<(1,10),(2,10),(3,10), ..., (10,10)>
데이터시퀀스2	<(11,10),(12,10),(13,10), ..., (20,10)>
...	...
데이터시퀀스9	<(81,10),(82,10),(83,10), ..., (90,10)>
데이터시퀀스10	<(91,10),(92,10),(93,10), ..., (100,10)>

3.1.3 저차원 변환(Ψ)

$$\Psi\ (T_N) \Rightarrow T_d'$$

N길이의 시계열 데이터 T_N를 d($d \ll N$)길이의 시계열 데이터 T_d'로 변환할 때 사용하는 연산자이다. 이 변환은 단순히 시계열 데이터의 길이만을 변화시키는 것이 아니라, N길이의 시계열 데이터의 정보를 대표할 수 있는 값으로 변환시킨다. 필요한 저차원 변환 조건으로는 저차원 변환의 종류 I과 변환할 차원 수 d가 필요하다. 데이터의 차원 수를 줄이는 방법에는 이산 퓨리에 변환[9], 구분적 집적근사, Keogh_PAA, Shasha_PAA 등을 이용하여 변환하는 방법들이 알려져 있다.

예를 들어, "100차원의 시계열 데이터 T: <(1,10),(2,10),(3,10), ..., (100,10)>를 구분적 집적근사 방법을 사용하여 8차원의 시계열 데이터로 변환하라"는 질의문

이 있으면, 이것은

$$\Psi_{PAA,8}(T)$$

으로 표현된다. 이 연산의 결과는 <(6,10), (18,10), (30,10), (42,10), (54,10), (66,10), (78,10), (90,10)>가 된다. 이 때, 구분적 집적근사가 저차원 변환의 종류가 되고, 차원 수는 8이 된다.

허밍 질의처리 시스템에서는 검색 성능을 향상시키기 위해 주로 다차원 인덱스를 사용하고 있는데, 다차원 인덱스들은 차원이 늘어날수록 검색 성능이 급속히 떨어지는 '차원의 저주'의 문제를 가지고 있다. 그러므로 이것을 피하기 위해서 N차원의 시계열 데이터를 N보다 아주 작은 d차원의 데이터로 변환하는 작업은 아주 중요하다.

3.1.4 스케일링(Δ)와 시프팅(Γ)

$$\Delta\ (T_k) \Rightarrow T_n$$

$$\Gamma\ (T_n) \Rightarrow T_n'$$

스케일링 연산자와 시프팅 연산자는 정규화 작업을 하는 연산자이다. 정규화란 두 시계열 데이터의 길이가 서로 다르고, 시계열 데이터의 음조 값의 위상 차이가 다른 데이터들을 동일한 길이와 동일한 위상을 가지는 시계열 데이터로 변환하는 것을 의미한다.

이 중에서 스케일링 연산자는 k길이의 시계열 데이터 T_k를 n길이의 시계열 데이터 T_n로 변환하는 연산자이다. 이것은 (n<k)나 (n≥k)인 어떠한 경우에라도 변환이 가능하다. 필요한 조건으로는 변환 후의 시계열 데이터의 길이를 얼마로 할 것인지를 결정하는 변수 n이 있다. 이 연산자를 이용하여 서로 다른 길이의 시계열 데이터를 동일한 길이를 가지는 시계열 데이터로 변환한다.

이에 반해, 시프팅 연산자는 n길이의 시계열 데이터 T_n의 음조 값들을 변환시키는데 사용한다. 이 연산자가 필요한 이유는 미디 파일에서 추출한 데이터와 허밍을 하는 사람의 목소리 상대적인 차이를 보정해 주기 위해서이다. 이 연산자를 수행하는데 필요한 조건으로는 무엇을 기준으로 음조 값들을 변환시킬 것인가를 정하는 c가 있으며, 종류에는 시계열 데이터의 음조 값들의 평균값을 빼는 방법과 시계열 데이터의 첫 번째 음의 값을 빼주는 방법 등이 있다.

"길이가 3인 시계열 데이터 T1: <(50,10),(60,10),(55,10)>와 길이가 6인 시계열 데이터 T2: <(25,10),(25,10),(35,10),(35,10),(30,10),(30,10)>이 존재할 때, T1을 길이가 6인 시계열 데이터를 만들어라"는 질의문은

$$\Delta_6(T1)$$

으로 표현이 가능하고, 그 결과는 <(50,10),(50,10),(60,10),(60,10),(55,10),(55,10)>이다. 길이가 6이라는 것이 스케일링 연산자의 조건이 된다.

또, "변환된 T1과 T2에 대해 평균값을 빼는 정규화 과정을 수행하라"는 질의문은

$$\Gamma_{평균값}(\Delta_6\ (T1)),\ \Gamma_{평균값}(T2)$$

로 표현이 가능하고 그 결과는 T1, T2 모두 <(-5,10),(-5,10),(5,10),(5,10),(0,10),(0,10)>가 된다. 평균값을 빼라는 것이 시프팅 연산자와 관련된 조건이다. 따라서, 길이가 3인 시계열 데이터 T1과 길이가 6인 시계열 데이터 T2는 스케일링과 시프팅의 정규화 과정을 거치고 나면 동일한 시계열 데이터가 된다.

3.1.5 미디에서 음조 추출(Θ)과 웨이브에서 음조 추출(Θ)

$$\Theta(\text{MIDI파일명, t}) \Rightarrow T$$

$$\Theta(\text{WAVE파일명}) \Rightarrow T$$

미디에서 음조 추출 연산자(Θ)는 미디 파일에서 시계열 데이터를 추출하기 위한 연산자이다. 미디 파일은 여러 트랙으로 이루어져 있기 때문에 추출하고자 하는 트랙의 번호 t를 지정해 주어야 한다. 주로 t는 가수가 노래 부른 멜로디가 적혀 있는 트랙의 번호가 사용된다. 예를 들어, "'남행열차.mid' 파일에서 4번 트랙의 멜로디로부터 시계열 데이터를 추출하라"는 질의문은

$$\Theta(\text{"남행열차.mid"}, 4)$$

로 나타낸다. 그 결과는 <(72,201),(72,201),(72,201),(70,201),(72,201), (75,402),(79,1811), ... >로 표현된다.

반면, 웨이브에서 음조 추출 연산자(Θ)는 허밍 데이터로부터 시계열 데이터를 추출하는 연산자이다. 허밍을 녹음할 때, 주로 컴퓨터의 마이크로폰을 이용하여 허밍을 모노 채널로 녹음하기 때문에 웨이브 파일에는 하나의 트랙만 존재한다. 그러므로 미디 파일에서 시계열데이터를 추출할 때 필요했던 트랙 번호는 필요가 없다. 예를 들어, "허밍 데이터 '남행열차.wav'에서 시계열 데이터를 추출하라"는 질의문은

$$\Theta(\text{"남행열차.wav"})$$

로 표현되며, 그 결과는 < (51,23), (51,23), (51,23), (51,23), (0,23), (0,23), (0,23), (52,23), (51,23), (51,23), (52,23), (51,23), (0,23), (0,23), ... >로 표현된다. 이 결과는 23초 단위로 하나의 음을 추출했기 때문이다.

3.1.6 스트림에서 음조 추출(Λ)

$$\Lambda(S) \Rightarrow T$$

이 연산자는 스트림 데이터로부터 시계열 데이터를 추출하기 위한 연산자이다. 사용자 인터페이스로부터 들어오는 스트림 데이터는 사용자 허밍에 대해 시계열 데이터를 추출한 정보가 스트링으로 변환되어 있다. 그러므로 이 연산자에서는 스트링으로 되어 있는 정보를 시계열 데이터로 변환하는 작업을 수행해야 한다.

"스트림 데이터 S가 '51 23 51 23 51 23 ...'일 때, 시계열 데이터를 추출하라"는 질의문은

$$\Lambda(S)$$

로 나타내고, 그 결과는 시계열 데이터 <(51,23),(51,23),

(51,23), ... >로 표현된다. 이 연산자를 이용하여 사용자 인터페이스에서 들어오는 스트림 데이터를 시계열 데이터로 변환한다.

3.1.7 중첩 루프 조인(⋈)

$$(Q) \bowtie_c (DSs) = \bowtie_c (Q, DSs) \Rightarrow Bag(R)$$

이것은 질의시퀀스 Q를 입력받아, 미디 파일에 대한 데이터시퀀스 DSs에서 질의 시퀀스와 유사한 것을 찾아 조인하기 위한 연산자이다. 이 연산자의 결과는 유사 검색을 수행한 결과인 데이터시퀀스들에, 각 데이터시퀀스가 나타내는 노래번호와 윈도우번호, 유사도 등을 추가한 Bag(R)이다. 이것은 유사 검색을 수행한 결과가 Bag(DS)로 반환될 수 있기 때문이다. 왼쪽 피연산자는 허밍에 대한 질의 시퀀스를 나타내고, 오른쪽의 피연산자는 미디 파일에 대한 데이터 시퀀스를 나타낸다. 조인 조건인 c는 질의 시퀀스와 유사한 결과를 몇 개 찾을 것인지를 나타내는 k와 어떠한 유사 검색 방법을 사용할 것인지를 나타내는 s로 이루어져 있다.

허밍과 유사한 데이터를 찾아 조인하라는 문제는 허밍 질의처리에서 허밍과 유사한 데이터 시퀀스를 찾는 문제로 변환하여 해결하도록 하였다. 왜냐하면 허밍 데이터에 존재하는 부정확성으로 인해 정확하게 같은 것을 찾는 기존의 조인 연산자로는 해결할 수 없기 때문이다. 그러므로 조인 조건에 유사 검색의 방법을 포함하여 되었다.

예를 들어, "표 3에 있는 5개의 데이터시퀀스 DSs가 주어졌을 때, 시계열 데이터 Q: <(51,10), (51,10),(51,10),(51,10),(51,10)>와 유사한 3개를 k-최근접 검색을 이용하여 찾아라"는

$$(Q) \bowtie_{3,k\text{-최근접 질의}} (DSs)$$

로 표현되며, 그 결과는 아래 표 6과 같다.

표 6 $(Q) \bowtie_{3,k\text{-최근접 질의}} (DSs)$

노래 번호	윈도우 번호	유사도	...	시퀀스 데이터
1	1	5		<(50,10),(50,10),(50,10),(50,10),(50,10)>
1	3	6		<(52,10),(52,10),(53,10),(52,10),(52,10)>
3	1	30		<(45,10),(45,10),(45,10),(45,10),(45,10)>

3.1.8 최소 그룹핑(||)

$$||(Bag(R)) \Rightarrow Set(R)$$

이것은 하위 연산자에서 받은 k개의 결과로부터 중복을 제거한 k개의 결과를 반환하는 역할을 수행한다. 중복이 생기는 이유는 서브시퀀스 매칭 방법을 이용하여 검색을 하므로 동일한 노래번호와 윈도우번호를 가진 결과가 여러 개 나타날 수 있기 때문이다.

이 연산자에서 결과를 처리하는 과정은 다음과 같다. 하위 연산자로부터 k개의 결과를 받는다. 이 결과들 중 동일한 노래번호와 윈도우번호를 가진 결과들이 있으면, 그 결과들 중에서 유사도가 가장 작은 것을 그 노래번호와 윈도우번호의 유사도로 결정하고 나머지 결과들은 제거한다. 제거된 결과 수만큼의 결과를 하위 연산자로부터 추가로 받으면서 동일한 제거 과정을 수행한다. 최종적으로 서로 다른 시계열 데이터를 가지는 k개의 결과를 상위 연산자에 반환한다. 예를 들어, "표 7과 같은 결과 R에 대해 최소 그룹핑 연산자를 적용하라"는 질의문은

$$||(R)$$

로 표현하고, 그 결과는 아래 표 8과 같다.

표 7 결과 R

노래 번호	윈도우 번호	유사도	...	시퀀스 데이터
1	1	4		<(50,10),(50,10),(50,10),(50,10),(50,10)>
3	1	20		<(45,10),(45,10),(45,10),(45,10),(45,10)>
1	1	25		<(50,10),(50,10),(50,10),(50,10),(50,10)>

표 8 ||(R)

노래 번호	윈도우 번호	유사도	...	시퀀스 데이터
1	1	4		<(50,10),(50,10),(50,10),(50,10),(50,10)>
3	1	20		<(45,10),(45,10),(45,10),(45,10),(45,10)>
2	1	45		<(60,10),(60,10),(60,10),(60,10),(60,10)>

3.2 질의처리 계획(Plan)

이 절에서는 질의문에 대한 파싱에서부터 앞 장에서 정의한 연산자를 이용하여 질의처리 플랜을 생성하고, 생성된 플랜을 실행하는 과정을 설명한다. 이러한 과정으로 질의문을 처리하는 것은 데이터베이스와 보다 쉽게 통합하기 위해서이다.

3.2.1 질의문 파싱

미디 파일을 삽입하고 사용자 허밍을 검색하는 플랜을 생성하기 위해, 먼저 질의문을 파싱한다. 두 개의 과정은 모두 SQL문과 유사한 형식으로 정의된 질의문을 통하여 처리된다.

질의문이 입력되면, 이것을 파싱하여 아래 그림 1과 같은 형태의 구조체에 정보를 저장한다. 질의문의 종류, 어떠한 방식으로 시계열 데이터를 나눌 것인지, 어떤 저차원 방법을 쓸 것인지 등 허밍 검색과 데이터베이스 구축을 위해 필요한 모든 정보를 저장한다.

각 항목에 대해 살펴보면, <queryType>는 질의문이 무엇을 수행하는 것인지를 나타내는 항목이다. 종류에는 미디 데이터 삽입, 미디 데이터 삭제, 허밍 검색, 인덱스 생성이 있다. <targetInfoVector>는 데이터를 어느 테

```
class PlanHead
{
public:
  QueryType                          queryType;
  std::vector<Info>                  targetInfoVector;
  std::vector<Info>                  sourceInfoVector;
  UnitType                           unitType;
  SplitInfo<int>                     splitInfo;
  AuxSplitType                       auxiliarySplitType;
  std::vector<NormalizationType>     normalizationTypeVector;
  DimReductionInfo                   dimReductionInfo;
  DistanceType                       distType;
  DTDInfo                            dtdInfo;
  MinGroupType                       minGrouppingType;
  int                                topK;
  Operator*                          rootOpNode;
}
```

그림 1 플랜 구조체

이블에 저장하고, 어떤 테이블에 대해 검색할 것인지를 가리키는 항목이다. <sourceInfoVector>에는 음악 데이터의 파일 포맷과 파일명이 저장된다. 만약, 파일 포맷이 미디 파일인 경우에는 메인멜로디 트랙 번호의 정보가 함께 저장된다. <unitType>은 시계열 데이터를 윈도우로 나눌 때 무엇을 기준으로 나눌 것인지를 설정하는 것으로, 기준 단위로 음표와 초(시간)가 있다. <splitInfo>은 시계열 데이터를 여러 개의 윈도우로 나눌 때 어떠한 기법을 쓸 것인지를 설정하는 정보가 저장된다. 윈도우를 나누는 방법에는 크게 슬라이딩 윈도우 방법과 디스조인트 윈도우 방법이 있다. 또한, 나눌 윈도우의 크기와 슬라이딩 크기에 대한 정보가 저장된다.

<auxiliarySplitType>은 어떤 종류의 보조윈도우를 생성할 것인지를 설정하는 것으로, 현재는 [2]에서 사용하고 있는 방법만을 지원하고 있다. <normalization-Type>은 정규화 방법들 중에서 어떠한 방법을 적용할 것인지 설정한다. <dimReductionInfo>는 저차원 변환할 때 사용할 변환 방법과 함께 차원의 수가 저장된다. <distanceType>은 검색 시, 데이터 시퀀스와 질의 시퀀스의 유사도를 측정할 때 사용하는 거리 함수를 저장한다. 거리 함수의 종류에는 씨티블럭, 동적 시간 와핑, 지역적 동적 시간 와핑, 유클리디안 등이 있다. <dtdInfo>는 음표의 개수를 비교할 것인지 결정하는 항목이고, <minGroupType>은 여러 개의 동일한 결과가 나왔을 시, 가장 작은 유사도를 가지는 결과로 결정할 것인지 안 할 것인지를 정한다. <topK>는 사용자에게 반환해야 하는 결과의 개수를 의미한다.

각 항목별로 저장된 데이터를 바탕으로 만들어진 구조체는 그림 2와 같은 형태를 지낸다. 그림 2는 아래의 예제에 나온 질의문을 파싱하여 만들어진 그림이다.

예제. 허밍이 녹음된 '남행열차.wav'파일과 유사한 10개의 노래를 찾아라. 단, 윈도우는 슬라이딩 윈도우 기법을 사용하여 나누고, 윈도우 사이즈는 16, 슬라이딩 사이즈는 4이다. 윈도우 사이즈의 기준은 4분 음표의 길이가 기준이다. 스케일링과 평균값에 의한 시프팅의 정규화 과정을 수행하고 구분적 집적접근을 사용하여 8차원으로 저차원 변환한다. 음조 차이의 분포와 최소 그룹핑은 사용하지 않는다.

<rootOpNode>는 지금까지 설정한 항목들을 컴파일해서 나오는 연산자 트리의 루트 노드를 가리키는 변수이다. 이 <rootOpNode>는 컴파일 과정을 거친 후에야 연산자 트리의 루트 노드를 가리킬 수 있다.

3.2.2 플랜 트리 생성 및 실행

컴파일 과정에서는 그림 1과 같이 정의된 구조체의 정보로부터 플랜 트리를 생성한다. 구조체에 있는 각 항목과 파라미터들을 이용하여 해당되는 연산자를 생성하고, <rootOpNode>에 생성된 연산자를 순차적으로 연

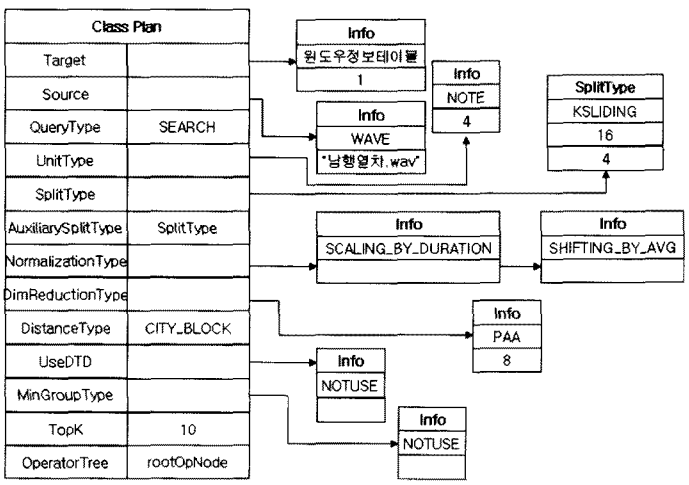

그림 2 플랜 구조체의 예

결합으로서 플랜 트리(또는 연산자 트리)를 생성한다. 각 연산자는 자신의 상위 연산자로 결과를 하나씩 전달하기 위해 반복(iteration) 방식을 지원하는 GetNext() 함수를 가지고 있다. 이것을 사용하여 결과들을 상위 연산자로 하나씩 반환한다.

컴파일을 통해 만들어진 플랜 트리는 실행 과정에서 최상위 연산자를 실행함으로써, 질의문을 실행한다. 데이터베이스 시스템 내부에서도 이와 유사한 과정으로 질의문을 처리하고 있으므로, 제안하고 있는 연산자를 보다 쉽게 데이터베이스 시스템에 통합할 수 있다. 다음 장에서는 앞에서 설명한 연산자와 플랜을 지원하는 시스템에 대해서 살펴본다.

4. 허밍베이스 시스템

본 절에서는 제 3절에서 설명한 허밍 대수를 지원하는 허밍베이스 시스템의 구조와 함께 데이터베이스 구축과 허밍 데이터의 검색 방법에 대해 논의한다. 제 4.1절에서는 허밍베이스 시스템의 구조에 대해 설명하고, 제 4.2절에서는 검색 방법에 대해 논의한다. 제 4.3절에서는 데이터베이스 구축 방법에 대해 설명한다.

4.1 시스템 구조

허밍베이스 시스템은 크게 서버 컴포넌트와 클라이언트 컴포넌트로 구성된다. 서버는 다시 검색 엔진, 데이터베이스 구축 엔진, 미디 파일에서 시계열 데이터를 추출하는 모듈로 구성된다. 반면에, 클라이언트는 사용자 인터페이스와 웨이브 파일에서 시계열 데이터를 추출하는 모듈로 구성된다. 시스템의 전체적인 구조는 아래 그림 3과 같다.

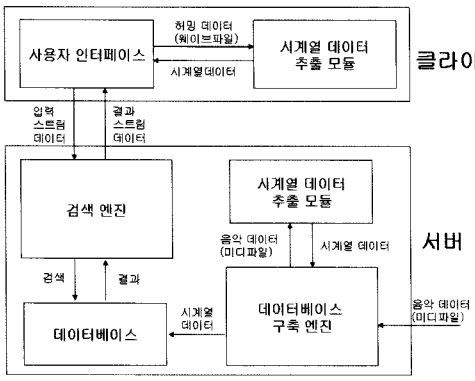

그림 3 HummingBase 시스템 구조

다음 절에서부터 서버와 클라이언트를 구성하는 각 모듈에 대해 살펴본다. 먼저 서버를 구성하는 모듈에 대해서 논의한 후, 클라이언트에 대해 설명한다.

4.1.1 검색 엔진 및 데이터베이스 구축 엔진

검색 엔진은 사용자 허밍과 유사한 멜로디를 가진 노래를 인덱스와 데이터베이스를 이용하여 찾는 역할을 수행한다. 사용자 인터페이스로부터 받은 질의문, 즉 스트림 데이터를 파싱하고, 이 정보들을 바탕으로 플랜을 생성한다. 생성된 플랜을 실행함으로써 허밍과 유사한 멜로디를 가진 노래를 찾는다. 검색된 결과는 스트림 데이터로 만들어져 사용자 인터페이스에 반환된다. 질의문이 검색 엔진에서 처리되는 과정은 제 4.2절에서 자세하게 설명한다.

데이터베이스 구축 엔진은 미디 파일에서부터 데이터를 추출하여, 추출된 데이터를 이용하여 인덱스를 생성하고 데이터베이스를 구축하는 역할을 수행한다. 추출된 데이터 뿐 만 아니라 노래 제목 등의 메타데이터 정보도 함께 데이터베이스에 저장한다. 이 작업 역시 질의문을 통해 수행된다. 데이터베이스를 구축하고 인덱스를 생성하는 과정은 제 4.3절에서 자세하게 설명한다.

4.1.2 시계열 데이터 추출 모듈들

본 시스템은 두 가지 종류의 음악 파일로부터 시계열 데이터를 추출하는 모듈을 제공한다. 하나는 데이터베이스를 구축하기 위해 미디 데이터로부터 시계열 데이터를 추출하는 모듈이고, 다른 하나는 검색을 위해 사용자 허밍인 웨이브 파일로부터 시계열 데이터를 추출하는 모듈이다.

미디 파일에는 전체 트랙 수, 곡의 빠르기(tempo), 박자(time signature), 어떤 악기로 어떠한 음을 얼마 동안 연주한다는 등 여러 정보들이 들어있다[12]. 일반적으로 하나의 트랙은 하나의 악기로 연주되는데, 대부분의 가수가 노래 부르는 멜로디 또한 이러한 트랙 중의 하나에서 연주된다. 본 시스템에서는 여러 정보들 중에서 가수가 노래 부르는 멜로디가 적혀 있는 트랙(본 논문에서는 메인멜로디 트랙이라 부름)에서 시계열 데이터를 추출한다.

미디 파일에서 시계열 데이터를 추출하는 모듈은 C++ 미디 라이브러리인 libjdkmidi[13]를 수정하여 만들었다. 이 라이브러리와 미디 파일에 기록되어 있는 메인멜로디 트랙, 노래의 박자와 빠르기, 연주되는 음의 높이 값, 음의 연주 시간을 이용하여, 메인멜로디 트랙에 해당하는 미디 데이터 정보로부터 음조와 음의 길이를 추출하여 시계열 데이터를 생성하였다. 한 곡의 노래를 하나의 시계열 데이터로 표현하였다.

허밍이 녹음된 웨이브 파일로부터 음조들을 추출하기 위해 Robert C. Maher et al's[14]의 방법을 사용하였다. 추출된 음조 값들은 '0'과 '0 아닌 값'으로 구성된다. 어떠한 소리가 있을 때는 '0 아닌 값'이, 소리가 없을 때는 '0'값이 나온다. 일반적으로 쉼표가 '0'의 값을 가진다. 추출된 음조 값들을 이용하여 허밍 데이터를 시계열 데이터로 표현했다.

4.1.3 사용자 인터페이스

사용자 인터페이스에서는 사용자로부터 허밍을 입력받아 서버에 있는 검색 엔진에 질의할 수 있는 형태로 데이터를 변환하고, 변환된 데이터를 질의하여 찾아낸 검색 결과를 확인하는 역할을 수행한다. 사용자 인터페이스는 아래 그림 4와 같다. 이 그림은 남행열차의 첫 소절을 허밍해서 질의한 결과를 보여준다.

사용자 인터페이스는 4개의 창으로 구성된다. 사용자로부터 허밍을 녹음하고, 녹음된 허밍을 연주하고, 시계열 데이터로 변환하고, 이러한 여러 데이터를 볼 수 있는 "WAV Analysis" 창, 미디 파일을 불러들여 연주하고 보여주는 "MIDI Analysis" 창, 허밍을 데이터베이스에 질의한 결과를 보여주고, 그 결과로 나온 노래를 연주할 수 있는 "Query Results" 창, 사용자 허밍과 매치되는 결과를 시계열 데이터로 보여주는 "Part Visualizer" 창으로 구성된다.

사용자 인터페이스에서 사용자가 허밍을 녹음하는 것에서부터 결과를 확인하는 과정은 다음과 같다. 사용자는 "WAV Analysis" 창의 "record" 버튼을 눌러 컴퓨터의 마이크로폰을 이용하여 모노 형식의 '타'로 허밍을 녹음한다. '타'로 허밍 하는 이유는 웨이브 데이터로부터 음조를 추출해 내기가 좋기 때문이다[15,16]. 녹음된 데이터는 "Convert" 버튼을 통해 시계열 데이터로 변환되고 이를 "WAV Analysis" 창에 그려준다. 또한, 변환된 시계열 데이터는 음표 형식으로 변환되어 "MIDI Analysis" 창에 보여준다. "Query"라는 버튼을 누름으로서 변환된 시계열 데이터는 검색 엔진에서 처리할 여러 정보와 함께 스트림 데이터로 변환되어 서버에 질의되고, 그 결과가 "Query Results" 창에 나타난다. 변환된 시계열 데이터는 (음조1 음의 길이1 음조2 음의 길이2 ...)의 형식으로 스트림 데이터 속에 포함된다. "Top-K" 항목의 숫자 값에 따라 서버로부터 반환되는 결과의 개수가 달라진다.

4.2 허밍 질의 검색 과정

검색 엔진은 사용자 인터페이스로부터 넘어온 스트림 데이터를 파싱하여 플랜을 생성하고, 이것을 실행함으로써 검색 과정을 수행한다. 스트림 데이터는 플랜을 생성하는 정보와 허밍에서 추출한 시계열 데이터로 구성되어 있다.

스트림 데이터를 이용하여 검색하는 과정은 다음과 같다. 우선, 스트림 데이터를 파싱하여 허밍에 대한 시계열 데이터에 해당하는 부분을 분리한 후, 플랜을 만드는데 필요한 정보로부터 플랜을 생성한다. 플랜을 실행

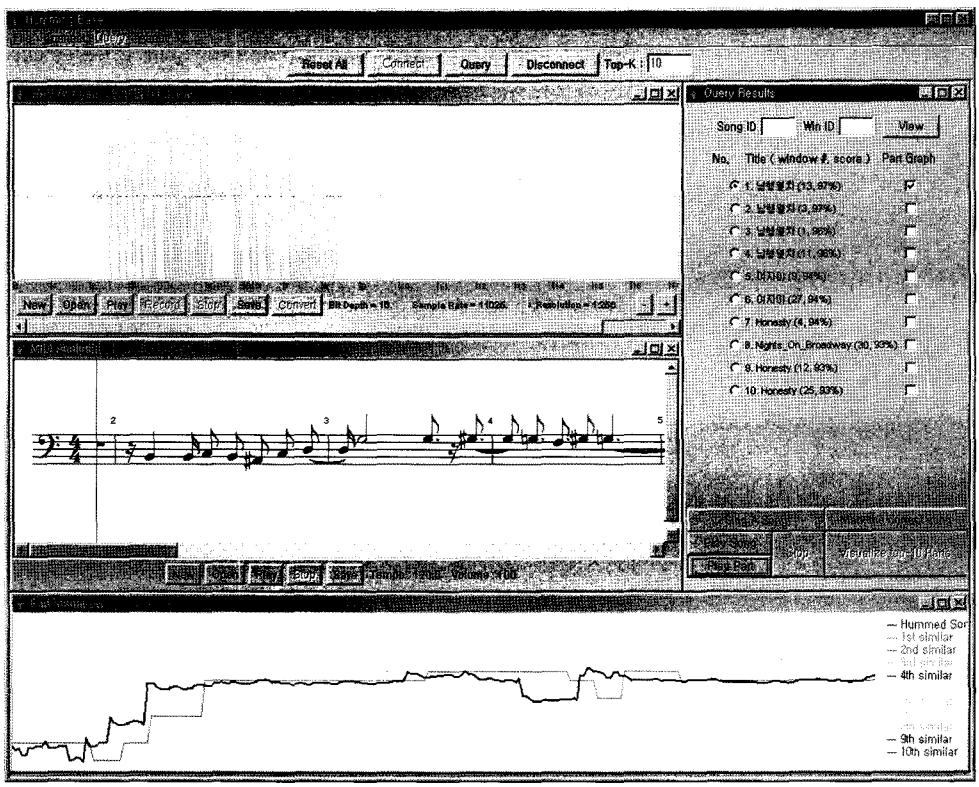

그림 4 사용자 인터페이스

해 순서대로 실행하면서 검색 과정을 수행하는데, 가장 먼저 시계열 데이터에 해당하는 스트림 데이터를 시계열 데이터로 변환한다.

변환된 시계열 데이터를 디스조인트 또는 슬라이딩 윈도우 방법을 사용하여 하나 이상의 윈도우로 나눈 다음, 각 윈도우마다 스케일링과 시프팅의 정규화 과정을 거친다. 정규화의 과정을 거친 후, 윈도우는 인덱스에 질의하기 위해 저차원 변환 과정을 수행한다. 일반적으로 저차원 변환된 데이터의 차원 수는 인덱스에 저장되어 있는 데이터의 차원 수와 동일하다.

스트림 데이터로부터 추출한 시계열 데이터와 저차원 변환된 데이터, 사용자가 찾기를 원하는 결과 개수 k와 유사 검색 방법을 이용하여 시계열 데이터와 유사한 k개의 결과를 찾아 사용자에게 반환한다. 만약 검색을 수행하여 반환되는 결과 중, 동일한 노래 아이디와 윈도우 번호를 가진 결과가 나타나게 되면 유사도가 적은 값을 가진 결과만 반환하고 줄어든 결과만큼 추가적으로 결과를 추출하여, 최종적으로 k개의 결과를 사용자에게 반환한다. 아래 그림 5는 검색 엔진에서 스트림 데이터가 질의되는 전체 과정을 나타내는 흐름도이다.

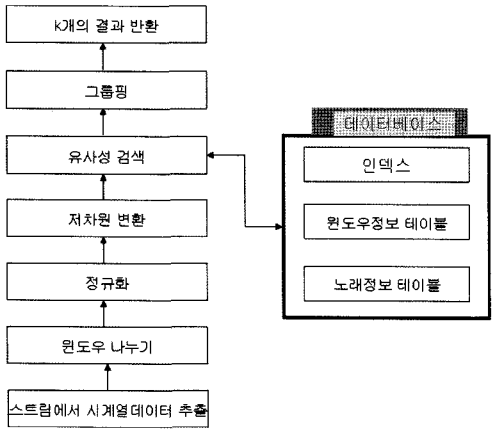

그림 5 검색 과정 흐름도

4.3 데이터베이스 구축 과정

데이터베이스는 미디 파일로부터 추출한 시계열 데이터를 이용하여 구축한다. 미디 파일에서 추출된 시계열 데이터는 노래 제목과 노래 아이디 등의 메타정보와 함께 "노래정보 테이블"에 저장된다. 하나의 미디 파일은 노래정보 테이블의 1개의 레코드와 대응된다. 노래정보 테이블의 스키마는 아래 표 9와 같다.

시계열 데이터는 윈도우를 생성하는 기법을 사용하여 하나 이상의 윈도우로 나누고, 나누어진 각 윈도우에 대해 정규화 과정을 거친다. 정규화 과정을 거친 시계열 데이터에 대해 저차원 변환을 수행하여 저차원의 시계열 데이터를 생성한다. 정규화 과정을 거친 시계열 데이터는 노래의 아이디와 노래에서 몇 번째 윈도우인지를 나타내는 정보와 함께 "윈도우 정보 테이블"에 저장된다. 윈도우 정보 테이블의 스키마는 아래 표 10과 같다.

저차원으로 변환된 시계열 데이터는 일반화된 검색 트리(GiST: Generalized Search Tree)[1]를 이용하여 인덱스로 구축한다. 이상의 과정을 정리해 보면 아래 그림 6과 같다.

5. 사례 연구

본 절에서는 지금까지 설명한 허밍 대수를 기존 허밍 질의처리 시스템에 적용시켜 본다. 이를 통하여 본 논문에서 제안하고 있는 허밍 대수로 기존 시스템을 표현할 수 있음을 보여준다. 다음부터 관련 연구에서 설명한 두 허밍 질의처리 시스템을 연산자로 표현한다.

5.1 Zhu et al's 시스템[4]

본 논문에서 제안하는 대수로 표현하기에 앞서 [4]에 대해 다시 살펴보자. 이것의 인덱스 구축 과정은 다음과 같다. 1) 미디 데이터로부터 멜로디를 추출한다. 2) 하나의 멜로디를 여러 개의 서브 멜로디로 나눈다. 3) 각 서브 멜로디에 대해 'w-upsampling'과 'shift-invariant technique'이라 불리는 정규화 과정을 거친다. 4) 저차원 변환을 수행한다. 5) 인덱스를 구축한다.

이러한 과정을 본 논문에서 제안하는 연산자로 표현해 보면 그림 7과 같다. 연산자 이름 밑의 소괄호에 적혀 있는 것은 이 논문에서 사용하고 있는 방법을 적어 놓은 것이다. 연산자 밑에 아무것도 없는 것은 논문에서 어떠한 방법을 사용했는지가 기록되어 있지 않는 것이다. 점선으로 표시된 것들은 있어도 되고, 없어도 상관없는 것을 나타낸다.

표 9 노래 정보 테이블 스키마

필드명	타입	설명	비고
SongID	정수	노래번호	주요키
MainMelodyTrackNo	정수	메인멜로디트랙번호	
Title	스트링	노래제목	
Data	<(음조, 음의 길이)>	시계열데이터	
Length	정수	시계열데이터 길이	

표 10 윈도우 정보 테이블 스키마

필드명	타입	설명	비고
SongID	정수	노래번호	주요키
KthWindow	정수	k번째 윈도우	주요키
StartPosition	정수	노래에서 윈도우 시작 위치 (second)	
EndPosition	정수	노래에서 윈도우 끝나는 위치 (second)	
WindowRange	(정수, 정수)	노래에 대한 시계열데이터 상의 위치	
Data	<(음조, 음의 길이)>	윈도우에 해당하는 시계열 데이터	

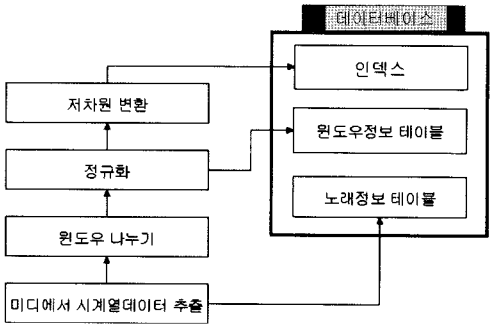

그림 6 데이터베이스 구축 흐름도

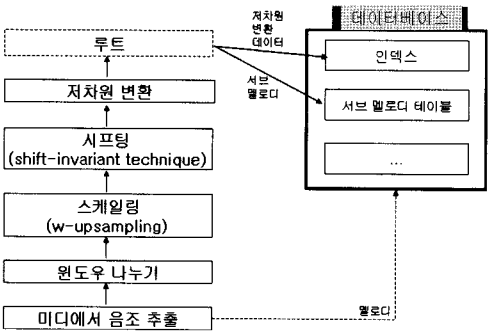

그림 7 [4]의 데이터베이스 구축 과정을 연산자로 표현

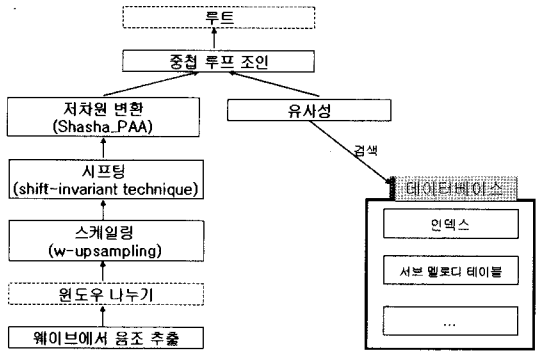

그림 8 [4]의 검색 과정을 연산자로 표현

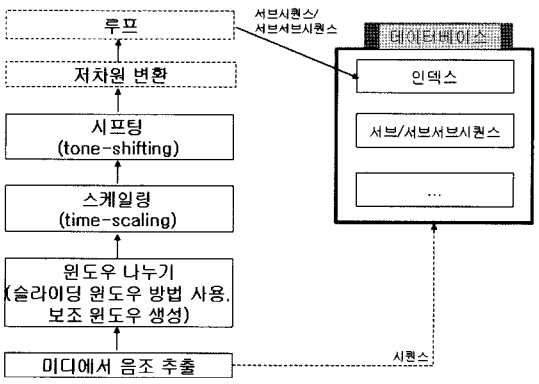

그림 9 [2]의 데이터베이스 구축 과정을 연산자로 표현

다음으로 검색 과정에 대해 살펴보자. 1) 우선, 사용자 허밍으로부터 음조를 추출하여 시계열 데이터를 만든다. 2) 전체 매칭 방법을 사용하기 위해 허밍 전체를 질의하는데 사용한다. 3) 서브 멜로디에 했던 것과 동일하게 정규화 과정을 수행한다. 4) 논문에서 제안하는 Shasha_PAA로 저차원 변환한다. 5) 저차원 변환한 데이터를 이용하여 인덱스에 ε-범위 질의를 수행하여 결과가 될 수 있는 후보 집합을 찾은 후, 이 후보 집합 중에서 허밍과 가장 유사한 결과를 찾는다. 이것을 연산자로 표현하면 그림 8과 같다.

5.2 SoundCompass 시스템[2]

이것의 데이터베이스 구축 과정은 다음과 같다. 1) 먼저 미디 파일에서 데이터를 추출한다. 2) 추출한 데이터를 서브/서브서브시퀀스로 나눈다. 3) 서브/서브서브시퀀스에 음가에 대한 음의 비율이라 불리는 정규화 과정을 거쳐 고정 길이의 윈도우를 생성한다. 음가에 대한 음의 비율은 'time-scaling'과 'tone-shifting'으로 이루어져 있다. 4) 추가로 가변 길이의 보조 윈도우를 생성한다. 5) 두 종류의 윈도우를 각각 인덱스로 구축한다. 일련의 과정을 본 논문에서 제안하는 연산자로 표현하면 그림 9와 같다.

검색 과정은 다음과 같다. 1) 먼저 사용자 허밍으로부터 음조를 추출하여 데이터를 추출한다. 2) 추출한 데이터를 서브시퀀스/서브서브시퀀스로 나눈다. 3) 음가에 대한 음의 비율이라 불리는 정규화 과정을 거쳐 고정 길이 윈도우를 생성한다. 4) 추가로 가변 길이의 보조 윈도우도 생성한다. 5) 두 개의 윈도우를 각각 해당하는

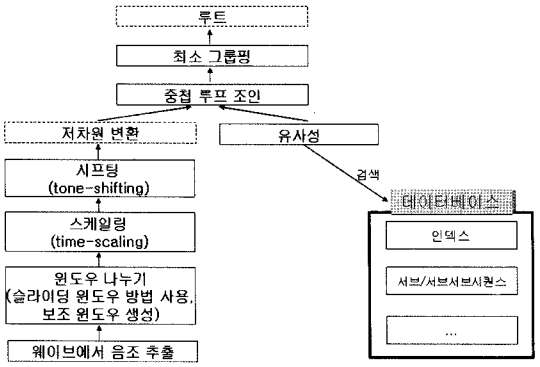

그림 10 [2]의 검색 과정을 연산자로 표현

인덱스에 질의하여 검색한다. 6) 검색된 결과에서 동일한 서브/서브서브시퀀스가 하나 이상 나왔을 경우, 가장 작은 유사도를 그 서브/서브서브시퀀스의 유사도로 결정한다. 이 과정을 연산자로 표현하면 그림 10과 같다.

지금까지 본 논문에서 제안한 대수를 이용하여 기존 허밍 질의처리 시스템의 주요 과정을 표현하였다. 각 연산자는 자신이 해야 하는 역할을 다양한 방법으로 처리할 수 있도록 일반화하여 설계되었기 때문에, 서로 다른 두 시스템을 그림 7부터 그림 10까지에서처럼 동일한 연산자를 이용하여 표현할 수가 있다.

6. 결 론

본 논문에서는 허밍 질의를 처리하는 시스템인 허밍 베이스를 제안하였다. 먼저 허밍 질의를 처리하기 위한 대수를 제안하기 위해, 기존 시계열 데이터에 대한 유사 검색 방법들을 고찰하였다. 다음으로, 이에 기반하여 10개의 연산자를 가지는 허밍 대수를 제안하였다. 그리고, 각 연산자의 의미에 대해서 정의하였으며, 주어진 질의에 대해서 질의문을 파싱하고 허밍 질의를 처리하는 자료구조에 대해서 설명하였다. 다음으로 구축한 시스템의 구조, 주요 모듈, 사용자 인터페이스에 대해서 설명하였다. 허밍 대수를 사용함으로써 데이터베이스 구축 및 사용자 허밍 질의 검색이 쉽게 표현될 수 있음을 보였다. 또한, 사례 연구를 통해 허밍 대수를 이용해서 기존의 두 가지 전형적인 허밍 질의 처리 시스템을 구현할 수 있음을 보였다.

향후 연구로는 실제 데이터베이스 시스템 내부에 논문에서 제안한 대수를 이용하는 허밍베이스 시스템을 통합하는 것을 고려해 볼 수 있다.

참 고 문 헌

[1] Ghias, A., Logan, J., Chmberlin, D., and Smith, C., "Query by humming: Musical information retrieval in an audio database," In *ACM Multimedia 1995*, pp.231-236, 1995.

[2] Kosugi, N., Sakurai, Y., and Morimoto, M., "SoundCompass: A Practical Query-by-Humming System," In *Proceedings of ACM SIGMOD*, pp.881-886, 2004.

[3] Uitdenbgerd, A., and Zobel, J., "Melodic matching techniques for large music databases," In *ACM Multimedia 99*, pp.57-66, 1999.

[4] Zhu, Y., and Shasha, D., "Warping indexes with envelope transforms for query by humming," In *Proceedings of ACM SIGMOD*, pp.181-192, June 2003.

[5] Agrawal, R., Faloutsos, C., and Swami, A., "Efficient similarity search in sequence databases," In *Proc. the 4th Int'l Conf. on Foundations of Data Organization and Algorithms*, pp.69-84, 1993.

[6] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y., "Fast Subsequence Matching in Time-Series Databases," In *Proc. Int'l Conf. on Management of Data, ACM SIGMOD*, Minneapolis, Minnesota, pp.419-429, May 1994.

[7] Moon, Y., Whang, K., and Han, W., "General Match: A Subsequence Matching Method in Time-Series Databases Based on Generalized Windows," In *Proc. Int'l Conf. on Management of Data, ACM SIGMOD*, Madison, Wisconsin, pp.382-393, June 2002.

[8] Keogh, E., "Exact indexing of dynamic time warping," In *Proceedings of VLDB*, pp.406-417, August 2002.

[9] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S., "Dimensionality reduction for fast similarity search in large time series databases," *Journal of Knowledge and Information Systems*, pp.263-286, 2000.

[10] Faloutsos, C., Lin, K., "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," *SIGMOD Conference*, pp.163-174, 1995.

[11] Seidl, T., and Kriegel, H., "Optimal multi-step k-nearest neighbor search," *SIGMOD Conference*, pp.154-165, 1998.

[12] Standard MIDI Files 1.0, http://jedi.ks.uiuc.edu/~johns/links/music/midifile.html

[13] J.D.Koftinoff Software, ltd. C++ MIDI Library - jdkmidi class library documentation, 2004.

[14] Maher, R., and Beauchamp, J., "Fundamental frequency estimation of musical signals using a two-way mismatch procedure," *Journal of the Acoustical Society of America*, vol.95, no.4, pp. 2254-2263, 1994.

[15] Kosugi, N., Nishihara, Y., Sakata, T., Yamamuro, M., and Kushima, K., "A Practical Query-By-Humming System for a Large Music Database," In *Proc. of the 8th ACM International Conference*

on Multimedia, pp.333-342, 2000.
[16] Rodger McNab, "INTERACTIVE APPLICATIONS OF MUSIC TRANSCRIPTION," Master's thesis, Computer Science at the University of Waikato, 1996.

신 제 용

2005년 경북대학교 컴퓨터공학과 졸업(공학사). 2007년 경북대학교 컴퓨터공학과 졸업(이학석사). 2007년~현재 ITMEX SYI 근무

한 욱 신

1994년 경북대학교 컴퓨터공학과 졸업(공학사). 1996년 한국과학기술원 전산학과 졸업(이학석사). 2001년 한국과학기술원 전산학과 졸업(공학박사). 2003년~현재 경북대학교 컴퓨터공학과 부교수

이 종 학

1982년 경북대학교 전자공학과(전자계산전공) 졸업(학사). 1984년 한국과학기술원 전산학과 졸업(공학석사). 1997년 한국과학기술원 전산학과 졸업(공학박사) 1991년 정보처리기술사. 1984년~1987년 금성통신(주) 부설연구소 주임연구원. 1987년~1998년 한국통신 연구개발본부 선임연구원. 1998년~현재 대구가톨릭대학교 컴퓨터정보통신공학부 교수