1 Introduction

We have been investigating sequence alignment methods for identifying structure in music. Given a page of lyrics and the associated chords (i.e., a chord sheet), even a person who is unfamiliar with the song is able to use repetition in the lyrics to identify choruses, repetition in chord sequences to identify verses, and repetition in phonemes to identify rhyme schemes. (Also able to identify other versions of the same song and distinguish between the song and meta-data encoded in the chord sheet?). This meta-data contributes significantly to how the song is perceived and appreciated (cite repetition paper).

We have endeavored to solve this problem using an alignment approach in order to perform unsupervised annotation of a large corpus of unlabeled chord sheets. Identifying repetitive elements or motifs in sequences has been widely-studied in the fields of bioinformatics and to a lesser extent in natural language. The Needleman-Wunsch (NW) algorithm is designed to discover optimal end-to-end alignments of sequence pairs. The Smith-Waterman (SW) algorithm varies slightly from the NW algorithm in that it is designed to find areas of high local-alignment.

We have developed a sequence alignment approach for identifying verse-chorus and rhyme-scheme structures in chord sheets. Our approach first uses pairwise Multiple Sequence Alignment (MSA) using the NW algorithm to identify and validate the actual song content in the chord sheets. Once the song content is validated, we use a novel hierarchical alignment approach loosely similar to a SW alignment of NW sub-alignments to identify choruses from repetitive subsequences of lyrics (and chords?) and verses from repetitive subsequences of chords. For this purpose we have developed a Generalized Alignment Module (GAM) which is capable of aligning sequences of any type (e.g., strings, chords, alignments, phonemes). Finally we use our GAM to align phonemes for the purpose of rhyme scheme detection. We report the results of our approach on a set of 100 chord sheets that have been manually labeled and outline our vision for how this work might be extended.

2 Related work

3 Methods

We first describe our Generalized Alignment Module (GAM) and then describe how the GAM is employed in multiple ways to detect structure in chord sheets.

3.1 Generalized Alignment Module (GAM)

Identifying musical structure depends largely on the ability to effectively identify regions of self-similarity. Self-similarity can be found in many different musical viewpoints: lyrics, chords, and even phonetics. To evaluate self-similarity across different view points without repeatedly solving the same basic problem of sequence alignment, we conceptualized a Generalized Alignment Module (GAM) capable of performing a NW alignment on a pair of generic sequences.

As input the GAM takes a SequencePair object p representing a pair of generic sequences $< m, n >$ (let t_m and t_n denote the type of elements of which the sequences m and n are respectively composed). The GAM handles the problem of setting up and populating a 2D scoring matrix according to the NW algorithm drawing on the specific SequencePair $score(i, j)$ of p to calculate the cost of aligning the ith element of m with the jth element of n. Each specific SequencePair implementation also provides an AlignmentBuilder implementation which the GAM uses in back-tracing to generate the optimal alignment for the input SequencePair. An AlignmentBuilder implementation includes functionality to initialize and append elements (or gaps) to a pair of gap-aligned sequences of type t_m and t_n. The final result is an Alignment object representing the fully gap-aligned sequences of type t_m and t_n. WOULD A FIGURE OR FORMAL ALGORITHM HELP?

By having different SequencePair implementations for different types of sequences, $score(i, j)$ can be customized to different element types. For example, the scoring function to align two chord symbols can incorporate musical theory to compute the musical relatedness of two particular chords whereas the scoring function to align two phonemes can reference published scoring matrices to determine the match score of two particular phonemes.

The GAM is optimally implemented to run in linear-time and linear-space

(cite ?).

3.2 MSA of lyric sources

WHAT ABOUT DOING AN MSA OF ALIGNMENTS OF WORDS?!?!? THAT WOULD REALLY DEMONSTRATE THE VALUE HERE.

Prior to identifying structure in chord sheets we identified and validated the musical content of the chord sheets. Many chord sheets contain headers and footers that do not constitute part of the encoded song. Identifying the musical content can be accomplished by comparing the lyric content with 3rd-party lyric sites. Lyric sheets from these sites may also contain headers and footers that are not part of the song. Our solution to this problem uses Multiple Sequence Alignment to align the lyrics from several 3rd-party lyric sites.

Multiple Sequence Alignment is an NP-complete problem with several polynomial-time heuristic solutions. One common approach is the Pairwise MSA in which all sequences are aligned pairwise and the pairwise alignments are then sequentially merged according to their pairwise alignment scores.

The GAM framework provides a fairly simple solution to the MSA problem. Similar to the Pairwise MSA approach, we first computed all pairwise sequence alignments of lyric sequences to determine a score of relatedness between each pair. The alignment a with the highest global alignment score was selected as the seed for our progressive MSA. We consider this progressive MSA as a sequence of columns, each column being an alignment of characters/gaps.

The full MSA is accomplished by selecting the unaligned sequence u with the highest accumulative pairwise alignment score with the sequences already in the MSA. A SequencePair is created to represent the pair $< u, a >$. In this particular SequencePair implementation, the first sequence is a character sequence and the second sequence is a sequence of aligned characters. The scoring function $score(i, j)$ computes the sum-of-pairs scoring scheme calculated by summing the scores of aligning the $i$the element of u with each of the characters in column j of a. The GAM framework aligns u and a resulting in a progressive MSA with u and a aligned. This process is repeated until no unaligned sequences remain.

Lyrics from three 3rd-party lyric sites were multiply aligned using GAM MSA for each song s for which title and composer were an exact match

(ignoring case). ??????WHAT COSTS????????. A consensus lyric sequence was obtained by backtracking (in reverse) from the highest-scoring alignment position in the MSA to (but not including) the first alignment position with an alignment score of $le0$. This consensus c_s we defined as the full lyrical content for s.

For each chord sheet t whose title and composer matched those of song s, a GAM alignment of t and c_s was performed (with tagged chords removed from t). The aligned region of t (plus chord lines associated with the aligned region of t and intro/outro chord lines without associated lyrics) was considered the musical content of t. The rest was discarded. The musical content was validated for completeness by ensuring that at least 80% of the characters in c_s matched in the alignment with t. Each character of t was replaced by the character with which it aligned in c_s.

3.3 Hierarchical lyric alignment for chorus identification

Choruses are denoted by blocks of consecutive lyric lines that are repeated within a song. To identify choruses in our validated chord sheets, we performed a hierarchical alignment of lyrics. This works by first aligning all lines of the song pairwise and then observing alignments of consecutive matching lines with other consecutive matching lines. The choice to align lines individually rather than align the entire lyrical content was grounded in the belief that newline characters constitute the most reliable structural annotation in chord sheet lyrics, denoting units of repetition at several levels even if the exact contents of those units may be subject to some variability.

Lines were aligned pairwise using the GAM framework. Alignment scores were binarized (0/1) in that line pairs with a global alignment score $> 0.9*$ the minimum of the two line lengths were considered matching. Alignments of consecutive matching lines was achieved by creating a triangular matrix of binarized alignment scores where the score in the ith row and the jth column represents the binarized alignment score of the jth lyrical line with the $(j+i)$th lyrical line. Thus rendered, consecutive matching lines appear as sub-rows of repeating 1's (see Figure ???).

From this triangular matrix, candidate choruses are extracted. A candidate chorus is any sequence of one or more consecutive lines that repeats. We also considered as a candidate chorus any sequence of one or more consecutive

lines that starts and ends with repetitive lines and in which no two consecutive lines are non-repetitive (more than one consecutive non-repetitive lines was determined to be more approximate of a pre-chorus than a chorus). This allows proper recognition of choruses for songs like John Denver's "Rocky Mountain High" where a line internal to the chorus structure is allowed to vary from one chorus to the next.

Candidate choruses are scored according to a weighted linear combination of the candidate chorus line count, the number of times the candidate chorus repeats, and how well the candidate chorus is distributed throughout the song:

EQUATION

Weights were trained on a hold-out portion of the labeled dataset (TODO?).

The candidate chorus with the highest score was determined to represent the chorus.

3.4 Hierarchical chord alignment for verse identification

Verses are denoted by blocks of consecutive chord lines that are repeated within a song. Ignoring lines already identified as chorus lines, we identified verses using the same process that was used to identify choruses, only rather than aligning lyric lines we aligned chord lines.

SAME WEIGHTS?

Once choruses and verses were identified, the remainder of the structure was determined as follows. Lines before the first verse/chorus were marked as intro. Lines after the last verse/chorus are marked as outro. All other lines are marked as bridge lines. THIS WILL PROBABLY CHANGE TO DISTINGUISH BRIDGE FROM MUSICAL INTERLUDE (i.e., ABSENCE OF LYRICS).

3.5 Phoneme alignment for rhyme scheme identification

Using the CMU pronunciation dictionary and the CMU G2PConverter, we converted the final syllable of each lyric line l into a list of phoneme sequences (each phoneme sequence representing a possible pronunciation of l). We then attempted doing pairwise alignments of phoneme sequences for

each line with the phoneme sequences for each of the following 4 lines. The maximum alignment score for each pair of lines was used as the pairwise alignment score for those lines. The paired line with the highest maximum alignment score $\geq$ MATCHINGLINETHRESHOLD was considered to be a rhyming line for l (if the highest maximum alignment score was $<$ MATCHINGLINETHRESHOLD, then l was determined not to having a rhyming line). The sequence of line numbers each (optionally) paired with a subsequent rhyming line number constitutes the rhyme scheme for the validated chord sheet. ???????COSTS????????.

3.6 Melody-Lyrics alignment for identifying melody staff in a MIDI file

4 Results

4.1 Data

4.2 Experiments

5 Discussion

We're ultimately interested in the problem of using machine learning to allow computers to generate new music. The ability of machine learning methods to learn the abstract structures of segmentation and rhyming is vastly enhanced with the automatic annotation of these elements in unlabelled data.

Pairwise alignment of intra-segmental chord lines for substructure identification (future work)

6 Conclusion

Algorithm 1 MAX-DIR GREEDY HEURISTIC

Input: Weighted bidirected graph, G, and min edge weight, w_{min}

Create a forest, F
For each vertex $v_i \in G$, add tree t_i to F containing v_i
Create a set S of all edges in G with weight $w_e > w_{min}$
while S is not empty **do**
    Remove an edge e with maximum weight from S
    if e connects two different trees, t_1 and t_2, **then**
        add e to F, combining t_1 and t_2 into one tree
        if e is not a directed edge **then**
            for all vertices v_2 in t_2 **do**
                Flip orientation assignment of v_2
    else if e is a directed edge **then**
        add e to F
    else
        discard e
return F, a weighted directed subgraph