Content-based Visualization of Classical Music for Recommendation and Pattern Discovery

Kevin Yen

Advisor: Dr. Han-Wei Shen

Committee: Dr. Christopher Charles Stewart

ABSTRACT

Recommendation system has been a hot topic in many areas. For example, in Isenberg's research[1], metadata such as author-assigned keyword were used to create hierarchical structure of topics that contains multiple keywords. By using such metadata, they have created Keyvis.org[2] that allows researchers to search for papers and navigate through keywords under the same topic for other related papers. The generation of keyword is a content-based process, since the keywords are basically a higher level view of the paper. For entertaining purposes, music recommendation has been a hot topic for major music streaming companies[3], e.g. Pandora, Apple Music, Spotify, Last.fm, etc. Majority of the solutions are based on metadata of a track, like artist, album, language, style, lyrics composer, year, etc. Such recommendation system relies on a well-constructed database that has all the song tracks' tags, and some companies like Pandora are manually tagging music for the best precision. Natural language processing (NLP) is one of the cores of many researches which allowed them to obtain information from a long paragraph or article. However, such method doesn't apply very well on music, since music is different from language. Fred Lerdahl and Ray Jackendoff have come up with a music theory[4] called *A generative theory of tonal music*, and it defined several rules that would allow a piece of music to be split into smaller pieces according to the rules that reflected musician's insight and knowledge of music, eventually become a structured tree. In this project, we have utilized the music theory to transform a piece of music into tree structure, and by calculating the difference of each tree, we are able to obtain a distance matrix, which can be used for calculating similarity after applying MDS. Also by utilizing the tree structure, we have come up with a visualization that allow user to see how a music look like in its tree form, and discover some possible patterns of a music which can be useful for music majored researchers to quickly understand the structure of a music track for further analysis purposes.

I. INTRODUCTION

Discovery of new things is the basic instinct of every human, and technology has helped us to make it more efficient and accurate. However, before internet and computer technology, books in bookstores and libraries are always categorized. In such fashion, people can quickly discover the books that they haven't seen yet, but are very similar to what they have been looking for. This habit of putting things together can be seen in almost every physical store, from outwear stores to CD stores. However, as technology evolves, people's habit of shopping has changed. Through Internet, shopping online is a norm that everyone embraces in their everyday life. For the companies that offer online shopping, it is different from what physical stores used to do. Instead of laying bunch of similar items upon user's screen, they have to carefully pick what they are going to recommend.

In many areas, the recommendation system has always been a hot topic. In academic area, researchers do not have the time and effort to go through every single paper that has been published. In fact, in medical area, the speed of generating the medical reports is way beyond single human's ability to digest while doing productive research. For example, Medline database[5] consists so many medical reports and paper, they need a group of experts to manually read through all the articles that are newly published and categorize them into a large hierarchical indexing structure, called Medical Subject Headings (MeSH). Such structure is a categorized tag that allow doctors and researchers to quickly obtain new knowledge about certain area without going through too much reading of irrelevant articles and reports.

Content-based recommendation has been a major focus for recommendation systems. In Isenberg's research[1], experts in Visualization field has manually chosen keywords for papers that are accepted by IEEE Visualization Conference. They have grouped several times to physically meet up to discuss about the precision and the selection of keywords to use. As a result, they have come up with a topic system that contains several high-level topics, and underneath them are groups of keywords. With 2,792 conference papers from 1990 to 2015 in their database, they have constructed an online system called Keyvis.org[2] that is used to recommend papers based on keyword. It would recommend other keywords that co-occur with the current one, so user would know what other keywords are more relevant to it. Also, it introduces the other keywords under its topic, ordering from the most frequent to the least. In such way, it is easy for user to find other publication based on a few keywords and therefore can extend the research faster and more efficient.

With examples that showed how other areas have been recommending based on content, many of the music recommendation systems are still focused on metadata recommendation. It's true that when a user is interested in one song of an artist, very likely can also be interested in other songs by the same artist, too. Metadata recommendation usually consists of the common tags used for songs and artists. For example, the genre, the year of publication, and the artist's nationality and language.

Instead of these obvious high-level data, Spotify[3] uses the data gathered from users to their recommendation for each user, "Discover Weekly". For example, when user A has history of listening to music 1, 2, and 3, while user B has a playback history of music 2, 3, and 4, having music 2 and 3 in common, Spotify would then recommend music 1 to user B and recommend music 4 to user A. This pattern is based on the belief that users tend to listen to and like certain kind of songs despite the metadata. In this system, when the user base is huge, like Spotify, it amazes people by how great the recommendation is on their "Discover Weekly", since nobody can know what you would like, but there's a good chance that the people listening to similar songs will share the same taste as you do. Echo Nest, a music data platform serving for developers and media companies like Spotify, has developed a content-based recommendation system. By analyzing the audio signals of music, they can retrieve information such as energy, loudness, danceability, liveness, speechiness, hotttnesss, tempo, etc.

With a different direction of content-based recommendation, this project focuses on analysing classical music based on music theory that would take the music notes of a music, pitch and duration, into account. Fred Lerdahl and Ray Jackendoff have developed a music theory[4] called *A generative theory of tonal music (GTTM)*. They are musician and linguistic experts, and they have laid down the rules that can be used to analyze music based on the musical insight and understanding they have developed over their career. Of many other music theories available, GTTM was simply one of many that have been chosen to be implemented into computer language. GTTM reveals the musical structure and concepts by using rules to describe the musical fundamental knowledge. The framework was well described consisting several rules for grouping, metrical-structure analysis, and time-span structure analysis. Hamanaka have translated most of the rules in his research[6]–[8] into computer language. By continuously introducing new rules from GTTM, Hamanaka make sure that enough rules are implemented to create reasonable results. The output of the implementation is time-span tree, which is a hierarchical binary tree that still keeps the order of the notes from left to right.

Music theory like GTTM introduces a way of seeing a music by transforming the data structure from plain music into hierarchical tree structure. The rules considered and the design of the structure can be crucial to one's recognition to music. In Koelsch's research[9], they have proved that the output of GTTM does preserve the music knowledge level information pretty well by doing brain electric response experiment. By comparing the same music with one of them having the rearranged layout of time-span tree, they have discovered that the structure of time-span tree can influence how human brain process music which is related to human cognition.

In this project, we try to utilize the time-span tree of several classical music from well-known composers to create a simple recommendation system based on the music themselves instead of the genre and other high-level information. Also, in this project we presented a visualization of combining time-span tree and piano roll, which would allow user to quickly identify the patterns in the music.

II. Related Work

1. A generative theory of tonal music

In this section, the music theory used to analyze music is briefly introduced. By understanding the rules that are used to make sense of a music in terms of music knowledge, it would be easier and more convincing for anyone to understand how music theory works and why is it a reliable way of content-based information retrieval of a music.

In Hamanaka's research[8], they focused on implementing the rules from *A generative theory of tonal music (GTTM)*[4] into computer language. In GTTM, there are three main structure used for constructing a time-span tree, grouping structure, metrical structure, and time-span tree reduction structure. Grouping structure focuses on the basic elements of the music, and converts music into a hierarchical structure representation. Metrical structure focuses on determining the strong and week beats at the hierarchical levels. Time-span reduction is based on the previous two, grouping and metrical structure. Moving from smaller levels to larger levels, time-span reduction constructs a tree named time-span tree.

The benefit of having such music theory implemented, is to reduce manual work from the pipeline. Before implementing GTTM into computer language, human would step into the pipeline, and determine the structures while considering all the rules for grouping, metrical analyzing, and time-span reduction.

In the implementation, not all the rules are included, since only monophony music is considered. The rules implemented doesn't include the ones that are related to harmony. In total, they have introduced 15 parameters for grouping structure, 18 parameters for metrical structure, and 13 for time-span reduction. The implementation grantees that by applying the rules, it can generate correct analysis results. Also, they have introduced weights in every parameter, which is intended for full automation of analyzing music with GTTM rules. By introducing weights to each parameter, they can solve some implementation problems, like the conflict in rules that tend to split a group into phrases at different position while there were no preferences between the rules.

1.1 Grouping structure parameters

The grouping structure divides notes into smaller phrases while minimizing the conflict to music knowledge. It's like playing a flute, and looking for the moments to breath without misinterpreting the music. The rules would introduce several levels of groups, and in each level, bigger groups will be split into smaller phrases. The process of grouping analysis is basically applying the rules to all groups that is able to be split into smaller groups until the rules don't apply. The parameters used for grouping analysis will be slightly introduced in the following. The i represents the i^{th} note.

ρ_i is the offset-to-onset duration
ι_i is the onset-to-onset duration
η_i is the pitch difference
δ_i is the velocity difference
α_i is the articulation difference
β_i is the duration difference

GPR2 consists two rules, 2a and 2b.

Rule **2a** focuses on the offset-to-onset duration, which is the offset of i^{th} note to the onset of $i+1^{th}$ note. If ρ_i is greater than ρ_{i-1} and ρ_{i+1} then GPR2a for i^{th} note will be 1. Otherwise 0.

Rule **2b** focuses on the onset-to-onset duration. If ι_i is longer than ι_{i-1} and ι_{i+1}, then GPR2b score for i^{th} note will be 1, otherwise 0.

GPR3 consists four rules, 3a, 3b, 3c, and 3d.

Rule **3a** focus on the pitch different, which is the pitch difference between $i\text{-}1^{th}$ note and $i+1^{th}$ note. If η_i is greater than η_{i-1} and η_{i+1}, then the GPR3a score is 1, otherwise 0.

Rule **3b** focus on the change of velocity. If the δ_i is greater than δ_{i-1} and δ_{i+1}, then GPR3b score for i^{th} note is 1, otherwise 0.

Rule **3c** focus on the change of articulation. If the articulation $\alpha_{i-1} = 0$, $\alpha_{i+1} = 0$, and $\alpha_i \neq 0$, then GPR3c score for i^{th} note is 1, otherwise 0. Articulation changes when certain notes need to be played with longer or shorter duration then normal. With D_i representing the full duration of i^{th} note, and d_i representing the played duration of i^{th} note, α_i can be presented with the following.

$$\alpha_i = \left|\frac{D_{i+1}}{d_{i+1}} - \frac{D_i}{d_i}\right|$$

Rule **3d** focus on the change of duration. Duration of ith note is the onset of ith note to onset of $i+1$th note. $\beta_{i-1} = 0$, $\beta_{i+1} = 0$, and $\beta_i \neq 0$, the the GPR3d score for ith note is 1, otherwise 0.

GPR4 is based on GPR2 and GPR3's score. At a given threshold T_{GPR4}, if the maximum effects of GPR2 and GPR3 at ith note are larger than T_{GPR4}, then GPR4 score is 1, otherwise 0. $P_R(i)$ are basically the local average of each effects of GPR2 and GPR3. $R \in \{\rho, \iota, \eta, \delta, \alpha, \beta\}$.

$$D_{GPR4}(i) \begin{cases} 1, if \max(P_R(i)) > T_{GPR4} \\ 0, otherwise \end{cases}$$

GPR5 focuses on the symmetry element of a group. The value is basically a normal distribution where in the middle of the group, the value is highest at 1, and the most boundary notes have the lowest GPR5 score.

GPR6 focuses on the parallelism of the group. By iterating through every notes in the group, with length starting from 1 to maximum possible, two pairs of notes are compared with each other. GPR6 is looking for the identical elements of these two pairs in terms of pitch and duration. The longer the pairs, the higher the score will be.

GPR1 takes most the GPR scores into account. Starts by calculating $B^{low}(i)$ value for every notes, where $B^{low}(i)$ is the sum of all the GPR rules, 2a ,2b ,3a ,3b ,3c ,3d, and 6, then normalized to 1 as the largest number. Upon summing the GPR rules, it also takes the weight/strength of each GPR score into account. For note i, GPR1 is 1 when the $B^{low}(i)$ is greater or equal to $B^{low}(i-1)$ and $B^{low}(i+1)$, while $GPR1(i-1) = 0$.

To start the hierarchy grouping, we need to first calculate $D^{low}(i)$, which is 1 if $B^{low}(i)$ is greater than a threshold T^{low} and $D_{GPR1}(i) = 1$. Then we calculate B^{high}, which is different from B^{low} since B^{high} considers GPR4 and GPR5 which changes as the group gets cut into smaller groups. Then we calculate the final parameter for grouping stage, D^{high}. $D^{high}(i) = D^{low}(i) * B^{high}(i)$. It will always be a value between 0 and 1, and it changes when the group gets split into smaller groups. The highest value of $D^{high}(i)$ will be where the cut happens.

1.2 Metrical structure parameters

The Metrical structure analysis tries to identify the characteristics of note beats, and tries to find strong and weak beats at several metrical levels. Basically, it is trying to make sense of the human instinct that we tend to clap at certain beats while listening to music. The following section will briefly introduce what rules has been used.

γ_i is the velocity of note i
λ_i is the length of note i
ζ_i id the duration of velocity of note i
κ_i is the length of slur of note i
ν_i is the pitch of note i

MPR1 focuses on parallelism. Using the exact same way of how GPR6 was calculated, but the interval now is the result of grouping analysis instead of any arbitrary range. It is basically trying to find similar notes from all the smaller groups. MPR1 ensures that some similar phrases will be remained in metrical analysis.

MPR2 focuses on the beat strength. It tends to let the earlier notes have higher score by using a linear distribution of score. The starting note will have 1, and the ending note will have 0.

MPR3 simply gives a score of 1 if γ_i is larger than 0, and gives 0 if $\gamma_i = 0$. Meaning, the rests will have 0, while all other notes that create sound gives 1.

MPR4 focuses on reflecting the stress. With a given threshold T_{MPR4}, if γ_i is larger than $2\times\gamma\times T_{MPR4}$ where γ is the average velocity of all notes, it gives score 1, and 0 if otherwise. This reflects if a certain notes are above the threshold of velocity, tends to find the strong velocity notes.

MPR5 consists 5 rules, 5a, 5b, 5c, 5d, and 5e.

Rule **5a** focuses on finding the notes that last longer than a set threshold. 5a gives 1 if λ_i is greater than $2\times\lambda\times T_{MPR5a}$ where λ is the average length of all notes.

Rule **5b** focuses on finding the notes that have long duration of velocity. If ζ_i is greater than $2\times\zeta\times T_{MPR5b}$ where ζ is the average duration of dynamic of all notes, then 5b will be 1, and 0 if otherwise. 5b tends to find the notes that are either having longer articulation or stronger velocity.

Rule **5c** focuses on finding the notes that are under slur. If κ_i is larger than $2\times\kappa\times T_{MPR5c}$ where κ is the average length of slur among all the notes, 5c will have value 1, and 0 if otherwise. 5c will most likely pick the notes that is the starting point of a slur if most of the other notes do not have slur.

Rule **5d** focuses on finding the notes that are the starting point of strong beat. It is based on MPR5a. If MPR5a(i) = 1 and MPR5a(i+1) = 1, then 5d gives 1, and 0 if otherwise. This will yield a similar result to MPR5a since it is based on it, but depending on the weight of 5a and 5d, there can be a preference between these two similar parameters.

Rule **5e** focuses on finding the neighboring notes that share the same pitch. If $\nu_i = \nu_{i+1}$, it returns 1, and 0 if otherwise.

To start constructing the hierarchical metrical structure, we need to first calculate $B^{metrical}$(i), which is the sum of 2, 3, 4, 5a, 5b, 5c, 5d, and 5e times by their individual weight. Then we calculate $D^{metrical}$(i), which is the value of $B^{metrical}$(i), except for all the notes where MPR1(i,j) =1, they will have the value $B^{metrical}$(i) summing the summation of $B^{metrical}$(j) for all MPR1(i,j) =1. Finally, using $D^{metrical}$(i) as the score for each note, several combinations of notes will be selected as a group, and the notes with the largest metrical score $D^{metrical}$(i) will be given a dot. The combinations are i = {1,3,5,…}, {2,4,6,…}, {1,4,7,…}, {2,5,8,…}, and {3,6,9,…}. In simple words, for every odd notes, the largest notes will be given a dot, then for every even notes, they will be given another dot, and so on. The combination is basically considering whether the music is composed in simple duple time or simple triple time. As a result of metrical structure analyzing, every notes or rests will be marked with dots if it is one of the highest scored notes in one of the combination.

1.3 Time-span reduction

Time-span reduction is the process of trying to differentiate important notes from the less important ones. It will yield a most important note as the head of a subtree, so time-span tree is able to describe a music with the important notes only by reducing the information that are less important which are the notes that are not the head of a subtree.

The process starts by considering all the notes as head, and choose the heads for next level by calculating local head strength. The process is repeated until there's no more than one head for subtrees. Applying the grouping analysis first, the music will be split into hierarchical groups. Then apply metrical analysis to every group that has more than one note. With the dots given from the metrical analysis as the criteria for selecting a head note, the hierarchical tree will then be formed.

μ_i is the number of dots marked from metrical analysis
ϕ_i is the offset-to-onset duration
ψ_i is the onset-to-onset duration
ξ_i is the difference in pitch

TSRPR1 focuses on looking for the notes that have the strongest beat in metrical analysis. It has a score $\mu_i / \max_j \mu_j$ which is normalizing the number of dots to 1.

TSRPR3 has 3 rules, 3a and 3b.

Rule **3a** focuses on finding the notes with the highest pitch, and prefer to use it as the head note. 3a has a value of $\xi_i / \max_j \xi_j$.

Rule **3b** is the opposite of 3a, which is focusing on finding the note with the lowest pitch. It has a value of $1 - \xi_i / \max_j \xi_j$.

TSRPR4 focuses on finding the heads that are parallel in time-span reduction. It evaluates every starting note of the time-span, with the time-span size as interval. The similarity score is normalized to 1.

TSRPR8 focuses on all the starting notes i_{start}, and they will have a score of 1, and 0 if the note is not the beginning position of a time-span.

TSRPR9 is similar to TSRPR8, but it is looking for the ending notes of time-spans. It will have a score of 1 if it is the i_{end} note, and 0 if not.

To start generating the time-span tree, first we need to calculate the $B^{timepsan}$(i) of every note. $B^{timepsan}$(i) is the summation of $TSRPR_R$(i) times the weighting value of each rule, where $R \in \{1, 3a, 3b, 8, and\ 9\}$. Then we calculate $D^{timespan}$(i), which is the value of $B^{timepsan}$(i), except for all the notes that has score 1 on TSRPR4. They will have the value of $B^{timepsan}$(i) summing the summation of $B^{timepsan}$(j) times the weighting value of TSRPR4 if TSRPR4(i,j)=1.

Finally, by iterating through the hierarchy of the metrical analysis then grouping analysis, and choosing head notes at each division of time-span. Repeat until there are no more time-span to iterate, then the time-span tree is finished constructing.

2. Tree editing distance

Two tree structure can be compared to each other by calculating the tree editing distance between them.

The tree editing distance is a concept of adding up the cost for one tree to be transformed into another. The cost function being considered are usually the insertion of a node, the deletion of a node, and the substitution between two nodes.

GTTM transforms music into time-span tree, which is an insightful way of representing music in terms of music knowledge. In this project, the tree editing algorithm used for comparing time-span trees are from Zhang and Shasha's research[10]. The algorithm they have developed is one of the most efficient way of comparing two trees. In the algorithm, the trees that are being compared are not only targeted to binary tree. It is possible for one node to have more than two children. The nodes are expected to have values, either a string or a single character. The algorithm is very simple, and can be described with the following example. Suppose there are two trees, tree1 and tree2, presented as below. The tree editing tree between Tree 1 and Tree 2 is 4, with the following edits:

- Substituting node H with node B
- Substituting node L with node D
- Inserting node W to node A
- Deleting node E from node F

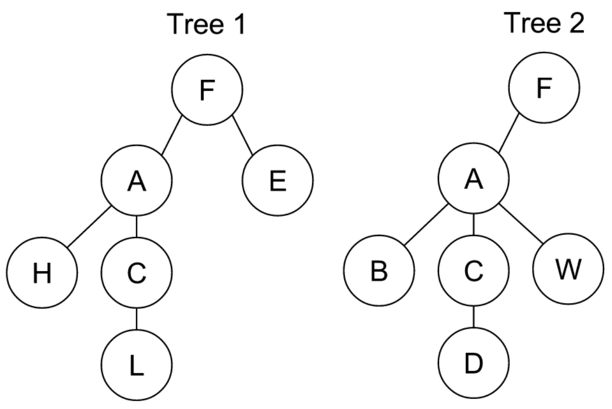


Figure 1, Tree 1 and tree 2 for tree editing distance example

III. TIME-SPAN TREE EDITING DISTANCE

Time-span tree is one of the results for *A generative theory of tonal music (GTTM),* and it has a few characteristics that makes tree editing distance a great method for comparing two music's time-span trees. First, the cost function used for edit distance calculating can be specified to fit time-span tree's need.

Second, the tree editing distance algorithm preserves the temporal variable of each time-span tree. Time-span trees are binary trees that every leaf is a music note, and those notes needs to be ordered from left to right to maintain the ordering of the notes of the original music.

Before coming up with the cost function for inserting, deleting, and substituting a node, we have to define the values that is going to be put into a node that represents a music note. In Time-span tree, every node should have a value, since every node in the tree is actually a music note. The elements that needs to be considered are basically the pitch and the duration of a music note. The expression of a note, like slur and accent, are not considered in this project for the sake of simplicity, but it can be included easily. Each node is represented with a string which can be divided into 2 parts. The first is the pitch, which is constructed through its step S_i, octave O_i,, and alter A_i for note i. Step considers only the step character of a note, and if it is flat or sharp, it will be taken care of by alter with – or + symbols. Alter leaves blank when its neither sharp or flat. O_i is an integer that represents the octave of the note. The second part of the string is the duration of a note D_i, which is a positive integer.

$$S_i \in \{C, D, E, F, G, A, B\}$$
$$O_i \in \mathbb{Z}$$
$$A_i \in \{-, +, blank\}$$
$$D_i \in \mathbb{Z}^+$$

For example, a note of center C that has duration of 1 will be have value of "C4[blank]1", and a note of flat low C of duration 4 will have value "C2-4", where the "-" represents the flat. This simple string encoding can easily be decoded back to the pitch and duration of a note.

Before comparing two time-span trees, we need to calculate the maximum duration of both music for normalizing the cost. This allows the cost function to come up with a score for duration $D^{cost}(i)$ for note i. However, for pitch, there's no way to normalize the cost while inserting or deleting a note since there's no context of how significant is the pitch being inserted or deleted is. Therefore, the cost for inserting $Cost^{insert}(i)$ and deleting $Cost^{delete}(i)$ is only considering $D_i / \max_j D_j$.

$$Cost^{insert}(i) = Cost^{delete}(i) = D^{cost}(i)$$

$$D^{cost}(i) = \frac{D_i}{\max_j D_j}$$

For cost of substituting node i with node k, the cost for pitch $P^{cost}(i,k)$ can be considered. ABS_i is the absolute value for pitch of note i. T^S is the numeric way of representing step, where C=0, D=1, E=2, …, B=6. T^A is the numeric way of representing alter, where flat is -1, sharp is 1, and 0 if there's no alter. The number 12 inside ABS_i represents the width of an octave, which is the quantity of all possible notes within one octave.

$$Cost^{substitute}(i,k) = D^{cost}(i,k) \times P^{cost}(i,k)$$

$$D^{cost}(i,k) = \frac{|D_i - D_k|}{\max_j D_j}$$

$$P^{cost}(i,k) = \frac{|ABS_i - ABS_k|}{\max_j ABS_j - \min_j ABS_j}$$

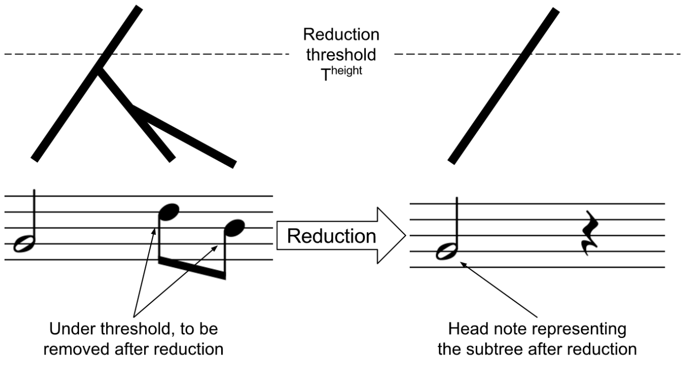
$$ABS_i = 12 \times O_i + T^s(S_i) + T^A(A_i)$$

After calculating all the costs from all the nodes, the final cost for two time-span tree x and y can be presented with $Distance(x,y)$, where the size of both trees is taken into account to show how significant was the change compared to the tree size. All the cost functions are symmetrical, meaning that the cost for editing tree x into tree y will be the same as editing tree y into tree x, the cost only needs to be calculated once, and $Distance(x,y) = Distance(y,x)$. In the process of calculating tree editing distance, i are the nodes to be inserted, j are the nodes to be deleted from the tree, and k are the nodes to be substituted with w. $Size(x)$ is the size of tree x.

$$Distance(x,y) = \frac{\left(\sum_i Cost^{insert}(i) + \sum_j Cost^{delete}(j) + \sum_{k,w} Cost^{insert}(k,w)\right)}{\max_{x,y}\left(Size(x), Size(y)\right)}$$

By iterating through every pair of time-span tree and obtaining the tree editing distance between them, we will have n by n result where n is the number of music.

One of the major characteristic of time-span tree is that in the process of constructing a time-span tree, nodes that has higher score will be determined as the head of the subtree. As a head, it can then be used as a preview of a subtree.



Reduction threshold T^{height}
Reduction
Under threshold, to be removed after reduction
Head note representing the subtree after reduction

Figure 2, head note will be representing a subtree

This characteristic can be utilized in tree editing distance by not always compare two trees, but compare two trees at a certain height. In this way, not all the notes will be represented, and only the head nodes height within the threshold height T_{height} will be presence while comparing time-span trees. To achieve this, a simple function that would return the time-span tree as usual, but removing all the notes that are too low in the time-span tree.

IV. Recommendation of Music

Multi-dimensional reduction to a 2D plane can be used for visualizing the distribution of individual data in terms of similarity. The result of tree edit distance calculation of time-span tree will give a distance matrix for every pair of music. With the matrix, we then can do Multidimensional scaling (MDS)[11] to transform the distance matrix into 2 dimensional values that represents each music in that matrix. By calculating the distance between two music, a similarity score can be obtained. By finding the points that are near of a point, those points can then be the candidate for recommendation.

Inherited the head node's characteristic from time-span tree, different reduction threshold T^{height} can be set for different level of resolution. When the threshold is set to only show a small number of notes, the result of MDS will cluster the music that are similar in the more abstract level. The similarity and recommendation of music will then be based on just a few head notes that describe the music. On the other hand, when the threshold is set to take all the music notes into account, the result of MDS will then be clustering the music that have more similarity in details.

The resolution of a time-span tree is related to the threshold for the tree height. The resolution is high when the threshold is large, and the resolution is low when the threshold is small. When T^{height} is set to -1, the threshold is completely neglected, and all of the nodes from every time-span tree are used to calculate the tree editing distance. When set to 0, only the tree's root is used to represent each time-span tree, which is the one note that can best summarize the music according to GTTM.

By changing T^{height}, the result of MDS would change dramatically, and it is un predicted of how the layout can be. T^{height} was implemented to the system that allow user to change the value. In most cases, clusters are what users should be looking for, since the more density a cluster has, the more similar the music of that cluster are when compared to all the other music outside the cluster. With the given dataset there are 5 classical composers, Ludwig van Beethoven, Frederic François Chopin, Wolfgang Amadeus Mozart, Franz Peter Schubert, and Pyotr Ilyich Tchaikovsky.

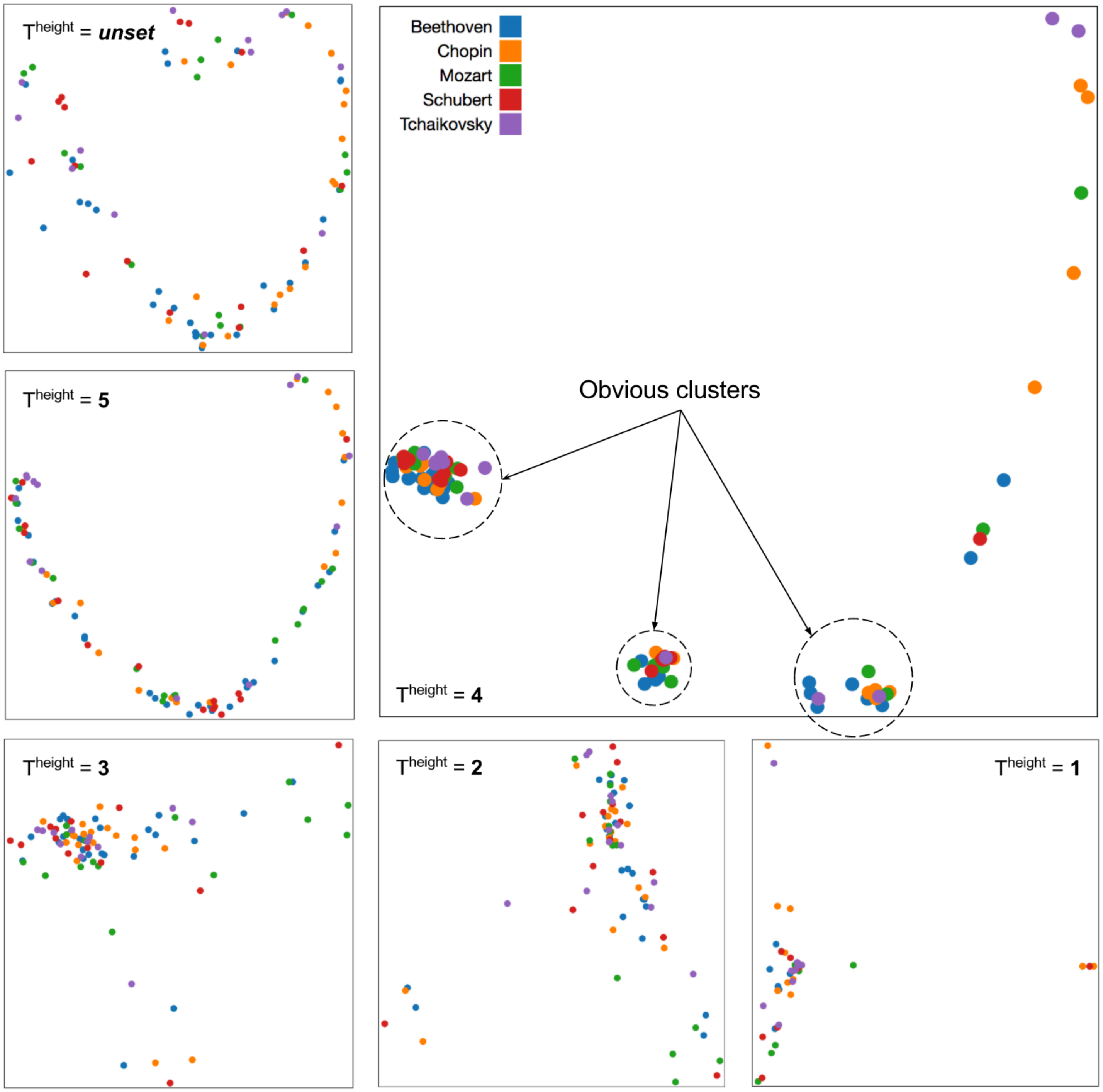


Figure 3, MDS results when the height threshold of time-span tree is set to different values. When the threshold is set to 4, obvious clusters can be quickly identified by user.

When the threshold is not set, the result of MDS doesn't show obvious clusters. It is still possible to define groups under this high resolution, but the points are just too scattered to be considered as a solid cluster.

When the threshold is set to a lower number like 5, the result of MDS still doesn't show obvious clusters.

When the threshold is set to 4, obvious clusters are easily shown in the result of MDS. A good explanation can be that the music that are clustered together have similar high level content similarity in terms of pitch and duration. The points can be roughly clustered into 4 groups, while the 4th group being very loose, and probably doesn't have much similarity in the points.

When the threshold is set to 3, 2, and 1, the result doesn't seem more promising than when the threshold is set to 4 since that no obvious clusters can be identified immediately through the MDS visualization.

The system also implemented selection feature. With a given music, when user see that music is included to one of the clusters, user may want to know more about this cluster. The selection can act as a navigation system, where the selection points will go through the process of MDS, and be plotted to 2D plane again, which is similar to zooming in. By zooming in onto the cluster that the selected music is in, it is more obvious that which points are closer to it even though all the music are considered similar enough when clustered together.

Instead of simply re-plot the cluster with the exactly same threshold, user can also change the height threshold used for calculating the time-span tree similarity.

A great case of use is when the cluster are formed at a low resolution, which is only when a few music notes are considered for every music, but user want to know how the MDS result change due to the change of height threshold. For example, when user want to find a recommendation for Schubert's *Ellens Gesang III Op.52-6 D.839*, and the music is within an obvious cluster at low height threshold setting.

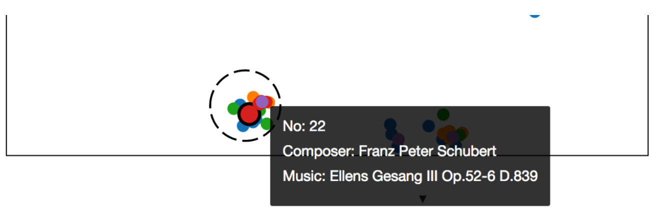

No: 22
Composer: Franz Peter Schubert
Music: Ellens Gesang III Op.52-6 D.839

Figure 4, Ellens Gesang III Op.52-6 D.839 being selected shown as circle with black border. It lays in an obvious cluster.

By selecting the points of that cluster, user can see a re-plotted MDS of that cluster under the same height threshold. At this point, user can see that the most similar music is Mozart's *Le nozze di Figaro, K.492, Act 2 "Voi che sapete"* and Beethoven's *Sonate Nr.3 C-dur op.2-3 1.Satz*.

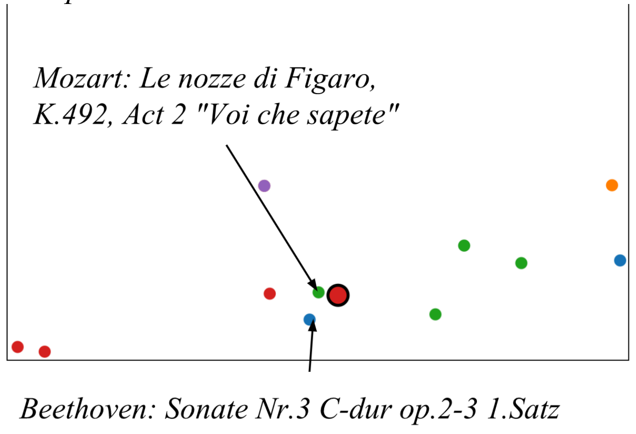


Figure 5, re-plot at the same height threshold shows clear recommendations.

However, when the height threshold is unset, the most similar music becomes Mozart's *Horn Concerto No.1 D major*.

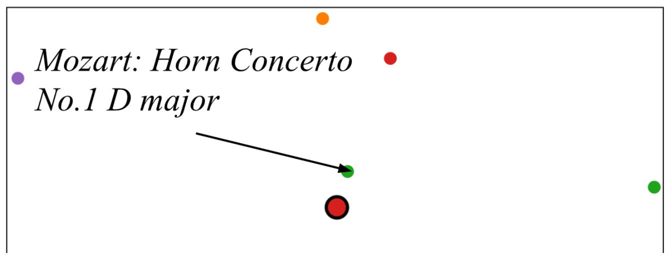


Figure 6, when the height threshold is unset at the second replot, recommendation changed due to similarity that is considering all the music notes.

This multilevel condition filtering mechanism allows user to find the music that are not only similar in high level, but also similar at the detailed music notes. If the user have not set the threshold at the very beginning, the most similar music would actually be Schubert's own music, *Winterreise D.911 Op.89 Der Lindenbaum* and *Winterreise D.911 Op.89 Frühlingstraum*. Without the ability to re-plot at different height threshold, the recommendations above can still be similar in detail music notes, but not in higher level where only small number of music notes are considered.

Through plotting the distance matrix from calculating tree editing distance of every time-span tree with a certain tree height threshold, and allowing user to replot at different height threshold with different subset of points, the MDS scatter plot of music enables user to find the best match for recommendation under different scope of resolution.

V. Pattern discovery

The result of time-span tree not only can be used for calculating the difference of each pair of music, itself is actually a well-structured tree that represents the music. At the level of constructing the time-span tree for a music, decision of branching is made to best show the musical characteristic of a music. Many rules are considered to determine the where the boundary should be for grouping.

With the combination of two kind of visualization, we have come up with a new visualization that represents a music and its time-span tree. The first part is a simple visualization of tree, from top starting with the root to bottom with the leaves. The leafs are aligned to the most bottom part to easily show how the leaves represents in the second part of the visualization. The tree visualization preserves time-span tree's temporal variable, where the notes are played from left-most leaf to the right-most. The second part is a table-like visualization, where the columns also represent the temporal data of the music, and the rows are the pitch of certain time. The row order is from top with the highest pitch of that music to the lowest pitch. This visualization can be widely seen in music related editing software, and is usually called "piano roll".

Through hovering the nodes inside a tree, user can easily identify the subtrees that are similar with the current subtree. The elements taken into account are the structure of that subtree, and the combination of music note duration that the subtree is representing. At each node, they are given a string that describe the subtree with the following simple algorithm. Inspired by the programming language's design of LISP, this is a simple way to represent a tree structure.

Algorithm 1 Create string for every nodes in a tree

```
1:  procedure treeString(A)
2:    if no children in A do
3:      A.string = A.duration
4:    else do
5:      A.string = "<"
                  + treeString (A.left)
                  + "|"
                  + treeString (A.right)
                  + ">"
6:      return A.string
7:  end procedure
```

Since visualization of pattern can be very useful when discovering repeated phrases of music notes, we also allow user to see two music side by side, and the highlight of a subtree can also be highlighted at another tree if there's a match. This way, it would be a lot easier to analysis two music that does share the same pattern, but using those patterns in different order.

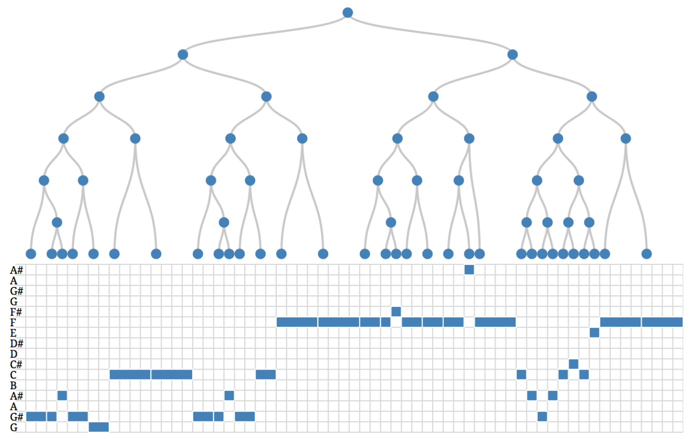


Figure 7, visualization of time-span tree and piano roll of Moments Musicaux, composed by Franz Peter Schubert.

Beside visualization, we have also allowed user to listen to the music that the current visualization is representing. It would be easier for user to know how a certain phrases sound like, and don't many to depend on their imaginary of how it should be like.

As an example, we can try to describe the pattern of *Moments Musicaux*, composed by Franz Peter Schubert. Through hover, from the root node to lower nodes, we can quickly see that the first half of the music is two parts of music phrases from the same pattern. The top-most node of the red subtree is where the mouse is located at. We can see that the orange subtree is similar to the red one.

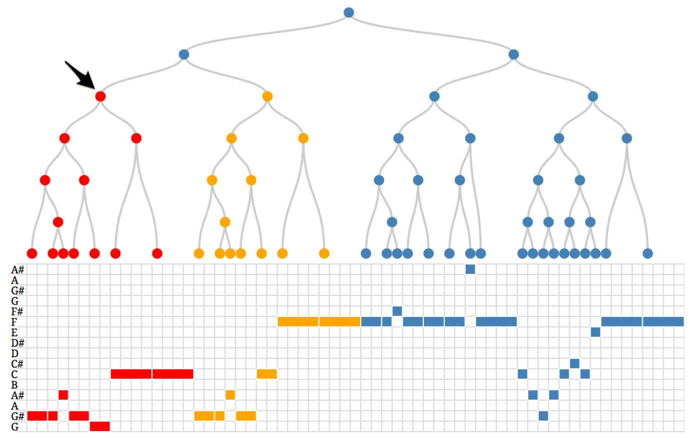


Figure 8, discovered the first half of the music is actually a repeated pattern.

Now we try to hover to the later part of the music, and no orange subtree has light up, meaning that there's no other similar pattern to the current mouse-hovered subtree.

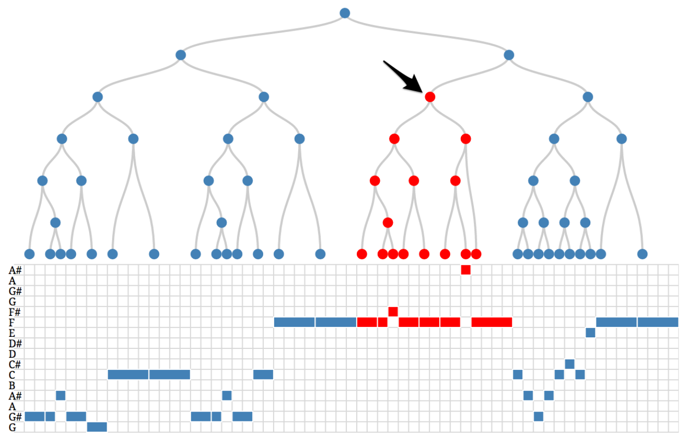


Figure 9, a music phrase that doesn't find any other phrases similiar to it.

Then we move down to the left child of the subtree, we can find that there's a patterned shared three times along the music.

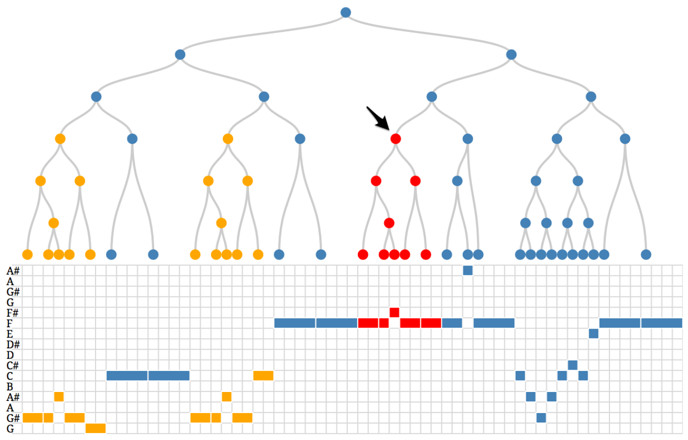


Figure 10, discovered a common pattern with smaller size.

Through this process, we can actually see that the music actually can be constructed with 4 phrases A, B, C and D, where A, B, and D are used repeatedly.

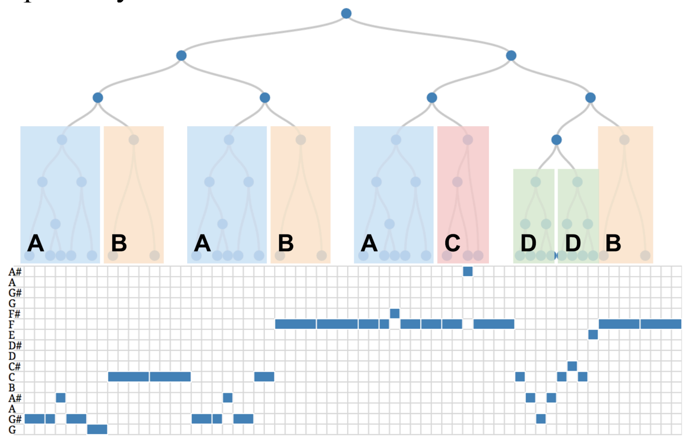


Figure 11, the overall patterns discovered are A, B, C, and D.

VI. Conclusion

In this project, by utilizing the results of GTTM analysis of music[4], [6]–[8], we obtained time-span tree. Such tree structure preserved the music knowledge and insight of how a music can be seen in tree structure, which reveals patterns, and the head notes which are rather important comparing to other notes inside the subtree. Through defining the cost function for tree editing distance calculation[10], we obtained a distance matrix of every music pairs. By reducing dimension to 2D with MDS[11], we can plot the music on a 2D plane, and discover similar music. Tree height threshold can be set by user depending on how high of resolution user want the time-span tree to be. The lower the resolution, the less notes will be used for cost calculation, meaning only the more important notes are considered. Allowing user to select points on the result of MDS, user can replot with any resolution that would affect the distance matrix's calculation. This way, user can find recommendations for a music with multiple conditions. We have also visualized the time-span tree combining with the visualization of the original music, and allow user to find patterns through highlighting the similar subtrees. By a side-by-side visualization of two music in their time-span tree form, common patterns of both music can be easily discovered.

The majority of information retrieval of music has been focusing on the metadata and related information of the music[3]. With the implementation of music theory like GTTM[8], content-based information can be obtained from music.

VII. References

[1] P. Isenberg, T. Isenberg, M. Sedlmair, J. Chen, and T. Moller, "Visualization as Seen through its Research Paper Keywords," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 771–780, Jan. 2017.

[2] "KeyVis." [Online]. Available: http://keyvis.org/. [Accessed: 09-Apr-2017].

[3] "Algorithmic Music Recommendations at Spotify." [Online]. Available: https://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify. [Accessed: 09-Apr-2017].

[4] F. Lerdahl and R. Jackendoff, *A generative theory of tonal music*. MIT Press, 1996.

[5] “Home - PubMed - NCBI.” [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed. [Accessed: 09-Apr-2017].

[6] “GTTM – Generative Theory of Tonal Music / Masatoshi Hamanaka.” [Online]. Available: http://gttm.jp/gttm/. [Accessed: 03-Apr-2017].

[7] M. Hamanaka, K. Hirata, and S. Tojo, “MUSICAL STRUCTURAL ANALYSIS DATABASE BASED ON GTTM.”

[8] K. Hirata, S. Tojo, and M. Hamanaka, “Implementing ‘A Generative Theory of Tonal Music,’” *J. New Music Res.*, vol. 35, no. 4, pp. 249–277, 2006.

[9] S. Koelsch, M. Rohrmeier, R. Torrecuso, and S. Jentschke, “Processing of hierarchical syntactic structure in music.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 110, no. 38, pp. 15443–8, Sep. 2013.

[10] K. Zhang and D. Shasha$, “SIMPLE FAST ALGORITHMS FOR THE EDITING DISTANCE BETWEEN TREES AND RELATED PROBLEMS*,” vol. 18, no. 6, pp. 1245–1262, 1989.

[11] J. B. Kruskal, “Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis,” *Psychometrika*, vol. 29, no. 1, pp. 1–27, Mar. 1964.