Clarinet: A Music Retrieval System

Kshitij Alwadhi
Dept. of Electrical Engineering
Indian Institute of Technology Delhi
2019EE10577

Rohan Sharma
Dept. of Electrical Engineering
Indian Institute of Technology Delhi
2019EE30121

Siddhant Sharma
Dept. of Electrical Engineering
Indian Institute of Technology Delhi
2019EE10531

Abstract

A MIDI based approach for music recognition is proposed and implemented in this paper. Our Clarinet music retrieval system is designed to search piano MIDI files with high recall and speed. We design a novel melody extraction algorithm that improves recall results by more than 10%. We also implement 3 algorithms for retrieval-two self designed (RSA Note and RSA Time), and a modified version of the Mongeau Sankoff Algorithm. Algorithms to achieve tempo and scale invariance are also discussed in this paper. The paper also contains detailed experimentation and benchmarks with four different metrics. Clarinet achieves recall scores of more than 94%. All of our code is publicly available here.

1 Introduction

Traditional Music recognition systems like Shazam and DejaVu rely on spectrogram analysis of samples to match songs. A major flaw in these algorithms is that they assume that the query given to the system has features (like sampling frequency) *precisely* similar to the studio recorded versions. Thus, matching song derivatives (like instrumental covers) is not possible.
To solve this problem, we need to find derivative invariant features. One such feature is the melody of the song. Derivatives of the same song are very likely to have the melody intact. Thus, if we could detect this melody, it would make music recognition more general by encompassing song derivatives. This project aims to tackle this problem by developing a MIDI based retrieval system that relies on monophonic melody features. The system is tested and evaluated on the Maestro Dataset [5] which is a dataset of Piano songs.

Why MIDI? MIDI files record the audio's notes at any given time, giving a standardized interpretation of the song. A melody extracted as a midi can thus we read as sheet music. Much of music theory (like scale and tempo invariance) depends on standardised time signatures and note values. Hence, MIDI helps us understand and deal with purely musical content (as opposed to audio signal content)

The following section describes our problem statement. **Section 3** provides an in depth discussion of the Clarinet model. **Section 4** discusses Experiments and evaluation of results. In **Section 5** we summarise our results. **Section 6** details future work that can be done in this field and **Section 7** talks about possible applications of our work.

1.1 Contributions

In this paper, we state many results of independent importance and also tie the concepts together to create a end-to-end robust and fast music retrieval system. The following is a list of the results:

1. **Melody Extractor**: Modified a state of the art melody extractor [13, 14], and improved it substantially both for independent use and for music retrieval.
2. **Similarity Computation**: Created RSA Note and RSA Time algorithms for music retrieval and modified a pre-existing technique Mongeau-Sankoff [2, 8] to make it more general.
3. **Sliding Window**: Ideated and implemented a sliding window concept to match query audios with audios of larger duration, that allowed for more generalisation of pre-existing algorithms.
4. **Evaluation Techniques**: Defined a metric called **Margin of Discrimination** that is the difference in confidence (similarity) scores between the target document and the next ranked document. This metric can be independently used in many applications.

2 Problem Statement

This paper contains many independent algorithms that can be used for various use cases. Hence, we address various problem statements and requirements that are listed below:

1. A need for a more **custom and robust melody extractor**, since the current state of the art method is too restrictive and generates unstable[1] melodies.
2. A **fast and accurate music retrieval system** that works when queries are of varying length.
3. A methodology to ensure data is **tempo and scale invariant** to ensure generality of data.

While we successfully tackle all the above prompts, we also touch upon other related ideas that have either been detailed in the paper itself or left as future work in **Section 6**.

3 Clarinet Model

Note that the Clarinet Model takes an input of a MIDI file. However, we also have added methodologies to convert MP3/WAV audio files to a Clarinet compatible format and back [16].

Overview Clarinet, after inputting the MIDI file, clips the audio if required and then processes it[2]. After the processing, the **melody is extracted**. For this, Clarinet provides two methods namely Skyline [1, 9, 13, 14] and the novel Modified Skyline. The following section contains the details on both the methods and compares them. Once we obtain the melody of the query, we can finally use it for **retrieving** the actual audio (which was pre-processed to contain the melodies as well). Clarinet provides three methods for the same, one of which is a modified version of the state of the art methods and the other two are novel algorithm dubbed RSA Time Algorithm and RSA Note Algorithm. All three methods use distinctly different ideas and have been detailed in subsequent sections. The RSA algorithms (RSA Time and RSA Note) are built on the principle of sliding windows of time and notes respectively and the Mongeau-Sankoff algorithm [2, 8] is based on edit distance changes which is an extension of Levenshtein Distance.

[1] Stability as defined in **Definition 3.1**

[2] Audio is processed for the usage of Mongeau-Sankoff Algorithm and its derivatives only

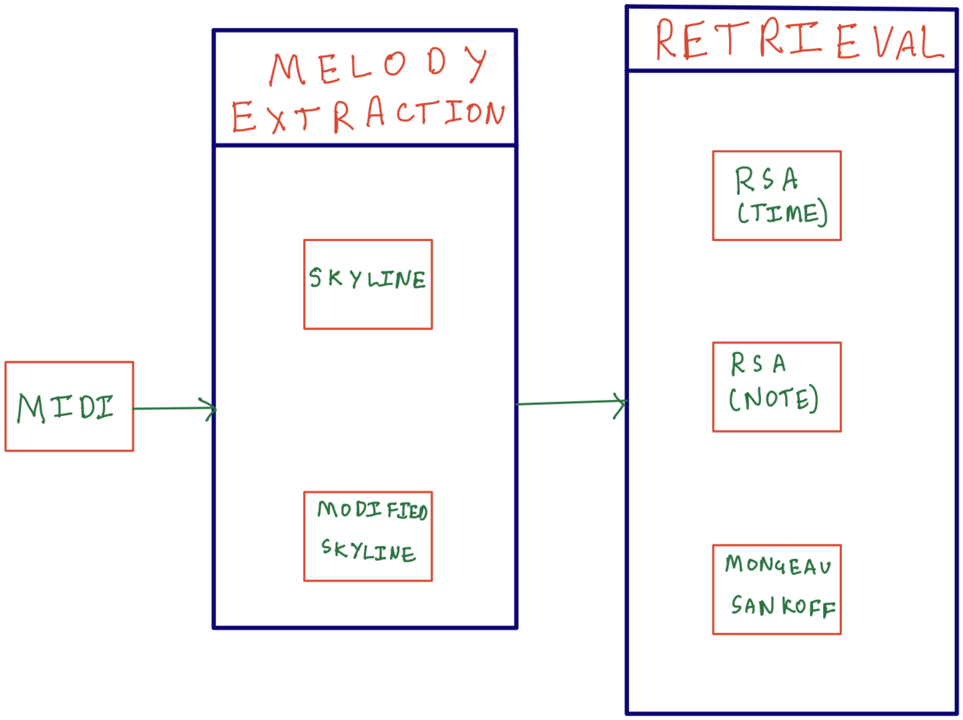


Figure 1: Pipeline of Clarinet

3.1 Melody Extraction

Raw MIDI files contain a lot of information of the audio. Extraction of the melody from the MIDI file is encouraged due to various reasons that are listed below:

1. **Monophonic Audio** - All retrieval models detailed below in **Section 3.4.1-3.4.3** require the audio to be monophonic at all times due to the representation as given in **Section 3.2**.
2. **Information Extraction** - The melodic element of the song is likely to be the same irrespective of the song derivative. Thus a majority of the relevant information of a song lies in its melody [7, 10].
3. **Noise Reduction** - Extracting just the melody allows us to ignore the noise present in the audio file.
4. **Data Compression** - Keeping only the melody of a song allows us to space taken to store our data.
5. **Faster Retrieval** - Pruning the melody to be more precise also gives us the benefit of a faster solution.

Let us now outline the current state of the art melody extraction algorithm, and the flaws with it in **Section 3.1.1**. After that in **Section 3.1.2**, we will detail the modified algorithm that takes care of the two prominent issues with the original skyline algorithm.

3.1.1 Skyline Algorithm

The **skyline algorithm** [1, 9, 13, 14] primarily works on the principal that the note with the **highest pitch constitutes the melody**. This is based on the assumption that the human ears tend to pick up on the higher frequencies and hence skyline uses these notes as the most prominent ones for the melody. The proposed algorithm is detailed below.

Algorithm 1: Skyline

```
Data: Raw Notes
Result: Melody Extracted Notes
$notes \leftarrow rawNotes$;
$skylineNotes \leftarrow []$;
for $note \in notes$ do
    $sameTimeNote$ = notes with same start as note;
    if $note \notin$ importantNotes && note = $\arg\max\{pitch(sameTimeNote)\}$ then
        $nextNote \leftarrow$ next Note with different start time;
        $note.end = \min(note.end, nextNote.start)$;
        note appended to skylineNotes;
    end
end
```

And here we see that the skyline algorithm **loses on important information** by directly truncating and discarding notes without saving them for later. It also is **very restrictive** on the methodology of deciding the notes relevant to the melody of a song. We develop a modified version of the skyline algorithm to address these short comings in the following section.

3.1.2 Modified Skyline Algorithm

The **modified version of skyline** perfects the original algorithm, while also adding **additional functionality** that replaces the current state of the art technology with ours. Working under the theory that only one note plays at any given time in the melody of the song, the Modified Skyline algorithm allows for a custom function that takes in the important note, and returns a real number that can be used to **compare between the importance of various notes**. Our preliminary implementation contains various criteria functions dependent on the pitch and velocity but this can be scaled up to any information the MIDI file contains. Our implementation also deals with the unfortunate event where **notes are prematurely truncated in the original skyline algorithm** by saving the notes till their true end time.

Algorithm 2: Modified Skyline

```
Data: Raw Notes
Result: Melody Extracted Notes
$criteria \leftarrow f : (note \rightarrow r \in \mathbb{R})$ ;      /* $r \in \mathbb{R}$ is the importance score of note */
$notes \leftarrow rawNotes$;
$importantNotes \leftarrow []$;
$skylineNotes \leftarrow []$;
for $note \in notes$ do
    $sameTimeNote$ = notes with same start as note;
    if $note \notin$ importantNotes && note = $\arg\max\{criteria(sameTimeNote)\}$ then
        note appended to importantNotes;
    end
end
for $note \in importantNotes$ do
    $currentNotes \leftarrow$ All notes playing at note.start time;
    $bestNote \leftarrow \arg\max criteria(currentNotes)$;
    $bestNote.start \leftarrow note.start$;
    $bestNote.end \leftarrow futureBestNote.start$;
    bestNote appended to skylineNotes;
end
```

As can be seen from the above formulation of the two algorithms, our modification deals with two pertinent issues with the original skyline algorithm by implementing the following:

1. Flexibility of defining one's own criteria for gauging importance of a note for comparison.
2. Not losing information by prematurely truncating and deleting notes from hash.

Another unexpected improvement was observed; the melody extracted was much more stable for the modified version of skyline when compared to the original version.

3.1.3 MIDI Representation

Definition 3.1 (**Stability of a melody**). *It is defined as the variation of the time a note plays compared to the average time that all notes play. Additionally, the lack of perturbations in the extracted melody's MIDI representation is also a measure of* ***stability of a melody****.*

Let us first look at the MIDI representation [11] of the original audio.

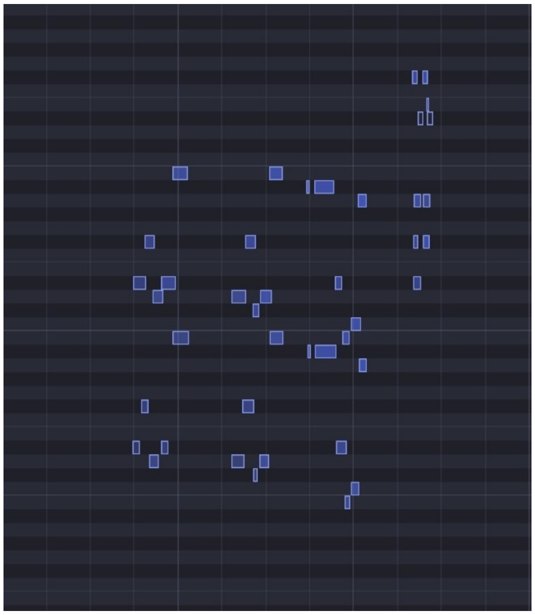
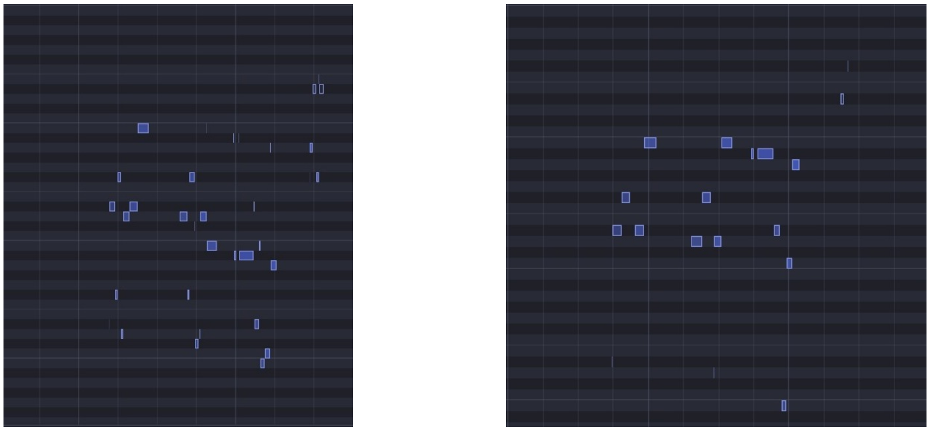


Figure 2: MIDI Representation of original audio

This representation contains the note being played plotted against the time stamp. We can see that at the same time, we sometimes have **multiple notes playing**. As discussed above, this is unnecessary and we need to extract the melody.
Now let us use the two melody extraction techniques and compare the results between the two.



(a) Skyline extracted melody

(b) Modified skyline extracted melody

Figure 3: MIDI Representation on extracted melodies

We see that the melody generated by original skyline algorithm (left) is very messy and contains anomalous notes being played as well. Clearly, this is quite poor in quality.
We can also see how much cleaner (and stable, as defined by **Definition 3.1**) this modified skyline algorithm (right) really is, by observing the left representation against the right one.

3.2 Text Representation and Fuzzy Matching

The approach we have taken involves representing the **melody midi file as a string** and using **sequential matching algorithms** to retrieve similar results. The simplest way to convert midi files to text is to ascertain the **notes** played and apply a **boolean** matching algorithm.

3.2.1 Naive Boolean Model

The algorithm is as follows

1. **Text Representation**-Convert document and query MiDi file to string
2. **Boolean Matching**-Check if the query string is a **substring** of the document

Example-Suppose the Document and query midi files are converted to text and obtained as

$$D = ABCBGC$$

$$Q = CBG$$

Since $Q \in D$, the system returns a **match**.

Since the algorithm relies on substring matching, it is extremely **fast**. However, this method is too **prone to errors**. A slight change in the query notes from document notes will cause the system to fail.

Example-Suppose the Document and query midi files are converted to text and obtained as

$$D = ABCBGC$$

$$Q = ACBG$$

Here the query contains another note **A**. This could be because the user played the note incorrectly or the midi conversion tool detected the wrong pitch. The system fails to find a match here since $Q \notin D$

The algorithm provides no ranking of similarity between the document and query. Thus, we don't get ranked similarity results even when other songs in the database might be similar. We need an algorithm more resilient to errors and one that provides similarity scores.

3.3 Fuzzy Logic Model

As discussed above, we need a string matching algorithm that doesn't give 0 scores for queries that don't *exactly* match the document. Levenshtein distance (a kind of Dynamic Time Warping) has been shown empirically to be the best distance measure for string editing. It measures how far two strings are using an operation set(insertion, deletion and substitution)

Definition 3.2 (**Levenshtein Distance**). *The Levenshtein (string editing) Distance between two sequences is the minimal number of substitutions, insertions, and deletions needed to transform from one to the other. Formally, the Levenshtein distance between the prefixes of length i and j of sequences S and T, respectively, is:*

$$lev_{s,t}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ \\ min \begin{cases} \text{lev}_{s,t}(i-1,j)+1 \\ \text{lev}_{s,t}(i,j-1)+1 \\ \text{lev}_{s,t}(i-1,j)+(S_i \neq T_j) \end{cases} & \text{otherwise} \end{cases}$$

Here we have assumed an ***unweighted*** *Levenshtein Distance (Weights may also be introduced according the value of either of the edit operations).*

The above Levenshtein distance suffers from a major drawback. If the query and document are of different lengths, the edit distance becomes very large. Due to this large edit distance, **query variations will have no effect**-leading to poor discrimination. Since the queries we have chosen are only 10% of the document length, the results will thus be very inaccurate.

3.4 Similarity Calculation

Based on the above discussion we arrive at the following requirements for a "good" model:

1. **Robustness**-The model should be able to handle noisy inputs. This includes queries having spurious additional notes ("ABDF" instead of "BDF"), substituted notes ("BGF" instead of "BDF") and fewer notes ("BF" instead of "BDF")
2. **Discrimination**-The model should be able to discriminate between documents effectively. In other words, the similarity scores of documents shouldn't be too close to each other. We have designed the Margin of Discrimination metric in Section 5 as a measure of this property.

3.4.1 RSA (Time Based)

RSA is a **time based sequential retrieval** algorithm inspired by dynamic time warping. It assumes the approximate query time length is known beforehand and performs chunk based matching for subsets of the document. A detailed description of the algorithm is given below:

Suppose the **approximate** query length is T_0. For each query :

1. **Window creation**-Create a window of length T_0 over D, ie. pick the first T_0 seconds of D. Call this window W. Thus, we have $T_W = T_0$
2. **Distance Computation**-Since $T_Q \approx T_0$ (from our assumption), we have $T_Q \approx T_W$. We can thus safely find the edit distance between Q and D. Store this as $L_{Q,D}$ (standing for Levenshtein distance).
3. **Translation**-Translate the window by stride length (T_S) and compute the distance between W_{new} and Q (since we still have $T_Q \approx T_{W new}$)
4. **Compare distance**-Update $L_{Q,D}$ if W_{new} has smaller distance.
5. **Repeat**-Repeat step 3,4 till the document has no more windows

From the above edit distance we can calculate the similarity as

$$\boxed{sim(Q,D) = 1 - \frac{L_{Q,D}}{len(Q)}}$$

The similarity between the document and the query is thus the same as the best similarity between the set ($S = S_{T_0}$) of document substrings of length T_0.

$$sim(Q,D) = max(sim(Q,W))$$
$$W \in S$$

Example

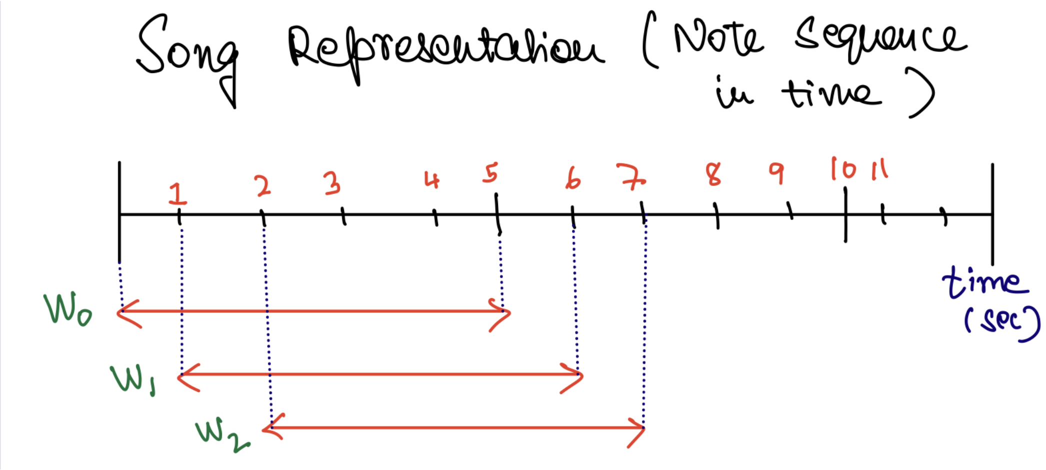


Figure 4: Working of RSA(Time)

In the above diagram we have **stride length** $T_S = 1s$ and **Approximate Query Length** $T_0 = 5s$ Using this we can calculate the window size as

$$T_W = T_0 = 5s$$

Thus our window will be of 5s and will move 1s per iteration. The value of stride length is a hyperparameter. *Lower stride lengths give higher accuracy but at the cost of time.*

3.4.2 RSA (Note Based)

Instead of looking at windows in the time domain, we can look at note windows. Suppose the **approximate** query length (in notes) is N_Q. For each query :

1. **Window creation**-Create a window of length N_Q over D, ie. pick the first N_Q notes of D. Call this window W. Thus, we have $N_W = N_Q$. *Note that here the window size depends on the query rather than being query invariant like the previous algorithm.*
2. **Distance Computation**-Find the edit distance between Q and W. Store this as $L_{Q,D}$ (standing for Levenshtein distance).
3. **Translation**-Translate the window by stride length (N_S) and compute the distance between W_{new} and Q
4. **Compare distance**-Update $L_{Q,D}$ if W_{new} has smaller distance.
5. **Repeat**-Repeat step 3,4 till the document has no more windows

From the above edit distance we can calculate the similarity as

$$\boxed{sim(Q, D) = 1 - \frac{L_{Q,D}}{len(Q)}}$$

The similarity between the document and the query is thus the same as the best similarity between the set ($S = S_{T_0}$) of document substrings of length N_Q (if $N_S = 1$).

$$sim(Q, D) = max(sim(Q, W))$$

$$W \in S$$

Example

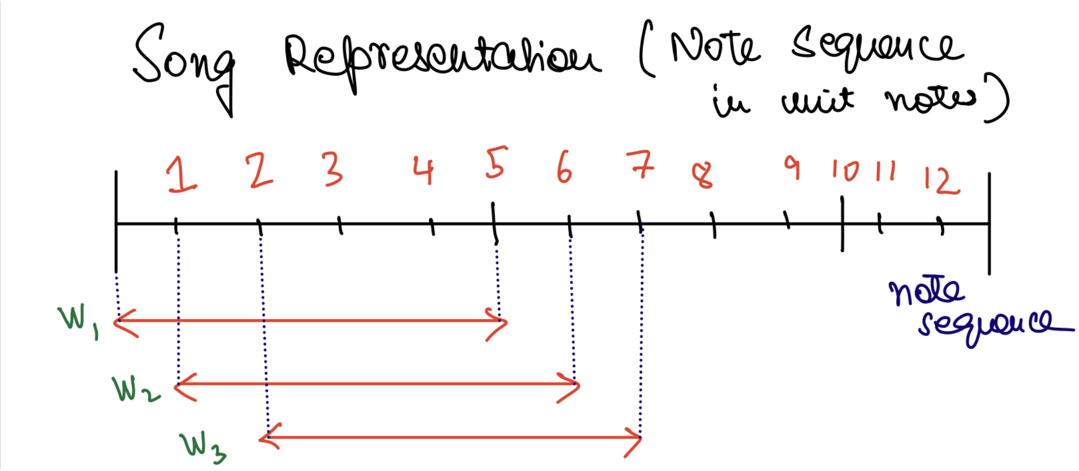


Figure 5: Working of RSA(Note)

In the above diagram we have **stride length** $N_S = 1$ and **Query Length** $N_Q = 5$
Using this we can calculate the window size as

$$N_W = N_Q = 5$$

Thus our window will be of 5 **notes** and will move 1 **note** per iteration. The value of stride length is a hyperparameter. *Again, Lower stride lengths give higher accuracy but at the cost of time*. For a given Query Q we can calculate the note window size as

$$N_W = len(Q)$$

For reasonable accuracy, the stride length can then be chosen from

$$N_S \in [0, N_W]$$

3.4.3 Mongeau Sankoff Retrieval

The **Mongeau-Sankoff** algorithm is a retrieval algorithm designed specifically for **Music Retrieval**. It improves on the traditional unweighted Levenshtein Distance by adding pitch based substitution weights and accounts for duration of notes(which the above algorithms did not do). The Mongeau-Sankoff Distance between two midi files is given by

$$mss, t(i,j) = min \begin{cases} \mathrm{ms}_{s,t}(i-1, j-1) & \text{if } \alpha_i = \beta_j \text{ (match)} \\ \mathrm{ms}_{s,t}(i-1, j-1) + \delta(\alpha_i \to \beta_j) & \text{(substitution)} \\ \mathrm{ms}_{s,t}(i-1, j) + \delta(\alpha_i \to \epsilon) & \text{(deletion)} \\ \mathrm{ms}_{s,t}(i, j-1) + \delta(\epsilon \to \beta_j) & \text{(insertion)} \\ \mathrm{ms}_{s,t}(i-k, j-1) + \delta(\alpha_{i-k+1} \dots \alpha_i \to \beta_j) & \text{(consolidation)} \\ \mathrm{ms}_{s,t}(i-1, j-k) + \delta(\alpha_i \to \beta_{b-k+1} \dots \beta_j) & \text{(fragmentation)} \end{cases}$$

Here it can be seen two new operations **Consolidation** and **Fragmentation** are introduced. A summary of these operations is provided below. For more information refer to [8] or [2]

Consolidation This operation compresses a few characters into a single one. For example, two 16th notes can be clubbed together to give an 8th note.

Fragmentation This operation is the opposite of consolidation. It involves splitting a larger note quantity into smaller units. compresses a few characters into a single one. For example, two 16th notes can be clubbed together to give an 8th note.

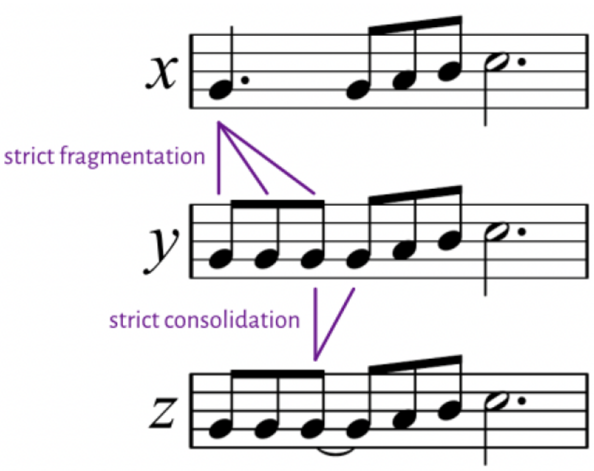


Figure 6: Fragmentation and Consolidation [2]-In the picture given above the **dotted note** is equivalent to **three notes**. Fragmentation involves splitting that note into three notes. Consolidation involves combining two notes(the line between two tied notes).

Assumptions and Issues

By construction the Mongeau-Sankoff algorithm relies on certain features being known beforehand.

1. **Tempo**-The Mongeau-Sankoff algorithm requires both **Note values** *and* the corresponding **Note Durations**. Here note durations are in **music units** (16th notes,8th notes etc.). This requires the tempo of the song to be known. We assume 16th notes as the basic unit. Note duration can then be converted from seconds according to the formula below

   $$\boxed{Dur(N) = \frac{4 * Tempo}{15} \cdot (Dur_s(N))} \tag{1}$$

   where $Dur(N)$ is the note duration in music units, $Dur_s(N)$ is the note duration in seconds, Tempo is the standard tempo value in beats per minute.

   While accounting for Note Durations is a good feature on paper, the issue is that the algorithm deals in music units. This means the tempo of every song needs to be known. This is not possible for most data since the tempo values store in midi files are often default values (Tempo=120). We found a workaround this issue by computing the tempo of each song through a Tempo detection algorithm([12]). Of course, this value is only an approximation and will thus lead to poor results.

2. **Scale Invariance**- In most cases songs are written in a certain Key (eg. Cmaj, Cmin etc.) with its corresponding scale. We can transpose from one scale to another (major→major, minor→minor) while communicating the same musical idea (melody). Thus, it is possible the input midi might be in a different scale compared to our database and will lead to poor matches. This is why we transpose all major scales to Cmaj and all minor scales to Cmin. Thus, the same song can be recognised even if it is presented in a different scale.

   Again, this seems like a good idea-and is something we thought of independently as well. However, this runs into major problems. When we transpose data into one scale (or equivalently, into 12 notes) we end up with a **loss of information**. Given a large enough database, the number of collisions(having same or very similar note representations) will increase substantially. We analyse these results in our experimentation section

3. **Length Similarity** Mongeau Sankoff assumes that the length of both strings are of the same length. This is combated by using sliding windows as discussed in the two RSA algorithms above.

3.5 Extensions to The Model

3.5.1 V.S.M. Approach

Another approach to retrieval could be using a modified Vector Space Model.Here the vector We can create a vector space model where the indices will be decided by the Note instead of the words in vocabulary and the weights associated with each index will be a basis function of the variables time duration and the velocity of the Note,

$$V[j] = \phi(vel_j, duration_j)$$

$$Sim_{cosine}(m, q) = \frac{\vec{m}.\vec{q}}{\vec{m}\vec{q}}$$

here, j represents the Note index, vel represents the velocity of the Note and duration represents the **total duration** the note is played in the song. This is equivalent to the **term frequency** parameter in VSM for text documents.

The pitch can lie between (-8192, 8191) so we will have a 2*8192 dimensional vector for each song snippet representation with the weights corresponding to some basis function of the velocity and the time duration while that pitch was on during the complete audio.

This process has a major con as it disregards the order of occurrence of the notes in the given 5 second clips, hence it can't be directly used for retrieval however, it **can be used for trimming down our documents to search** as VSM based similarity can be computed quickly and if it is above some threshold, only then we look into the edit distance similarity computation.

3.5.2 Extension to Dejavu

The approach followed in methods like Dejavu [15] is that they create a spectrogram of the original song by using FFT over small windows. Once they have spectrogram, they proceed to the next step of finding peeks (local maxima of amplitudes) in that spectrogram. Now, the data left to store is the position of these maximas (only frequency and time required). These discrete (time,frequency) pairs are in theory resistant to noise. To create these fingerprints [3], they store these peaks parametrized by its neighbours (time difference between nearest k peaks) by passing this information to a hash function and storing the output (SHA1). The value of k (termed as Fan Value), is a hyper-parameter and controls how many peaks should a particular peak be associated with before passing to the hash function. A higher fan value will lead to more number of fingerprints but better accuracy.

Dejavu suffers from a problem where if we change the speed of playing the notes in the query, then the system wont be able to recognize this as its time dependent. Our Text-based similarity approach should theoretically outperform Dejavu on the basis of metrics such as Recall etc.

4 Experiments

We evaluate the performance of Clarinet over the **Maestro** Dataset [5] with the metrics as defined.

Definition 4.1 (**Recall@K**). *The definition of Recall@K is as one would assume intuitively. It is the recall of the model assuming the correct audio (or document) file is present in the top K when similarity scores using a certain method are sorted in descending order.*

Definition 4.2 (**Mean Reciprocal Rank, MRR**). *Mean reciprocal rank of the desired document over the queries. **MRR only cares about the single highest-ranked relevant item**. Higher ranks are penalised by decreasing MRR.*

$$MRR = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{rank_q}$$

where $rank_q$ is the rank of the document from where the query was sampled.

Definition 4.3 (**Margin of Discrimination(MD)**). *Difference between similarity scores of the desired document with the next document. Measures how well the system could discriminate between the document from where the query was sampled and other documents.*

Definition 4.4 (**Time Taken**). *The average time taken per query is recorded and compared.*

4.1 Dataset

The original dataset we chose (MAESTRO dataset) had 1287 songs with on average every song being roughly 20 minutes in length. These were originally recordings from ten years of International Piano-e-Competition.

Documents We generate the documents by clipping the original dataset into **20 seconds** each from the beginning. The reason behind doing so was so as to increase the retrieval time as the songs in this dataset were huge(10 minutes).

Queries As there weren't any queries available on the internet for this task, we created the queries ourselves by running a script which randomly selects 40 songs from our dataset and then randomly clips **5 seconds** from anywhere in between. The start time need not be in multiple of seconds and can start from anywhere.

4.2 Melody Creation

Creating the melody from the MIDI audio file was done via the skyline algorithm and its modified version. Both the methods resulted in a MIDI file that we could use our retrieval models on, with the modified version doing substantially better as seen from the results of the experimentation as seen below which contains the MIDI representation of the files.

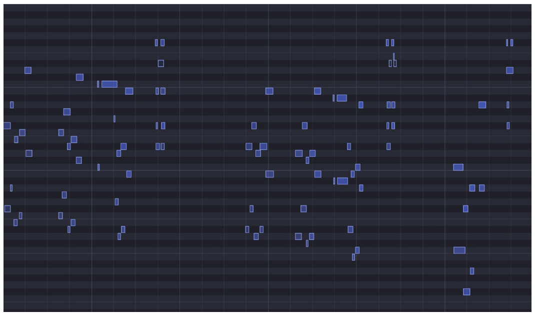


Figure 7: MIDI Representation of original audio

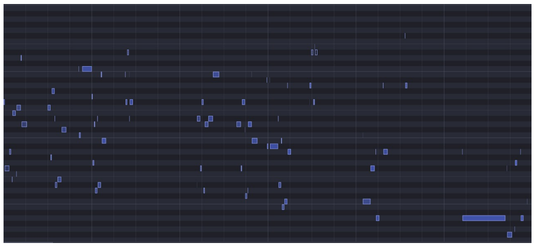


Figure 8: MIDI Representation of skyline extracted melody

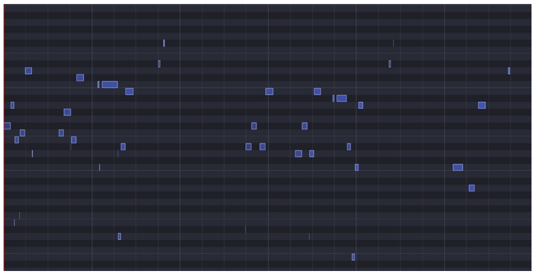


Figure 9: MIDI Representation of modified skyline extracted melody

Hence we can see that the modified skyline algorithm is cleaner and stable[3]. While we have not shown any numeric metric for comparison here (we do so in the next section), it is clear from the data representation that our modified algorithm surpasses the existing state of the art method for extracting melodies by a large margin.

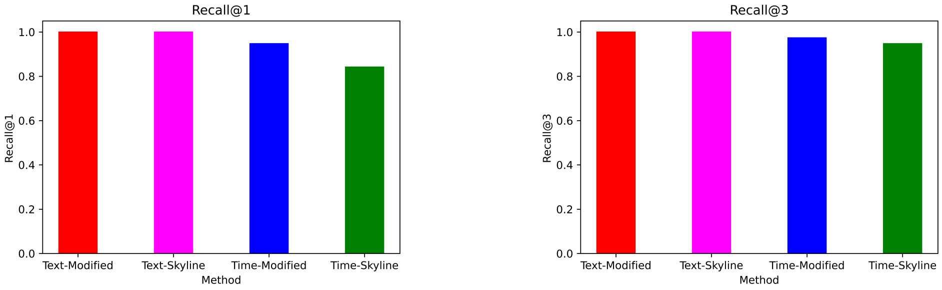
4.3 Retrieval Evaluation

We will now utilise this section to showcase our results and the improvements on existing state of the art methods utilising our own modifications or novel methods.
The first metric we will be showcasing is Recall@K for $K = \{1, 3, 5, 10\}$.

4.3.1 Recall@K

Similarity Method	Melody Extractor	Processed	Recall@1	Recall@3	Recall@5	Recall@10
RSA Note	Modified	Unprocessed	1.0	1.0	1.0	1.0
RSA Note	Skyline	Unprocessed	1.0	1.0	1.0	1.0
RSA Time	Modified	Unprocessed	0.947	0.973	0.9736	1.0
RSA Time	Skyline	Unprocessed	0.842	0.947	0.973	0.97368
RSA Note	Skyline	Processed	0.526	0.657	0.684	0.6842
RSA Note	Modified	Processed	0.473	0.60	0.631	0.657
RSA Time	Modified	Processed	0.0526	0.07	0.1578	0.236
RSA Time	Skyline	Processed	0.0526	0.157	0.210	0.236

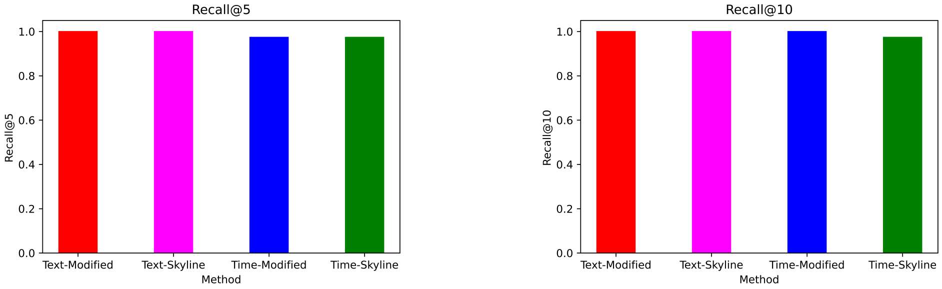
Table 1: Recall Comparison



(a) Recall@1

(b) Recall@3

Figure 10: Recall@K for K = 1, 3



(a) Recall@5

(b) Recall@10

Figure 11: Recall@K for K = 5, 10

[3] as defined by **Definition 3.1**

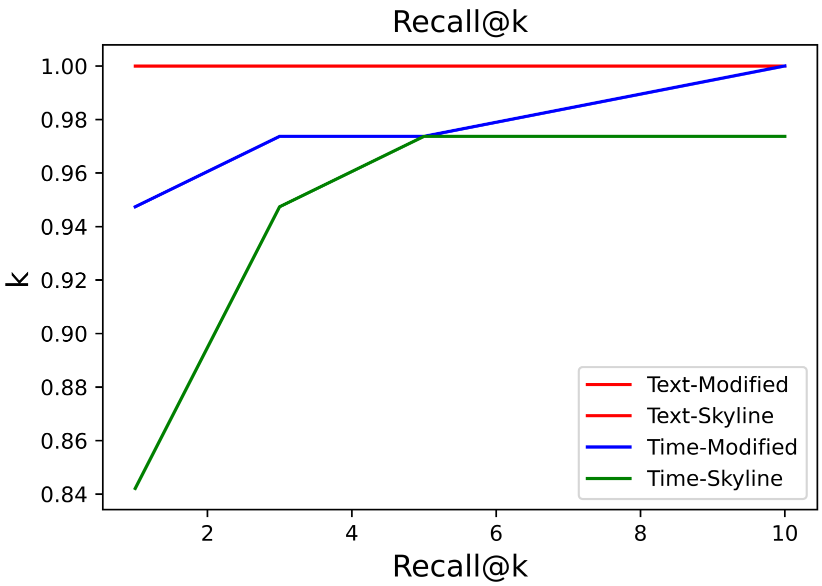


Figure 12: Recall@K vs K

From these results, we can make the following observations:

1. **RSA Text based methods reach 100% recall@1**, i.e. always succeed in predicting the correct result at the first position. This happens because it matches the query and document by looping through windows of length = query length.
2. The **Modified Skyline method outperforms** the classical Skyline method in both methods.
3. The **RSA Time** method also has a decent recall@1 and mostly achieves **100% recall@3**, i.e. the correct result lies in the first 3 documents always. (RSA Time - Skyline).
4. **Effect of Processing**-Processing increases time of retrieval substantially but leads to **unacceptablly bad results**. This is due to low dimensional data. It can be seen using **RSA Time** on Processed queries led to results being randomised. This is because RSA Time is susceptible to noise and Processing queries makes our model **robustness poor**.

For the analysis below we exclude Processed queries due to very poor retrieval quality.

4.3.2 Mean Reciprocal Rank (MRR)

Similarity Method	Melody Extractor	Mean Reciprocal Rank (MRR)
RSA Note	Modified	1.0
RSA Note	Skyline	1.0
RSA Time	Modified	0.964
RSA Time	Skyline	0.900

Table 2: Mean Reciprocal Rank Comparison

The mean reciprocal rank was found for all combinations. This metric has similar results as recall. RSA note performed better than RSA Time and Modified Skyline performed better than traditional Skyline.

4.3.3 Margin of Discrimination

Similarity Method	Melody Extractor	Processed	Recall@1	Margin of Discrimination (MD)
RSA Note	Modified	Unprocessed	1.0	28.5%
RSA Note	Skyline	Unprocessed	1.0	29.09%
RSA Time	Modified	Unprocessed	0.947	19.10%
RSA Time	Skyline	Unprocessed	0.842	20.27%

Table 3: Margin of Discrimination Comparison

Margin of Discrimination is the difference in confidence scores(similarity) between the similarity made for the target document and the similarity of the document just after the target document in ranking. These scores also normalized. This is an indicative of how **confident the model is in discriminating** its prediction from the other documents.

We observe that **RSA Text based approaches get the highest Margin of Discrimination** followed by the RSA Time approach. The choice of melody extraction algorithm seems to be playing little role here.

4.3.4 Time Taken

Similarity Method	Melody Extractor	Time per query (sec)
RSA Time	Modified	29.17
RSA Time	Skyline	62.91
RSA Note	Modified	183.86
RSA Note	Skyline	572.43

Table 4: Time per query

We observe that there's a **trade-off between recall scores and the time taken** to answer every query. Although RSA Text approach gets 100% recall@1, it has a major drawback of **increased retrieval time**. RSA Time approach still gets very decent results with significantly lower retrieval time.

Moreover, we also see that **Modified Skyline approach outperforms the classical Skyline** approach in terms of time taken per query as well.

5 Conclusion

This paper contains an entire pipeline for building a robust music retrieval system, outlining various state of the art methods for melody extraction and similarity functions, our modifications of them and three novel algorithms in the same domain.
The novel algorithms are listed below:

- **Modified skyline algorithm**: It clearly outperformed the original skyline algorithm in a music retrieval task that is generally robust to quality of melody extractor! We could see an improvement in various other parameters as well like stability, flexibility and retention of additional information.
- **RCA Time and Note**: These novel algorithms in music retrieval performed extremely well benchmarked against each other and the modified Mongeau-Sankoff algorithm, achieving a recall of 100% when paired with the novel melody extractor.

Alongside these novel algorithms, we also modified the Mongeau-Sankoff algorithm to utilise it for music retrieval by changing the following:

- **Tempo Invariance**: Mongeau-Sankoff requires the tempo to be known beforehand which is not always known to us. Thus we implemented a technique that will auto detect the tempo of the audio, and thus allows us to use Mongeau-Sankoff without knowing tempo beforehand.
- **Scale Invariance**: We transposed all scales of the audio to Cmajor (or Cminor depending on original scale) to make the data scale invariant, which allows the Mongeau-Sankoff algorithm to work in general as well.
- **Length Similarity**: Mongeau-Sankoff assumes the length of the files to be same. This is rarely the case, and hence we implemented a sliding window technique to ensure Mongeau-Sankoff can work for general audio files as well.

The end conclusion of all this work entails that we built a music retrieval system that had the following improvements and properties:

- **Robust**: Due to the melody extraction being significantly improved, our data is more robust to noise that may creep in. This is because it will be filtered out by the melody extractor, and won't disturb our retrieval system at all.
- **Discriminatory**: As seen by the experimentation, our model gains a high margin of discrimination and thus improves the confidence that our model will work well even under exceptional circumstances.
- **General**: As mentioned above, our model is invariant to a lot of variations in the input. That allows us to utilise our model on a wide range of data, making it extremely general.
- **Speed**: Our model pre-processes data in a way that allows for a huge speed bump. Coupled with the fast retrieval mode (RSA Time), we can achieve high speeds to retrieve audio files.
- **Accurate**: Finally, the most important aspect of any model, our retrieval system is extremely accurate. Using the proposed novel model(s), we get a 100% accuracy while also being extremely fast!

6 Future Work

- **Stride Length Analysis**: As talked about previously, both RSA algorithms rely on sliding windows. We know accuracy decreases with increase in stride length-but by how much? Increasing stride length would cause fewer number of notes to be considered and would thus be faster. This is thus a **hyperparameter** which we can tune according to our data.
- **Humming implementation**: We plan on testing the dataset on real world scenario by singing tones such as Happy birthday into the microphone and then converting the recorded WAV file into a MIDI file which can be further used as a query.
- **Composer detection**: What is usually done for these tasks is that they perform detection by feeding the spectrograms of the songs to a classifier [6]. Our IR implementation can also be used for this task by picking out the most common composers with high similarity scores and giving that as the output.
- **Compare to DejaVu**: This is one scope of improving the evaluation we had so far. Dejavu suffers from a problem where if we change the speed of playing the notes in the query, then the system wont be able to recognize this as its time dependent. Our Text-based similarity approach should theoretically outperform Dejavu on the basis of metrics such as Recall etc.
- **Vector Space Models assistance**: We can trim down on the searches by first performing a VSM based similarity of our queries with the data. If the similarity is above some threshold, only then we move on to comparing the similarities on the basis of our Levenshtein distance (costly operation).
- **Edit distance improvement through R* trees** : The current implementation of edit distance is of the complexity O(mxn) where m and n are the lengths of the note representations. This can be further improved to O(mlogn) by using R* trees. Moreover, the current implementation is not vectorized as well and this is something which can be done to improve the retrieval speeds. [4].

7 Possible Applications

1. **Music Recognition**-The most obvious application of our project is in music recognition. The user can input a song derivative as their query and get back the song from which the sample was excerpted-thus working as an extended version of Shazam
2. **Melodic Similarity**-Suppose a musician writes a melody. He could send this melody as a query and get back a list of results of pre-existing songs that sound similar. He could then use ideas from these songs to continue his song.
3. **Copyright Infringement Detection**-Since the project is based essentially on melody matching, it could also be used on platforms like Youtube where detecting whether a video has used a copyrighted song becomes important

References

[1] Melody extraction using skyline.

[2] Henry Boisgibault, Mathieu Giraud, and Florent Jacquemard. What does the mongeau-sankoff algorithm compute? 2019.

[3] Sungkyun Chang. Neural audio fingerprint for high-specific audio retrieval based on contrastive learning. original-date: 2021-02-10T07:16:19Z.

[4] Xueyuan Gong, Simon Fong, and Yain-Whar Si. Fast fuzzy subsequence matching algorithms on time-series. *Expert Systems with Applications*, 116:275–284, 2019.

[5] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[6] Zhen Hu, Kun Fu, and Changshui Zhang. Audio classical composer identification by deep neural network, 2016.

[7] Sylvie Hébert and Isabelle Peretz. Recognition of music in long-term memory: Are melodic and temporal patterns equal partners? *Memory Cognition*, 25(4):518–533, Jul 1997.

[8] Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, 1990.

[9] G. Ozcan, C. Isikhan, and A. Alpkocak. Melody extraction on MIDI music files. In *Seventh IEEE International Symposium on Multimedia (ISM'05)*, pages 8 pp.–.

[10] Matevž Pesek, Špela Medvešek, Anja Podlesek, Marko Tkalčič, and Matija Marolt. A comparison of human and computational melody prediction through familiarity and expertise. *Frontiers in Psychology*, 11:3418, 2020.

[11] ryohey. *Signal-The Online Midi Editor*. Nov 2021.

[12] George Tzanetakis, Georg Essl, and Perry Cook. Audio analysis using the discrete wavelet transform. In *in Proc. Conf. in Acoustics and Music Theory Applications. WSES*, 2001.

[13] Alexandra Uitdenbogerd and Justin Zobel. Melodic matching techniques for large music databases. pages 57–66, 01 1999.

[14] Alexandra Uitdenbogerd and Justin Zobel. Manipulation of music for melody matching. *Proceedings of the 6th ACM International Conference on Multimedia, MULTIMEDIA 1998*, 08 2002.

[15] worldveil. Dejavu. `https://github.com/worldveil/dejavu`, 2019.

[16] Yu-Te Wu, Yin-Jyun Luo, Tsung-Ping Chen, I-Chieh Wei, Jui-Yang Hsu, Yi-Chin Chuang, and Li Su. Omnizart: A general toolbox for automatic music transcription, 2021.