Markov Decision in Tonal Music Generation

David Benoit
University of Massachusetts Lowell
Department of Computer Science
david_benoit@student.uml.edu

Brian Chiang
University of Massachusetts Lowell
Department of Computer Science
brian_chiang@student.uml.edu

Author Keywords
Markov, Decision, Tonal, Music

INTRODUCTION

The objective of this project was to explore applications of Markov Decision Process (MDP) to the generation of tonal music. The system creates a Markovian state space based on a user-supplied harmonic progression, and applies an algorithm based on MDP and value iteration. The intended result is for the system to approximate the most suitable voicing of each chord in the progression, such that the created musical excerpt contains smooth melodic lines which interact with each other correctly based on a Schenkerian musical analysis of the excerpt.

Definitions

Definitions of terms used in this paper. Markov Decision Process (MDP) - A mathematical model for approximating the best path through set of states.

MDP State - A structural representation of state in an MDP model which consists of the state to be modeled, and a value representing the state's suitability toward reaching the MDP model's goal.

MDP State Space - The set of all states in an MDP, arranged by adjacency to other states.

Factored MDP - An approximation of an MDP computations of the values of states are carried out on groups of states rather than individual states. Factored MDPs are employed when state spaces are so large that they are cumbersome or impossible for computers to work with.

Value Iteration - An iterative process over a state space which updates an MDP state's value with a reward or penalty based on the state's suitability.

Music Theory - The formalization of implicit rules by which composers have written music throughout history.

Schenkerian Musical Analysis/Theory - A commonly used system for analyzing music quality devised by Heinrich Schenker.

Harmony/Chord - A general group of notes which sound good together.

Harmonic Progression - A sequence of harmonies.

Chord Voicing - A specific subset of all possible notes in a harmony.

Voicing Progression - A sequence of chord voicings.

Melodic Lines - A melody played by a particular voice or instrument in an excerpt.

Counterpoint - The interaction of melodic motion between two voices or instruments.

Stepwise Motion - Occurs when two notes are a single letter name apart.

Melodic Leaps - Occur when two notes are more than a single letter name apart.

Tonic - The tonal center of a key, progression, or chord. For example the note "C" is the tonic of the key of "C major".

Convolution Kernel/Matrix - An AI technique used in graph/-matrix calculation.

PROJECT DESCRIPTION

When given a set of chord symbols representing a harmonic progression, the tonal music generation system's goal is to approximately determine the most compatible voicing for each chord in the progression. The more compatible the chord voicings in the progression are with one another, the more pleasant the generated excerpt will sound to the ear. Compatibility of voicings with one another is determined by analysis of the melodic lines resulting from the sequential playing of chords in a progression.

To determine the approximate best set of chord voicings for a specified progression of symbols via Markov Decision Process, the first action the system must perform is to enumerate the state space. The state space (Q) of all possible voicings of a given progression can be represented by the tuple {C, V , R}, where:
C = the sequential list of chords in the progression

V = the set of voices/instruments
R = C x V = the set of notes per chord which are playable given the range of each voice in V
Q = R x R = the set of possible relationships of each chord in the progression to each other chord

Th reason that Q is equal to R x R is that the entire sequence of chords matters in a harmonic progression. The suitability of each chord is determined not only by its immediate neighbors, but by its relation to the entire progression. Such a large state space is difficult to compute because Q is the cartesian-product of four sets, and its size grows on the order of magnitude O($(CV)^2$). Since the state space grows quadratically as either C or V grows, the amount of computing power required to compute the best chord voicings in a progression quickly becomes unfeasible for most computers.

As such, the tonal music generation system is unable to populate the entire state space which would be required for a traditional MDP algorithim. Instead, the system attempts to factor the state space by grouping similar subsets of progressions of voicings together, and inferring their compatibility with other subsets as a group. The resulting factored state space (Q^1) can be represented as Q^1 = R = C x V, and the factored state space grows on the order of magnitude O(CV), which is linear growth as either C or V grows. To create better space constraints, the program limits V to exactly 4 voices, which is the most common number of voices in academic chorale style writing. Therefore, the final, optimized state space of the system grows on the order of magnitude of O(C), which is linear as the length of the chord progression grows.

Since the factored MDP's state space is {C x V , V = 4}, each state corresponds to a single chord voicing. As such, the tonal music generation system's first action is to populate into memory each possible chord voicing for each chord in the progression. It factors the voicings into a two-dimensional table, where voicings in the x-direction (rows) progress sequentially corresponding to the harmonic progression, and voicings in the y-direction (columns) are each different voicings of the same chord. Each voicing is treated as a Markovian State, and is given a value which represents its suitability for the progression.

All state values are initialized to 0 when the table is created. Since the table is generated algorithmically, it is shuffled to ensure there are no existing relationships between. Due to the inherent randomness of shuffling the state space, the system will generate a different voicing progression each time it run, even if supplied the same harmonic progression as input.

Once the table is constructed, the system begins performing value iteration on each isolated row, treating each as its own progression of chord voicings. The voicings are given a value based on their suitability to both the harmonic progression, and to the other voicings in the row. The result is that small groups of adjacent voicings within each row will end up with similar state values, because values are determined partially based on suitability to the surrounding voicings.

Once value iteration has been performed once on each row, the columns in the table are individually sorted such that the states with the highest values ascend to the top of the table, and states with the lowest values descend to the bottom. Since many small groups of voicings have similar state values based on their suitability to each other, the groups have a tendency to rise together if they are compatible with both each other and the progression as a whole, and fall together if they are incompatible.

This sequence of value iteration and column sorting is carried out by the system as many times as is specified by the user. Ultimately, the top row of the table converges to be the approximate best series of chord voicings for the given harmonic progression. Finally, the system generates an audio recording of the resulting best progression of voicings, and prints out the progression data.

Code

This project was implemented using the Racket programming language. It's musical facilities were built using the RSound-Composer library, developed by David Benoit in parallel with this project. RSound-Composer is a library which provides a high-level, music-theory based API for Racket's RSound library, including data structures for use in music theory representation, and procedures which perform common operations on musical data. It also provides automatic scheduling and playback of created music. RSound is a sound library written by John Clements.

All of the code which implements the project-specific functionality for Markov Decision in Tonal Music Generation is contained within three files, mdp-state.rkt, mdp-reward.rkt, and mdp-main.rkt. All other files at the root of GitHub repository are either documents or legacy code (code which is no longer used by the project).

mdp-state.rkt

This file provides an implementation of Markovian states, state-spaces, and musical state functionality which is specific to this project.

<#struct>state: Represents a state. A state contains a single chord voicing, a reward/penalty value for use in value iteration, and the user supplied harmonic progression. In retrospect, storing harmonic progression data in every state was unnecessary.

<#struct>part-range: Represents the range of notes a voice is able to play. Stores notes representing the highest and lowest notes that a particular voice has the ability to play.

<#procedure>voicing-compare: Legacy function which returns 0 if voicings are equal.

<#procedure>pitch: Creates a note with a duration of *null-beat*

<#procedure>enumerate-part-range: Takes a part-range structure and returns a list of all notes between the two end points.

<#procedure>part-range-valid-pitches: Takes a part-range structure and a harmony structure, and returns a list of all notes in the enumerated part-range which are also part of the given harmony.

<#procedure>populate-state-space: Takes a harmonic progression, a note structure representing the key of the progression, and list of part-ranges, and creates a Markovian state table. The table is arranged such that the x-direction rows represent the harmonic progression of the excerpt, and the y-direction columns represent individual possibilities for each harmony.

<#procedure>shuffle-state-space: Takes a state space, and shuffles the y-direction columns to ensure there are no implicit relationships present in the state table due to the algorithmic generation of the state space.

<#procedure>print-state-space: Takes a two dimensional list of states and prints it

<#explicit-data-value>example-part-range-list: A list of part-range structures representing the soprano, alto, tenor, and bass voices of a classical four part chorale.

mdp-reward.rkt

This file implements functionality for a Markovian reward function. The reward function provides reward points and penalty points based on a Schenkerian Musical analysis of the given progression of voicings. There was not enough time to implement all aspects of Schenkerian analysis that were initially planned, however, enough functionality was implemented that the reward function provides a partially accurate measure of suitability of a chord voicing in its given context.

<#procedure>convolution-matrix: A constructor which takes a progression of chord voicings and the index of an individual voicing, and returns a three-column convolution matrix consisting whose leftmost column is a list of all notes in the chord preceding the indexed chord (or null if indexed chord is the first chord in the progression), whose center column is a list of notes in the indexed chord, and whose rightmost column is a list of all notes in the chord succeeding the indexed chord (or null if indexed chord is the final chord in the progression). The convolution matrices returned by this function are used by the reward function and its helpers.

<#procedure>map:* Applies a function to an arbitrarily long list of equal-length lists. This overcomes a limitation of Racket's built-in mapping procedure, which only has the ability to map over a fixed number of lists in any given expression.

<#procedure>mostly-stepwise?: A helper function of the main reward function, which checks to see if a segment of the voicing progression follows the rule of Schenkerian music theory that musical excerpts contain mostly step-wise motion. Takes a convolution matrix and returns false if total number of melodic skips in all melodic segments produced by individual voices in the matrix is greater than one. Returns true otherwise.

<#procedure>unresolved-skips?: A helper function of the main reward function, which checks to see if a segment of the voicing progression follows the rule of Schenkerian music theory that any skips in a musical excerpt are followed by stepwise motion in the opposite direction. Takes a convolution matrix and returns false if there exists a melodic segment in any voice in which there is a skip of an interval greater than a major 3rd (equivalent to the interval of C to E) which is not directly succeeded by step-wise motion in the direction opposite the skip.

<#procedure>soprano-proper-cadence?: A helper function of the main reward function, which checks to see if a segment of the voicing progression follows the rule of Schenkerian music theory that the melody of the highest voice in an excerpt must end on the tonic, and must progress step-wise to to the tonic (For example, given an excerpt in the key of C Major, the highest voice must either end with the notes D $\rightarrow$ C or B $\rightarrow$ C).

<#procedure>bass-begins-on-tonic?: A helper function of the main reward function, which checks to see if a segment of the voicing progression follows the rule of Schenkerian music theory that the first note played by the lowest voice is the tonic.

<#procedure>bass-proper-cadence?: A helper function of the main reward function, which checks to see if a segment of the voicing progression follows an approximate rule of Schenkerian music theory that the final note played by the lowest voice is the tonic. Schenkerian music theory is more sophisticated regarding the treatment of the final notes played by the lowest voice, but due to project time constraints, such an analysis was not possible to fully implement.

<#procedure>reward-function: This function takes a generated voicing progression, and updates the values of each state in the progression based on a partial Schenkerian analysis provided by its helper functions. To determine the value of each state, the reward function analyses the state's musical fitness in relation to the two states immediately adjacent to it in the progression. It does this by creating a convolution kernel based on the voicings of the three given states, calling helper functions on the convolution kernel to analyze the local states fitness in relation to each other, then moving the kernel to the next state and repeating the process. The result is that each state is given an updated value based on the context it is currently present in.

mdp-main.rkt

<#procedure>value-iteration: This function recursively carries out value iteration over a factored state space. Each re-

cursive call corresponds to a complete iteration over the state space.

The function takes a state-table and a desired recursion depth. First, it checks to see that the length of all table columns is equal. It will trim any column which is longer than the shortest column.

Since the state table is represented as list of linked lists, each column in the table corresponds to a linked list whose members all have the same harmony. However, the system's reward function requires the voicings to be factored as chord progressions. This can be accomplished by reconstructing the table such that the linked lists correspond to the state-tables rows. Therefore, this function creates a temporary table, the iteration table, which is identical to the state table but the lists represent rows instead of columns.

For each row in the state table, this function passes the entire row of states to the reward function to be analyzed. Each state's value is updated based on its fitness in the context of the row it is currently in.

Once all rows have been passed to the reward function, a new state table is created from the iteration table, which is identical to the old state table but with updated Markov values. To decide which states will be compared to which others during the next call of value iteration, the new state table is re-factored. Each column in the state space is sorted such that the states with the highest reward values (or smallest penalty), rise to the top of the column. Groups of suitable states tend to rise together, because the reward function measures each states suitability based on its context in surrounding states.

Eventually, the top row of the state table converges to a single progression of voicings, where each state has the highest state value in the column. Suitability of voicings in the particular context of the top row is ensured, because voicings which are not suitable in the top row are penalized, and more suitable voicings take their place.

Therefore, the top row of the state table after many value iterations ultimately results in an approximate best progression of chord voicings.

<#procedure>best-path: This function takes a state table and returns a list represinting the top row, or approximate best voicing progression.

<#procedure>run-markov-system-trial: This function takes a harmonic progression and a key. It creates a factored state space, and displays the result of performing value iteration 0 times, 10 times, 100 times, and 1000 times.

<#procedure>play-markov-path: This takes a list of voicings and stream it as a musical audio excerpt.

For example:

```
(require "mdp-state.rkt")
```

```
(play-markov-path
  (best-path
    (value-iteration
      (map
        shuffle
        (populate-state-space
          progression1
          (note 'C 5 null-beat)
          example-part-range-list))
        100)))
```

Would create a shuffled state space, apply value iteration 100 times, and play the audio of the best voicing progression of progression1 in the key of C.

ANALYSIS OF RESULTS

The Markov Decision in Tonal Music system was tested several times using different data sets as input each time. The resulting output of the system, including audio samples, is available at:

https://github.com/benoid/tonal-music-generator/tree/master/trial-results

Five different harmonic progressions were supplied to the system, and ten trials were performed per progression for a total of 50 trials. Each trial consists of an audio and textual snapshot of the best progression after zero, ten, one hundred, and one thousand value iterations.

The system was moderately successful at learning better music composition techniques when compared to the randomized voicings of the initial state space, but no musical output of any system trial comes close to passing even the most superficial Schenkerian contrapuntal analysis.

Therefore, a more qualitative analysis of each individual trial was performed by ear, and the results were recorded at:

https://github.com/benoid/tonal-music-generator/tree/master/trial-results/notes.txt

Analysis of system output confirms that greater magnitudes of value iteration tend to increase the smoothness of the excerpt's melodic lines when compared to excerpts produced with lesser or no value iteration.

The success of a trial was determined by whether or not it produced a result with one of the following improvements over the original zero-value-iteration result:

1) Smoother, more stepwise melodic lines

2) Melody or Bass ends on the tonic of the key, whereas it originally did not.

3) Bass ends with stepwise motion (Ti-¿Do or Re-¿Do) or dominant-tonic motion (Sol-¿Do) whereas it originally did not

Failure of a trial was determined by the following conditions:

1) Melodic lines become significantly more jagged after value iteration, even if there were some other improvements

2) There were no tangible improvements to the output after value iteration.

The success rate based on this criteria was that 86% of trials were generally successful and showed some tangible improvement after value iteration.

14% of trials were generally unsuccessful, producing either no tangible improvement, or producing output which was determined to be worse than before value iteration. Such a high failure rate is possibly a result of the algorithm's restriction of the state space. Due to program time and space constraints, the system sacrifices many comparisons which would otherwise be performed in a traditional MDP.

When analyzing the number of value iterations performed before the factored MDP converged on an approximate best solution, it was determined that 90% of trials converged between zero and ten trials, 10% converged between ten and one-hundred trials, and 0% converged between one-hundred and one-thousand trials. The overwhelming tendency to converge early suggests the reward function may have been overly biased to certain features found in the voicing progressions.

DISCUSSION

There were many obstacles to creating this project, the largest of which was managing the size of the state space. Since specific context is important when determining the suitability of a chord voicing, voicings can not be treated as individual states the way most MDP systems treat them. As such the original design for the project called for each state to contain a current chord voicing, and a list of the sequence of chord voicings leading up to the current voicing. The issue was that such a state space would have a memory footprint orders of magnitude greater than an average computer could store, and would take an incredibly long time to compute.

To remedy this problem, the system could not attempt a brute force approach. Instead, chord voicings are treated as individual states, and the state space is factored into harmonic progressions. When value iteration is run, states are compared to adjacent states in their factored harmonic progressions using a technique based on kernel convolution. After each iteration of value iteration, the state space is refactored, grouping states with the highest values together into their own progressions.

There are some ethical considerations to consider when discussing computer generated music. Is music still art if it is generated by an algorithm? Will the quality of computer composed music surpass the quality of human composed music? Will render human composers obsolete, crushing an industry?

CONCLUSION

The Markov Decision in Tonal Music system was ultimately less successful than anticipated, but was mildly successful at learning the rules of music composition and counterpoint. The large state space of the problem domain could not modeled adequately enough by the system's factored MDP. In conclusion, Markov Decision Process may not be the most ideal method of modeling tonal music composition.

ACKNOWLEDGMENTS

The work described in this paper was conducted as part of a Fall 2016 Artificial Intelligence course, taught in the Computer Science department of the University of Massachusetts Lowell by Prof. Fred Martin.