Musical Style Replication Using Apprenticeship Learning

Joel Acevedo, Stephen Gliske, Magesh Jayapandian

1 Introduction

Computer-based analysis of tonal music has been an active area of research for over a decade. In particular, machine learning based methods have been applied to composing music and solving various problems in musicology such as classification, visualization, search and stylistic analysis. Building a model of a composer's style is a challenging problem of great interest in the Music Information Retrieval and Musicology research community. Interesting applications include style characterization tools for the musicologist, generation of stylistic metadata for intelligent retrieval in musical databases, music generation for Web and game applications, machine improvisation with or without interaction with human performers, and computer-assisted composition [4].

Simply generating music by computer is a field of active research. However, in our research, we have been unable find a specific instance of the AL algorithm. Markov chains are used for stochastic composition, but the probabilities are usually given as inputs to the algorithms, rather than learned. Learning techniques often focus on artificial neural networks or genetic programming techniques, although the MUSE system by Schwanauer utilizes an algorithm to learn voice leading rules. David Cope's work presented in [3] combines grammatical generation system with a rule-based approach to compose music in a composer's style. The novelty of our approach lies in the Machine Learning method chosen. This method models the composer's inclinations and learns from musical pieces without incorporating rules or making assumptions about musical structure.

1.1 Problem Statement

The problem is one of learning the distinctive style of a composer of classical music and using that knowledge to compose a new piece that also exhibits that style. Using a corpus of musical compositions as training data, all from the same composer and belonging to the same genre, our goal is to compose music that resembles them stylistically. Specifically, we would like to determine whether Apprenticeship Learning (AL) via inverse Reinforcement Learning is capable of accomplishing this task.

2 Markov Decision Processes and Problem Representation

A finite-state Markov Decision Process (MDP) can be represented as a tuple (S, A, T, γ, D, R) where S is a finite set of states; A is a set of actions; $T(s, a, s') = \Pr(s'|s, a)$ is a set of state

transition probabilities; $\gamma \in [0, 1]$ is a discount factor; D is the initial-state distribution from which the start state s_0 is drawn; and $R : S \to \mathbb{R}$ is a reward function that encodes the desirability of a state.

In order to use Apprenticeship learning we model music composition as a Markov Decision process. Each state in $\in S$ of the process is a musical sample of a piece described by a set of features. An action $\in A$ is an action the composer took to change the features of a musical state and therefore transition to another. The transition model T tell us what actions, from all the possible actions in our world, the composer took when writing the songs we used as training data, at what states he took each action and their frequencies. We delay the interpretation of γ to Sec. 4. In our model, D represents the probability a song starts with a particular musical state s. The reward function R is used to model the style of the composer; specifically, we think of the reward function as the preference or desirability the composer assigns to each musical state. There is no fixed definition for musica style, therefore explicitely stipulating the reward function is a difficult task. This is the reason why we decided to agree on the form of the reward function and the features that *might* define the style of a composer, i.e. the reward function without explicitely stating it. Instead, we use Apprenticeship learning and the composer's work to learn how this features affect R.

We express the properties that can affect the reward function as a vector of features ϕ over states and we assume that there is some "true" reward function, $R^*(s)$ that is a linear combination of these features : $R^*(s) = w^{*T}\phi(s)$. The role of the Apprenticeship Learning algorithm is to learn the weights vector w^* that will produce a reward function close to the unknown true reward function given the feature vector provided by our model. In the following paragraphs

One can think of sampling a piece of music at fixed intervals, equal to the smallest division in the piece, say every eighth or quarter note. A state will correspond to the result of such a sampling, and will be labeled by a set of eight numbers $\langle r_1, r_2, p, i, c, c_i, c_T, \sigma \rangle$. Each musical phrase is conceptually divided into $N_1 = \|domain(r1)\|$ portions, with r_1 denoting this state occurs in the r_1-th portion. Likewise, the piece is conceptually divided into $N_2 = \|domain(r_2)\|$ portions, and r_2 denotes in which portion is the current phrase. The values of r_1 and r_2 are given by,

$$r_1 = \lceil \frac{m-1}{M_k - 1} N_1 \rceil \tag{1}$$

$$r_2 = \lceil \frac{k-1}{K_t - 1} N_2 \rceil \tag{2}$$

$$\tag{3}$$

where $m \in \mathbb{Z}^+$ and $k \in \mathbb{Z}^+$ are the position within a phrase and phrase number on which the current state is, respectively. M_k denotes the number of states in the k-th phrase of the song and K_t represents the total number of phrases in song t. For our experiments we chose N_1 and N_2 to be 8 and 4 respectively.

The value $p \in \{-7, -8, \ldots, 17, 18\}$ is the pitch of the melody, referenced with $p = 0$ being the key of the piece. For instance, if the key of the piece is F, the tone denoted as F4, in musical terminology, is taken as the reference ($p = 0$). The value $i \in \{0, \pm 1, \cdots \pm 24\}$ is the difference between the current pitch and the pitch of the previous state. The value $c \in \mathbb{Z}_{12}$

is the chord at the current time slice, of type $c_T \in \{1, 2, \ldots 6\}$ corresponding to the labels {*Major, Minor, Major 7, Dim., Half Dim. 7, Fully Dim. 7*}. The value $c_i \in \{0, \pm 1, \cdots \pm 11\}$ is the difference between c and the root of the previous chord. The value $\sigma \in \{0, 1, 2\}$ refers to the status of the melody, whether it is resting, continuing to sound a previous note, or starting a new note. We shall restrict the set of available states to those which occur in the training data.

Actions can be denoted by a set of six numbers $\langle \Delta r_1, \Delta r_2, \Delta p, \Delta c, c'_T, \sigma' \rangle$. The values $\Delta r_1, \Delta r_2 \in \{0, 1\}$ correspond to whether the next state is in the same portion or the next portion of the phrase and piece, respectively; $\Delta p \in \{0, \pm 1, \cdots \pm 24\}$ and $\Delta c \in \{0, \pm 1, \cdots \pm 11\}$ denote the change in the pitch and the root of the chord, respectively; The value of $c'_T \in \{1, 2, \ldots 6\}$ refers to the type of the chord in the state resulting from taking the action. The value σ' refers to the status of the target musical state, i.e. whether this action is going to rest, hold, or start a new note. If $\sigma' = 0, 1$ then Δp is defined to be zero, enforcing that a new pitch can only begin if a new note is played. Note that not all actions are available in all states. For instance, in the state $p = 12$, no actions are available which increase the pitch further, i.e. with $\Delta p > 0$. We shall also restrict the set of available actions to those which occur in the training data.

There exists an isomorphism between states and integers as well as actions and integers. Consider a finite set of finite sets of integers (in our case, either the set of all states or the set of all actions), such that the ith set has N_i values. A particular choice of one element from each set (i.e. the state variables of a particular state, or likewise for an action), denoted $\{x_i\}$, can be encoded as $\sum_i \prod_{j<i} N_j x_i$. It can be shown that this is an isomorphism, and is equivalent to interpreting each x_i as a digit, with each digit being in base N_i. In coding our algorithm, both representations, the integer and the set of integers, were useful and accounted for code efficiency.

The transition model $T(s, a, s')$ is defined as the probability of reaching state s' given that action a was taken in state s. For all allowed combinations of s and a, we define $T(s, a, s')$ to be one if a is the action which relates s and s', and zero otherwise. We are assuming deterministic actions—that the composer does not fail to move to the note the composer intended.

The feature vector ϕ is defined as a set $\phi^{(i)} : S \to [0, 1]$. We have indicator functions for the value of r_1 (8), r_2 (4), i (25), $c_i \mathrm{mod} 12$ (12), as well as regarding the pitch relative to the key (12), whether the pitch is in the scale of the key (1), the pitch relative to the chord (12), whether the pitch is in the chord (1), the chord root relative to the key (12), whether the chord root is in the scale of the key (1), and whether the chord type is type expected for the key (1).

We do not expect all features to be relevant for all composers, but cannot say a priori which are most important. Reducing the complexity is a standard dimensionality reduction problem. Since we are only interested in a linear combination of ϕ, PCA is a good choice. We computed the sample variance of ϕ of the training data, i.e. $\Sigma = \sum_{i=1}^{n} (\phi(s_i) - \mu)(\phi(s_i) - \mu)^T / n$, where n is the total number of states in the training data (over all music pieces) and $\mu = \sum_{i=1}^{n} \phi(s_i)$. We compute the eigenvalues and eigenvectors of Σ. We then take the K vectors with the largest eigenvalues, $u_1, u_2, \ldots u_K$ and form the matrix $U_{i,k} = u_k^{(i)}$. The actual ϕ vector we used is $\phi'(s) = U^T \phi(s)$, and now we only need to determine K weights

in the AL algorithm, where K is less than the original dimension of ϕ.

The other components necessary to relate our problem to a finite-state MDP are the initial state distribution D and the discount factor γ. In the generation portion of the algorithm, we assume that the number of states per phrase and the number of phrases are given, as well as the starting state. Thus D is trivially the given starting state. The discount factor is related to when the decision process terminates, which is slightly complicated since we fix the number of states. We treat this as a parameter of the system.

The other main component to represent the problem domain are policies. A policy $\pi : S \rightarrow A$, is a set of rules which determines which action should be taken in a given state. We can represent π as a vector such that $\pi(s) = a$ implies that action a should be taken in state s. As we restrict ourselves to only states that occur in the training data, many of the entries will be zero, meaning no action is defined since the state does not occur. MatLab has a sparse vector datatype, so that even though the dimension of a policy is on the order of a hundred million (all possible combinations of state variables), the memory size and query time is the number of non-zero elements.

3 Algorithm

We use the a machine learning technique called "Apprenticeship Learning" [7] that enables learning by observing expert behavior by maximizing an unknown reward function. It is sometimes called inverse reinforcement learning because the reward function is not given but learned. In our problem, the expert's behavior, or specifically Bach's music composition style, is observable from his Chorales which are encoded and treated as training data for our system. This data is used to compute $\hat{\mu_E}$, the expert's feature expectations (accumulated feature value vector). The apprenticeship learning algorithm tries to find a policy whose feature expectations comes closest to that of the expert. The main steps in this learning approach are as follows.

1. Start with an initial policy $\pi^{(0)}$ (selected at random) that maps states to actions. Compute the initial feature expectations $\mu^{(0)}$ using this policy and a valid start state. Initialize i to 1.

2. *Projection Method* [1]
   - Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T(\mu^{(i-1)} - \bar{\mu}^{(i-2)})}(\mu^{(i-1)} - \bar{\mu}^{(i-2)})$
   - Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
   - Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$

3. Terminate if $t^{(i)} \leq \epsilon$. Here ϵ is a pre-specified constant used to identify convergence.

4. With the current value of w, $w^{(i)}$ and the set of features ϕ, use the reinforcement learning algorithm (value iteration) to obtain the optimal policy $\pi^{(i)}$, i.e., one that maximizes the expected utility.

5. Compute the new feature expectations $\mu^{(i)}$ from the new policy.

6. Set $i = i + 1$ and repeat step 2 until convergence.

7. Obtain a start state as user input and use the learned policy to generate a new piece of music.

8. Decode the piece or convert it to MIDI to play on any sound device.

Note the AL algorithm outputs a set of policies $\{\pi_i\}$ as well as the set of feature expectation values for each policy $\{\mu_i\}$. The method ensures that at least one of generated policies has its feature expectations within some threshold of the composer's feature expectation vector. The optimal policy is found by solving the quadratic program (QP) $\min_\lambda \|\mu_E - \sum_i \lambda_i \mu_i\|$, subject to $\lambda_i > 0$ for all i and $\sum_i \lambda_i = 1$. MatLab has precoded routines to solve this QP. The weights λ_i are interpreted as the probability of following policy i. Thus in realizing the policy, for each state generated, a policy is picked at random, according to the weights λ_i, to determine the next generated state. Such a realization of the optimal policy can transcribed into recognizable music by specifying a key and a tempo with which to play the piece. We have currently implemented MatLab's sound routine to create a wave file of the output, although we hope to also implement a routine to convert the output to MIDI, a standard representation of music by notes instead of sound frequencies.

4 Reinforcement Learning

One of the steps of the AL algorithm requires the determination of the optimal policy for a MDP given a reward function. We decided to use a value iteration algorithm to exactly determine the optimal policy. It is assumed that one or more terminal state are specified, i.e. a musical states , as defined in previous sections, in which the musical piece ends. In actuality, terminal states are specified in the algorithm by inflating the reward of terminal states so that the policy direct understands that is a "very good thing" to reach the terminal state.

The Value Iteration algorithm calculates the utility (desirability) of each state of the world and then uses the state utilities to select optimal actions in each state. The utility of a state is defined as the expected reward from that state onward. In other words, the utility of a state is the utility of the state sequences, ending with a terminal state, that might follow it. The state sequences that follow a particular state s depends on the policy executed. First we define utility with respect to a specific policy π as,

$$U^\pi(s) = E\left[\sum_0^\infty \gamma^t R(s_t)|\pi, s_0 = s\right] \tag{4}$$

where s_t is the state after executing π for t steps. Using this definition, the *true* utility of a state, $U^{\pi*}(s)$is defined as the expected sum of discounted rewards if an optimal policy is executed. The presence of the expectation operation in the past equation is needed because the general model account for *unreliable* actions, i.e. actions that transition to an unexpected state with some non-zero probability. In our application, all our actions are modeled as deterministic but we use the general form of the algorithm. Also, notice that the sum in 4 is infinite when $\gamma = 1$. Assigning a value of 1 to γ implies that rewards obtained now have the same importance (weight) as those received later in the process therefore, the best policy

could encourage to wander around the terminal state and never finish. Mathematically, the sum in equation 4 does not converge.

It was stated above that the utility of a state is the expected sum of discounted rewards from that point onwards, therefore the utility of a state is related directly to that of its neighbors: the utility of s is the immediate reward for that state plus the expected discounted utility of the next state assuming that the agent chooses the optimal action. Mathematically,

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U(s) \tag{5}$$

where $R(s)$ is the reward at state s, s' stands for all possible neighbors of s, a is an action and $T(s, a, s')$ represents the transition model.

The set of N simultaneous equations that result from having N states, are non-linear. Therefore we rely on an iterative algorithm to find a solution.

The algorithm, called *Value Iteration*, starts with arbitrary initial values for the utility of each state and then updates the utility of each state with the information of the utility of its neighbors. The update performed at each iteration can be expressed as,

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U_i(s') \tag{6}$$

In other words, the algorithm propagates utility information through the state space by means of local updates.

4.1 Algorithm

The algorithm is specified below in pseudocode using syntax similar to that of a Matlab function.

function policy = Value-Iteration(mdp,ϵ)
%inputs: mdp - An Mdp with states S,
% transition model T,
% reward Function R,
% discount factor γ
% ϵ - maximum error allowed in the utility of any state
U_1 = vector of utilities for states in S, initially zero
U_2 = vector of utilities for states in S, initially zero
policy = vector that stores the optimal action for each state
δ = the maximum change in the utility of any state in an iteration
while $\left(\delta < \frac{\epsilon(1-\gamma)}{\gamma}\right)$
U1 = U2
$\delta = 0$
for each state s in S
if(s is goal state)
$U_2(s) = maxReward + \gamma \max_a \sum_{s2} T(s, a, s2)U_1(s2)$
else

$U_2(s) = R(s) + \gamma \max_a \sum_{s2} T(s, a, s2) U_1(s2)$
end
$policy(s) = \arg\max_a \sum_{s2} T(s, a, s2) U_1(s2)$
if $|U_2(s) - U_1(s)| > \delta$
$\delta = |U_2(s) - U_1(s)|$
end
end
end
end

5 System Architecture

A high level overview of our system is shown in Fig. 1. We describe the different components and how they work together in this section.

5.1 Training Data

The input to the music learner is a collection of chorales composed by Johann Sebastian Bach. Musical compositions are input as MIDI files. MIDI is a format that represents music as a set of messages that instruct a MIDI-compatible device to play specified notes of music at specified time instants for specified durations.

5.2 Encoder Module

Each piece is translated into a trajectory of states in a Markov Decision Process (MDP) as defined in Sec. 2. The translation of MIDI messages into states under our MDP representation is preformed one message at a time sequentially by the encoder module. However, not all metadata required by our MDP is available in the MIDI format. Hence, we manually add information such as chord root, chord type, relative position (within piece and within phrase) to each state generated by the encoder. The action between any two states in the MDP is generated automatically by computing the difference between their state variables. The result of encoding a piece of music is a trajectory usable by the training module. In our implementation, we used of the MatLab MIDI toolkit developed by Ken Schutte [8] for front-end MIDI parsing.

5.3 Training Module

Training of the system is based on the Apprenticeship Learning algorithm described in Sec. 3. Given the set of observed states, actions and reward features, the training module produces the optimal policy whose feature expectations come closest to the feature expectations of the observed expert (computed from the training data). Each iteration uses reinforcement learning to produce a policy that is closer to the hidden policy assumed to be used by the expert.

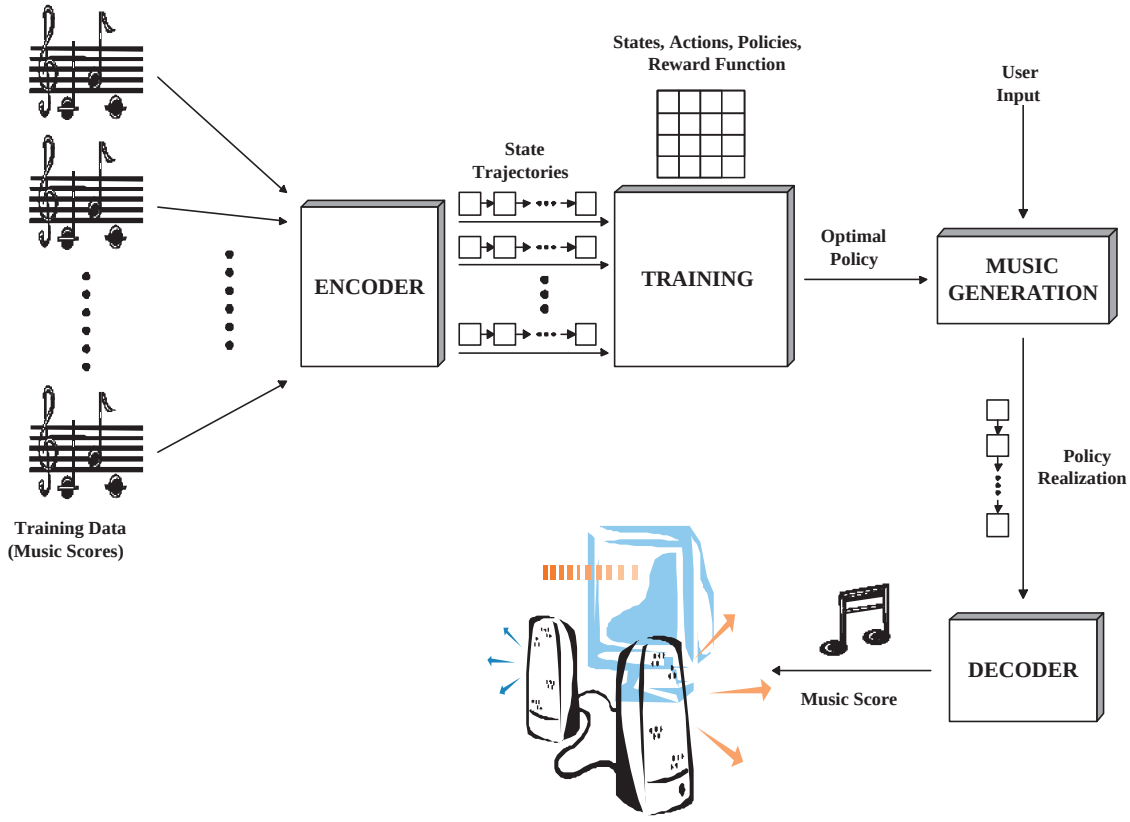

Figure 1: System Architecture

5.4 Music Generation Module

Upon receiving the optimal policy from the training module, the music generation module realizes this policy, i.e., produces a state trajectory which can used to generate a music score. To realize a policy, the system requires a starting state which is a user input. Transitions from this state to the next and between subsequent states are dictated by the policy and the MDP. All states and actions taken must have occurred in the training data and music generation terminates when the last state reached has no valid successor. The produced trajectory, which replicates the music style of the observed expert is then decoded into actual music.

5.5 Decoder Module

The trajectory of states received from the music generation module is converted into a music score in the decoder module. Since we use the MIDI format to represent music, we convert this trajectory into a sequence of MIDI messages by computing their start times, end times, durations and pitches. These messages are then ordered by start time start time and written out to a MIDI file. The file generated can be played by any standard MIDI player such as a PC with a MIDI-enabled sound card.

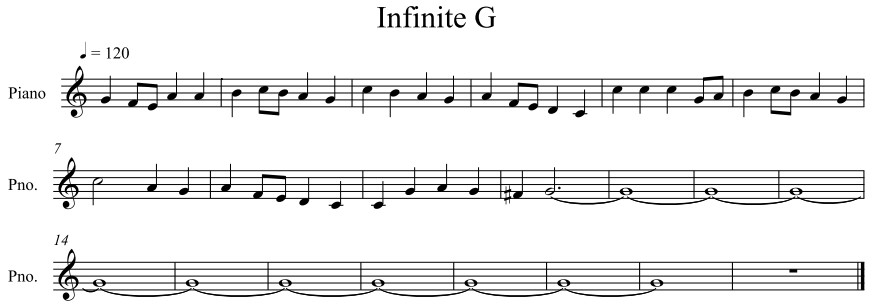

Figure 2: One example of music generated by the system. This piece shows the possibility of a loop in the policy.

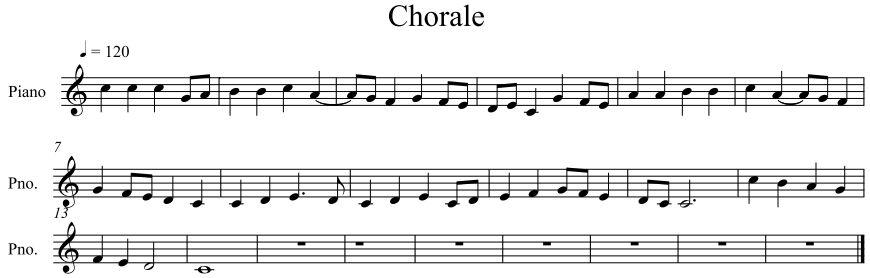

Figure 3: Another example of music generated by the system.

6 Evaluation

In this section we present a qualitative evaluation of the performance of our system. The data was encoded both manually and routines were written in Matlab to extract some features automatically from midi files. The pitches were read from midi files; we input the harmony and phrasing, and the other state variables were computed automatically. The system was trained using eleven chorales in major keys. Although more data is available, the encoding of each piece is tedious and time consuming. Time constraints did not allow us to enlarge our corpus of training data. Figures 2 and 3 show pieces generated by the system.

6.1 Melody

Although portions of the melody can be related to themes in the training data, the generated chorale is original. The AL algorithm was advanced enough to put the portions together in a way that makes musical sense, rather than just arbitrarily combining portions of the training pieces. For instance, the first fives beats of Figure 3 are the same as the first five beats of Chorale 302. Nonetheless, the algorithm smoothly transitions to original music before returning to copying small characteristic portions from seen chorales.

The overall shape of the melody of the generated chorales is characteristic of Bach's work. The piece mostly stays in the key, except for characteristic deviations. For example, the only note in Figures 2 and 3 not in the key of the piece is the F# in measure , 10 of Figure 2. However, this is a classic V/V progression—one of the most common methods of introducing notes outside the key used in the Baroque style.

Note that many of the features learned from Bach's chorales are not unique to Bach and are in fact characteristic of the Baroque style in which he composed.

Figure 2 also shows one disadvantage of the current approach. Deterministic policies allow for infinite cycles. In this particular case, the policy states to stay in the same state of holding the G in measure 10, and there currently exists no way to get to another state. Similar situations can happen with note sequence repeating infinitely.

Although our focus is on creating a stylistic melody, other aspects were also captured by the system. The harmony seems characteristic of the chorales seen, and accordingly support the melody. The rhythm is also typical of that found in the training data. One thing that stands out is the predominant use of quarter notes and that the eighth notes occur primary on the weak beats, 2 and 4, of 4/4 time signature.

7 Conclusions

We used pieces of music, namely Bach's Chorales, and the Apprenticeship Learning algorithm to learn the compositional style of the music composer, Johann Sebastian Bach. We then generated a piece of music that reflects this style of music composition. We chose this composer for his distinctive style and chose his chorales because they are relatively short, simple and available to us in digital form.

The complexity of encoding this problem entails did not allow us to train with a large number of chorales, but the results obtained were satisfactory for the training data used. With more time and computational power, the results could be even better.

In one of the pieces we observed a problem that caused a phrase to be repeated ad infinitum. This was because we deterministically chose the final policy generated by reinforcement learning and also because of a design choice to select MDP actions deterministically. The infinite loop encountered could have been escaped if multiple actions are available to choose from from any state.

8 Description of Individual Effort

Steve implemented the various components of the Apprenticeship Learning Algorithm and devised an efficient representation of states and actions of the training data. He also used Principal Component Analysis to reduce the number of reward features required. Joel implemented the Reinforcement Learning component of the training module that generates the optimal policy for music generation. Magesh worked on the policy realization of the apprenticeship learning algorithm, as well as the MIDI encoding and decoding modules. He also obtained the training data in music and sheet forms. We collectively defined the states, actions and reward features to use in the training module and wrote test cases. We collectively worked on the design, integration, and refinement of all components.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, Banff, Canada, 2004.

[2] Pieter Abbeel and Andrew Y. Ng. Exploration and Apprenticeship Learning in Reinforcement Learning. In *Proceedings of the Twenty-second International Conference on Machine Learning*, 2005.

[3] David Cope. *Computers and Musical Style*. Oxford University Press, 1991.

[4] Shlomo Dubnov, Gerard Assayag, Olivier Lartillot, and Gill Bejerano. Using Machine-Learning Methods for Musical Style Modeling. *IEEE Computer*, 2003.

[5] Andrew Y. Ng and S. Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[6] George Papadopoulos and Geraint Wiggins. AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects.

[7] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Second edition, 2003.

[8] Ken Schutte. MIDI and Matlab: http://www.KenSchutte.com/midi/.

9 Glossary

- *Eighth Note*: An eighth note is a musical note played for one eighth the duration of a whole note.
- *Phrase:* In music a phrase is a section of music that is relatively self contained and coherent over a medium time scale. In common practice phrases are often four and most often eight bars, or measures, long.
- *Key of a Piece:* A piece of music is said to be in a particular key if it is harmonically centered on that particular note. For example, a piece can be in the key of C, or in the key of $F\#$.
- *Pitch:* Pitch is the perceived fundamental frequency of a sound. In music, the label of a pitch is the letter corresponding to a musical note with that pitch, and optionally a number corresponding to the octave.
- *Chord:* In music and music theory a chord is three or more different notes that sound simultaneously. For example, *C major* is a chord that consists of notes C, E and G. Other examples of chord types include *minor*, *major 7*, *diminished*, *half diminished 7* and *fully diminished 7*.

- *Melody:* A melody, also tune, voice, or line, is a series of linear events or a succession, not a simultaneity as in a chord (harmony).
- *Harmony:* Harmony is the use and study of different pitches occurring simultaneously, and chords, actual or implied, in music.
- *MIDI:* Musical Instrument Digital Interface (MIDI) is an industry-standard protocol that enables electronic musical instruments, computers and other equipment to communicate, control and synchronize with each other.