멜로디의 유사성 측정을 위한 멜로디 형태 표현법 고안

안단태 ○ 서진욱
서울대학교 컴퓨터공학부
dtan@hcil.snu.ac.kr, jseo@snu.ac.kr

Designing a Melodic Contour Representation for Measuring Melodic Similarity

Dantae An○ Jinwook Seo
Department of Computer Science and Engineering, Seoul National University

요 약

듣기 좋은 음악을 자동으로 생성할 때 주로 기존의 곡으로부터 음악적 특징을 분석하고 학습하는 접근법을 취한다. 본 연구에서는 중요한 음악적 특징 중 하나인 멜로디 형태를 컴퓨터에서 표현하고 다루는 방법을 제안한다. 이 표현법은 리듬 패턴과 음 높이 변화 패턴을 동시에 나타내며 화음 정보와는 독립적이다. 그리고 사람이 느끼는 멜로디의 유사성을 반영하여 두 멜로디 형태 사이의 거리를 정량적으로 측정하는, 동적 프로그래밍 기법을 이용한 알고리즘을 제안한다. 이 표현법을 활용하면 멜로디 형태에 임의의 화음을 결합하여 새로운 멜로디를 생성할 수 있고, 음악이 가진 반복적인 구조 또한 쉽게 파악할 수 있다.

1. 서 론*

자동 음악 생성은 음악 정보 검색 및 인공지능 분야에서 활발히 연구되고 있는 주제이다. 하지만 컴퓨터를 이용하여 듣기 좋은 음악을 만드는 것은 현재까지도 잘 정의되지 않은 문제이다[1]. 어떤 음악이 듣기 좋은 음악인지에 대한 정의가 존재하지 않고, 이들이 가지는 공통된 특징도 명확하게 알려져 있지 않기 때문이다. 그래서 사람이 작곡한, 듣기 좋은 곡으로부터 자동으로 특징을 학습하여 음악을 생성하려는 시도가 다수 존재했다. 이때 음악을 화성 진행 패턴, 리듬 패턴, 음 높이 변화 패턴, 곡 구조 패턴 등의 음악적 특징에 따라 분리하여 학습하는 것이 이론적으로 가능하지만, 이 중에서 리듬 및 음 높이 변화 패턴의 경우 적절한 표현법이 존재하지 않아 다루기 어려웠다.

우리는 멜로디 형태(melodic contour)의 특징을 표현하는 새로운 표현법을 도입한다. 이 표현법은 리듬 패턴과 음 높이 변화 패턴을 동시에 나타내며 화음 정보와는 독립적이다. 따라서 알려진 멜로디에서 추출한 멜로디 형태에 임의의 화음을 결합하여 새 멜로디를 생성하는 것이 가능하다. 또한 사람이 느끼는 멜로디의 유사성을 반영하여 두 멜로디 형태 사이의 거리를 정량적으로 측정하는 알고리즘을 제안한다.

2. 멜로디 형태 표현법

2.1 멜로디 형태

우리는 음악 시퀀스의 비유사도를 정의한 Mongeau와 Sankoff의 논문[2]을 참조하여 새로운 멜로디 형태 표현법을 고안하였다. 우리의 표현법에서 하나의 멜로디 형태는 단성(monophony) 멜로디의 한 마디를 나타내며, 이를 첫 쉼표 하나와 음표 목록으로 정의할 수 있다. 첫 쉼표는 마디의 시작부터 첫 음표의 시작 위치(onset)가 등장하기까지의 시간을 나타내며, 이 길이를 64분음표 단위로 0 이상의 정수로 표현한다. 그 뒤로 놓이는 음표들은 음표 목록에 앞에서부터 순서대로 추가된다. 각각의 음표는 길이 값, 음 높이 클러스터 값, 그리고 음 높이 변화 값을 가진다. 길이 값은 현재 음표의 시작 위치와 바로 다음 음표의 시작 위치(또는 마디의 끝) 사이의 시간을 64분음표 단위로 표현한 1 이상의 정수이다. 이때 음표의 끝 위치(offset)나 음표 사이의 쉼표는 고려하지 않는다. 음 높이 클러스터 번호 값은 다른 음표에 상대적인 현재 음표의 음 높이를 나타내는 실수이다. 임의의 두 음표의 음 높이 클러스터 번호 값을 비교하여 서로 같으면 두 음표의 실제 음 높이가 같고, 한 쪽이 더 크다면 실제 음 높이도 큰 쪽이 더 높다. 음 높이 변화 값은 음표들의 순서와 음 높이 클러스터 번호 값이 결정되면 자동으로 계산되는 값이다. 현재 음표의 바로 앞에 놓인 음표와 비교하여 현재 음표의 음 높이 클러스터 번호 값이 더 높으면 '/', 더 낮으면 '\', 같으면 '0'으로 표현한다. 이때 마디의 첫 번째 음표의 음 높이 변화 값은 항상 '0'이다.

이렇게 음 높이 변화를 절대적인 음정으로 세세히 표현하지 않고 방향 정보로만 표현하는 것이 듣는 사람들로 하여금 멜로디의 유사성을 결정하는 데 더 효과적이며[3] 멜로디 인식에도 중요한 역할을 한다[4]. 그리고 음 높이 변화 값은 두 멜로디 형태 간의 거리를 정의할 때 중요하게 사용된다.

한편 이 표현법은 화음 정보를 담고 있지 않다. 따라서 두 멜로디의 화음이 다르더라도 리듬과 음 높이 변화 패턴에 따라 유사한 멜로디 형태로 인식할 수 있고, 멜로디를 생성할 때 그림 1처럼 어떤 화음과 어떤 멜로디 형태가 주어지더라도 이들을 결합하여 새로운 멜로디를 생성할 수 있다.

* 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF2019R1A2C208906213).

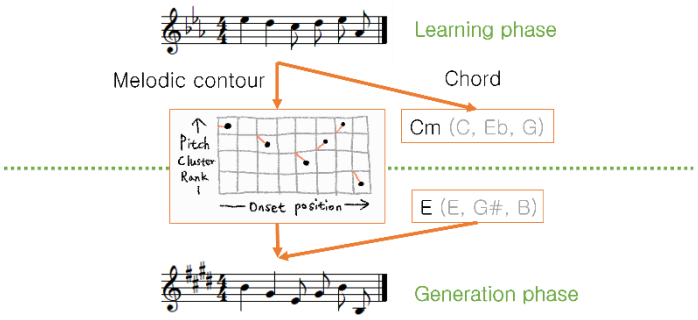

그림 1 멜로디 형태 표현법과 화음의 분리 및 결합

2.2 멜로디 형태 편집 연산

두 멜로디 형태의 차이를 정량적으로 측정하는 방법 중 하나로, 편집 연산과 편집 연산 수행 비용을 정의하고 한 멜로디 형태에서 편집 연산들을 적용하여 다른 멜로디 형태로 변형할 때 발생하는 비용의 최솟값을 사용하는 방법이 있다[2]. 우리는 멜로디 형태에 가할 수 있는 여섯 가지 편집 연산을 다음과 같이 정의하였다.

$Delete(mc, i)$ 는 멜로디 형태 mc 의 음표 목록에서 앞에서부터 i번째에 위치한 음표 하나를 제거하는 연산이다.

$Insert(mc, i, note)$는 멜로디 형태 mc의 음표 목록의 i번째 위치에 새 음표 $note$ 하나를 추가하는 연산이다.

$Replace(mc, i, note)$ 는 멜로디 형태 mc 의 음표 목록에서 i 번째에 위치한 음표 하나의 길이 값과 음 높이 클러스터 번호 값을 새 음표 $note$가 가진 값으로 변경하는 연산이다.

$Delay(mc, rest)$는 멜로디 형태 mc의 첫 쉼표의 길이 값을 0 이상의 정수 $rest$로 변경하는 연산이다.

$Move(mc, i, note_1, note_2)$는 멜로디 형태 mc의 음표 목록에서 앞에서부터 i번째에 위치한 음표를 새 음표 $note_1$로 변경하고 $i+1$ 번째에 위치한 음표를 새 음표 $note_2$ 로 변경하는 연산이다. 이때 연산 수행 전 mc의 i번째 및 $i+1$번째 음표의 길이 값의 합이 $note_1$ 및 $note_2$ 의 길이 값의 합과 같아야 한다. 이 연산을 통해 다른 음표들의 시작 위치를 바꾸지 않고 $i+1$번째 음표의 시작 위치만 바꾸는 것이 가능하다.

$Delay\ and\ Replace(mc, rest, note)$ 는 멜로디 형태 mc 의 첫 쉼표의 길이 값을 0 이상의 정수 $rest$ 로 변경하고 음표 목록에서 첫 음표를 새 음표 $note$로 변경하는 연산이다. 이때 연산 수행 전 mc 의 첫 쉼표 및 첫 음표의 길이 값의 합이 $rest$ 및 $note$ 의 길이 값의 합과 같아야 한다. 이 연산은 $Move$의 효과를 첫 음표에 적용하는 특별한 연산이다.

Mongeau와 Sankoff[2]가 정의한 편집 연산 중에는 하나의 긴 음표를 같은 음 높이의 여러 짧은 음표들로 쪼개는 fragmentation과 같은 음의 연속된 여러 음표들을 하나로 합치는 consolidation이 있다. 그러나 이 연산들은 수행하는 비용에 비해 멜로디를 크게 변형시키기 때문에 음악을 생성하는 상황에는 잘 맞지 않는다[5]. 우리는 이 연산들을 제외하는 대신 $Move$ 등의 새로운 편집 연산들을 도입하였다.

2.3 멜로디 형태 편집 비용

편집 연산을 한 번 수행할 때마다 연산을 적용한 음표들과 해당 음표의 바로 뒤에 있던 음표(존재한다면)의 음 높이 변화 값을 새로 계산하며, 상황에 맞게 연산 수행 비용을 발생시킨다. 우리는 멜로디 형태의 리듬 패턴과 음 높이 변화 패턴의 변동을 나타내는 여섯 종류의 비용을 정의하였다.

불가능한 연산 비용(IC)은 연산 적용 전후의 멜로디 형태가 동일하거나 $Move$ 와 $Delay\ and\ Replace$ 에서 두 음표 및 쉼표의 길이의 합이 연산 전후로 달라지는 경우에 발생하며, 비용 값은 양의 무한대(∞)이다. 이는 거리 계산 알고리즘이 의미 없는 연산을 수행하지 않도록 하기 위해 필요하다.

음표 길이 변경 비용(DC)은 하나의 음표 또는 쉼표의 길이가 변하고 이로 인하여 그 뒤에 놓인 하나 또는 여러 음표의 시작 위치가 모두 변할 때 발생한다. $Delete$, $Insert$ 수행 시 항상 발생하고, $Replace$, $Delay$의 경우 연산 적용 후 해당 음표 또는 쉼표의 길이 값이 달라지는 경우에 발생한다. $Move$, $Delay\ and\ Replace$ 수행 시에는 발생하지 않는다. 비용 값은 설정 가능한 변수(hyperparameter)인 w_{DC}로 정의한다.

음표 시작 위치 변경 비용(OC)은 한 음표의 시작 위치가 변하고 다른 모든 음표의 시작 위치가 변하지 않을 때 발생한다. $Move$, $Delay\ and\ Replace$ 수행 시 항상 발생하고 다른 연산 수행 시에는 발생하지 않는다. 비용 값은 w_{OC} 로 정의한다.

음표 음 높이 변화 변경 비용(PVC)은 음표 하나의 음 높이 변화 값이 변할 때 발생한다. $Delete$ 수행 시, 제거되는 음표의 음 높이 변화 값이 '0'이 아니었을 때 한 번 발생한다. $Insert$ 수행 시, 삽입되는 음표의 음 높이 변화 값이 연산 수행 후에 '0'이 아닐 때 한 번 발생한다. $Replace$ 수행 시, 변경된 음표의 음 높이 변화 값이 연산 전후로 변할 때 한 번 발생한다. $Move$ 수행 시 비용 발생 조건은 $Replace$ 와 동일하지만, 음표 두 개를 변경하므로 그 중 음 높이 값이 바뀐 음표의 개수($k_{PV} \in \{0,1,2\}$)만큼 반복하여 비용이 발생한다. $Delay$ 또는 $Delay\ and\ Replace$ 수행 시에는 발생하지 않는다. 비용 값은 $k_{PV} \times w_{PVC}$로 정의한다.

음표 음 높이 순위 변경 비용(PRC)은 연산을 적용한 음표의 음 높이 클러스터 번호가 바뀌어서, 멜로디 형태의 음표 목록에서 해당 음표를 포함하여 이보다 앞에 놓인 모든 음표들을 고려하여 음 높이 클러스터 번호의 순위를 계산하였을 때, 해당 음표의 음 높이 클러스터 번호의 순위가 연산 전후로 바뀌는 경우에 발생한다. 연산 적용 당시 변경하려는 음표보다 음표 목록에서 뒤에 놓인 음표는 고려하지 않는다. PVC와 유사하게, $Delete$, $Insert$, $Replace$ 수행 시 최대 한 번($k_{PR} \in \{0,1\}$) 발생할 수 있고, $Move$ 수행 시 음 높이 클러스터 번호의 순위 값이 달라진 음표 개수($k_{PR} \in \{0,1,2\}$)만큼 반복하여 비용이 발생한다. 비용 값은 $k_{PR} \times w_{PRC}$로 정의한다.

서로 다른 음 높이 개수 변경 비용(PCC)은 위의 비용들과 달리 편집 연산을 적용할 때마다 발생하는 비용이 아니다. 두 멜로디 형태의 비유사도(dissimilarity)를 계산할 때 두 멜로디 형태에서 각각 서로 다른(unique) 음 높이 클러스터 번호의 개수 k_{PC1}, k_{PC2}를 세어 이 둘의 차이만큼 반복하여 발생하는

비용이다. 비용 값은 $|k_{PC1} - k_{PC2}| \times w_{PCC}$로 정의한다.

우리가 경험적으로 찾은 최적의 값은 각각 $w_{DC} = 3$, $w_{OC} = 3$, $w_{PVC} = 4$, $w_{PRC} = 1$, $w_{PCC} = 3$이다. 이는 필요에 따라 다르게 설정할 수 있다. 한편 각 편집 연산에서 IC가 발생하지 않았을 때 발생하는 비용을 수식으로 정리하면 다음과 같다.

$$cost_{Delete}(mc, i) = DC + PVC + PRC$$
$$cost_{Insert}(mc, i, note) = DC + PVC + PRC$$
$$cost_{Replace}(mc, i, note) = DC + PVC + PRC$$
$$cost_{Delay}(mc, rest) = DC$$
$$cost_{Move}(mc, i, note_1, note_2) = OC + PVC + PRC$$
$$cost_{Delay\ and\ Replace}(mc, rest, note) = OC \quad (1)$$

2.4 멜로디 형태 거리

동적 프로그래밍(dynamic programming) 기법을 이용하면 다항 시간(polynomial time) 내에 최적에 가까운, 두 멜로디 형태의 비유사도를 구할 수 있다. 음표 개수가 m인 한 멜로디 형태 A에서 음표 개수가 n인 다른 멜로디 형태 B로 변형하는 비유사도를 구할 때, $d_{i,j}$를 A의 앞에서부터 i개의 음표들을 B의 앞에서부터 j개의 음표들로 변형하는 비용 합의 최솟값으로 정의한다. 여기에서 i, j는 각각 $0 \le i \le m$, $0 \le j \le n$인 정수이다. 주의할 점은 모든 편집 연산을 임의의 순서로 적용할 수 있다는 것이다. 항상 앞에 놓인 음표부터 편집 연산을 적용해야 할 필요는 없다.

최종 비유사도 계산에 필요한 값 $d_{m,n}$을 구하기 위해, i와 j를 각각 0, 0부터 시작하여 1씩 늘리면서, 모든 $d_{i,j}$를 아래 네 수식으로부터 재귀적으로 구한다. 수식에서 $p_{i,j}$는 최적의 $d_{i,j}$를 계산하기 위해 수행한 연산들을 수행한 시간 순서대로 정렬한 시퀀스이며, $length(p_{i,j})$는 $p_{i,j}$의 연산 개수이다. $mc_{i,j,k}$는 연산 시퀀스 $p_{i,j}$의 앞에서부터 k개의 연산($0 \le k \le length(p_{i,j})$)을 멜로디 형태 A에 순서대로 모두 수행하여 얻은 멜로디 형태이다. 이때 $mc_{i,j,0} = A$ 이고 $mc_{m,n,length(p_{m,n})} = B$이다. mc_i는 mc의 i번째 음표이고, mc_{rest}는 mc의 첫 쉼표의 길이 값이며, $index(mc_{i,j,k}, note)$는 $mc_{i,j,k}$의 음표 목록의 앞에서부터 음표 $note$가 놓인 순서이다.

$i = 0$, $j = 0$일 때,

$$d_{0,0} = 0 \quad (2)$$

$i > 0$, $j = 0$일 때,

$$d_{i,0} = \min(\{d_{i-1,0} + cost_{Delete}\left(mc_{i-1,0,k}, index\left(mc_{i-1,0,k}, A_i\right)\right) \mid 0 \le k \le length(p_{i-1,0})\}) \quad (3)$$

$i = 0$, $j > 0$일 때,

$$d_{0,j} = \min(\{d_{0,j-1} + cost_{Insert}\left(mc_{0,j-1,k}, index\left(mc_{0,j-1,k}, B_j\right), B_j\right) \mid 0 \le k \le length(p_{0,j-1})\}) \quad (4)$$

$i > 0$, $j > 0$일 때,

$$d_{i,j} = \min \begin{cases} \{d_{i-1,j} + cost_{Delete}\left(mc_{i-1,j,k}, index\left(mc_{i-1,j,k}, A_i\right)\right) \mid 0 \le k \le length(p_{i-1,j})\} \\ \{d_{i,j-1} + cost_{Insert}\left(mc_{i,j-1,k}, index\left(mc_{i,j-1,k}, B_j\right), B_j\right) \mid 0 \le k \le length(p_{i,j-1})\} \\ \{d_{i-1,j-1} + cost_{Replace}\left(mc_{i-1,j-1,k}, index\left(mc_{i-1,j-1,k}, A_i\right), B_j\right) \mid 0 \le k \le length(p_{i-1,j-1})\} \\ \{d_{i-2,j-2} + cost_{Move}\left(mc_{i-2,j-2,k}, index\left(mc_{i-2,j-2,k}, A_{i-1}\right), B_{j-1}, B_j\right) \mid 0 \le k \le length(p_{i-2,j-2})\} \\ (i > 1, j > 1) \\ d_{0,0} + cost_{Delay\ and\ Replace}\left(mc_{0,0,0}, B_{rest}, B_1\right)\ (i = 1, j = 1) \end{cases} \quad (5)$$

최종 연산 시퀀스 $p_{m,n}$에 $Delay\ and\ Replace$가 포함되어 있지 않다면 마지막에 $Delay$ 연산을 한 번 수행해야 한다. $Delay$ 연산 수행 시 발생하는 비용을 $cost_{Delay}(A, B_{\text{rest}})$로 정의하자. 멜로디 형태 A에서 B로 변형하는 비유사도를 계산하는 식은 수식 (6)과 같다.

$$dissimilarity(A, B) = d_{m,n} + cost_{Delay}(A, B_{\text{rest}}) + PCC \quad (6)$$

두 멜로디 형태 A와 B의 거리(distance)는 A에서 B로 변형하는 비유사도와 B에서 A로 변형하는 비유사도 중 더 작은 값으로 정의한다. 이는 수식 (7)과 같이 계산한다.

$$distance(A, B) = \min(dissimilarity(A, B), dissimilarity(B, A)) \quad (7)$$

3. 토의 및 결론

본 연구에서는 곡의 구조를 분석하거나 멜로디를 생성할 때 유용하게 사용될 수 있는 새로운 멜로디 형태 표현법을 도입하였다. 임의의 단성 멜로디 악보에서 각 마디 별로 리듬 정보와 음 높이 변화 정보를 추출하여 멜로디 형태로 표현할 수 있다. 이 멜로디 형태에는 화음 또는 구체적인 음 높이 정보가 들어있지 않다. 그리고 멜로디 형태 간의 거리를 정의하여 두 멜로디 형태의 유사한 정도를 정량적으로 측정할 수 있게 하였다. 이때 리듬 정보를 더 중요하게 고려하려면 2.3에서 도입한 w_{DC}와 w_{OC} 값을 크게 설정하고, 음 높이 변화 정보를 더 중요하게 고려하려면 w_{PVC}, w_{PRC}, w_{PCC}의 값을 크게 설정하면 된다.

향후 연구에서는 멜로디 형태 표현법을 활용하여, 곡의 구조를 인간 참여형(human-in-the-loop)으로 분석할 수 있는 시스템을 구현하고 사례 연구를 진행하고자 한다. 대부분의 음악에는 반복되는 구조가 나타나는데, 이를 우리의 멜로디 형태 표현법이 효과적으로 포착할 수 있을 것으로 기대한다. 또한 멜로디 형태 거리 값이 실제로 사람이 느끼는 멜로디 간 유사성을 얼마나 잘 반영하는지 검증할 예정이다.

참 고 문 헌

[1] D. Herremans *et al.*, "A Functional Taxonomy of Music Generation Systems," *ACM Computing Surveys (CSUR)*, 50(5):1-30, 2017.

[2] M. Mongeau and D. Sankoff, "Comparison of Musical Sequences," *Computers and the Humanities*, 24(3):161-175, 1990.

[3] S. Handel, *Listening*, Cambridge, MA: MIT Press, 1989.

[4] W. J. Dowling, "Scale and contour: Two components of a theory of memory for melodies," *Psychological Review*, 85(4):341-354, 1978.

[5] F. Pachet *et al.*, "Sampling Variations of Sequences for Structured Music Generation," *The 18th International Society for Music Information Retrieval Conference*, 2017.