Dissertation

Title: The Use of Apprenticeship Learning Via Inverse Reinforcement Learning for Musical Composition

Student: Orry Messer
Student Number: 0605213d
Supervisors: Pravesh Ranchod, Prof. van Alten

Date: 14 August 2014

Abstract

Reinforcement learning is a branch of machine learning wherein an agent is given rewards which it uses to guide its learning. These rewards are often manually specified in terms of a reward function. The agent performs a certain action in a particular state and is given a reward accordingly (where a state is a configuration of the environment). A problem arises when the reward function is either difficult or impossible to manually specify. Apprenticeship learning via inverse reinforcement learning can be used in these cases in order to ascertain a reward function, given a set of expert trajectories. The research presented in this document used apprenticeship learning in order to ascertain a reward function in a musical context. The agent then optimized its performance in terms of this reward function. This was accomplished by presenting the learning agents with pieces of music composed by the author. These were the expert trajectories from which the learning agent discovered a reward function. This reward function allowed the agents to attempt to discover an optimal strategy for maximizing its value.

Three learning agents were created. Two were drum-beat generating agents and one a melody composing agent. The first two agents were used to recreate expert drum-beats as well as generate new drum-beats. The melody agent was used to generate new melodies given a set of expert melodies. The results show that apprenticeship learning can be used both to recreate expert musical pieces as well as generate new musical pieces which are similar to with the expert musical pieces. Further, the results using the melody agent indicate that the agent has learned to generate new melodies in a given key, without having been given explicit information about key signatures.

Declaration

I, Orry Messer, hereby declare the contents of this dissertation to be my own work. This document is submitted for the degree of Master of Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Signature: ______________________________ Date: ______________________________

Acknowledgements

I would like to thank my supervisors, Pravesh Ranchod and Prof. Clint van Alten. Their insight and support throughout this research has been invaluable.

Contents

Chapter 1

Introduction

Different musicians approach the task of writing music in different ways. Whilst they may use the same instruments, the actual music that they produce can and should be completely new. There are as many ways of creating music as there are musicians, and the wide variety of musical genres and the differences within these genres show just how diverse music is. The way in which a musician creates a sound, the style of the musician, is influenced by many factors, though it seems natural that the music the musician listens to is one of the most influential factors.

The task this research addresses is that of using machine learning techniques (specifically apprenticeship learning via inverse reinforcement learning [Abbeel and Ng 2004]) to generate music. The use of apprenticeship learning stems from the idea that musical style is difficult to manually specify and that one gets the best sense of a musician's style by listening to the musician play. It is assumed in this research that a musician has a unique internal reward system which dictates how and when the musician will strike a note, how long the musician will hold the note for, whether or not he will play that note in conjunction with another and many other musical factors. In short, it is assumed that this internal reward system governs how the musician plays. When the musician plays, he is therefore producing the output which maximizes his reward according to this reward system. The use of apprenticeship learning in this context is thus an attempt by a learning agent to learn this internal reward system and thereafter use this system to generate music.

The way the reward system is learned by the agent is by providing the agent with *expert trajectories*. These expert trajectories are example musical pieces from which the learning agent can find the reward function which led the musician to create the music that he did. The aim was to have the learning agent generate new music by attempting to maximize the discovered reward function. Three learning agents were created for this task: Rhythm Agent 0 (RA-0), Rhythm Agent 1 (RA-1) and the Melody Agent (MA). RA-0 and RA-1 were tasked with generating drum-beats while MA was tasked with generating melodies. The results of the experiments run with these learning agents show that these agents are capable of recreating expert trajectories, when only one expert trajectory is presented to them, as well as creating new musical pieces when multiple expert trajectories are presented to them.

A similar study by Acevedo *et al.* [2007] found that by using apprenticeship learning via inverse reinforcement learning a learning agent could learn the style of the music composer Johann Sebastian Bach and generate a piece of music which reflected this. This is further expounded upon in Section 2.5.

The rest of this document is structured as follows: the following chapter, Chapter 2 provides the background and related works. The research methodology and implementation details common to the three learning agents are discussed in Chapter 3. Chapters 4, 5 and 6 discuss RA-0, RA-1 and MA, respectively. Chapters 4, 5 and 6 each contain implementation details specific to the agent as well as results of the experiments performed using that agent. A short concluding chapter, Chapter 7, ends this document.

Chapter 2

Background and Related Work

2.1 Introduction

The work presented in this paper made use of machine learning techniques for music generation. Reinforcement learning, a form of unsupervised machine learning in which the agent learns by interacting with its environment, was employed in this research. The agent's goal is to maximize the reward it receives from the environment it is interacting with. Although the reward function is considered external to the learning agent (it is part of the environment), it is the implementer of the algorithm who must specify the reward function. In the case of this research, the reward function is difficult to manually specify. It is not apparent what the reward signal for musical composition should be. Thus, apprenticeship learning via inverse reinforcement learning [Abbeel and Ng 2004] was used to infer the reward function from expert trajectories. An expert trajectory is an example of expert behaviour within a certain context, in this case a musical context. Apprenticeship learning requires the agent to first distil a reward signal from a set of expert observations. Only then can reinforcement learning commence.

This chapter presents work relevant to this research starting with Section 2.2.1 which discusses the elements of music as well as some fundamental music theory. Following this, Section 2.2.2 gives a brief overview of the intertwined history of music and mathematics. Section 2.3.2 then discusses *states* and *actions* which refer to configurations of an environment and ways in which the agent can behave, respectively. The following section, Section 2.3.3 then discusses the Markov property as well as Markov decision processes (MDPs). Sections 2.3 and 2.4 then discuss reinforcement learning and apprenticeship learning via inverse reinforcement learning, respectively, followed by Section 2.5 which presents work done using reinforcement learning as well as apprenticeship learning via inverse reinforcement learning within a musical context. Finally, Section 2.6 provides a conclusion to this chapter.

2.2 Music

2.2.1 The Elements of Music

As with any art form, there are certain fundamental elements which are used in the compositional process. The fundamental elements of music - *rhythm*, *harmony*, *melody* and *tone colour*, and the *forms* which result from their interaction with one another constitute the basics from which musical pieces are created [Boyden 1971].

The sounds which can be produced by instruments and voices are discretized into 12 *notes* which are repeated over many *octaves*. Rhythm can be defined as the succession of these notes with respect to time [Ammer 1995]. This describes how fast the notes move (the *tempo*) as well as the pattern of long and short notes. The melody refers to the *choice* of notes which are played. It is impossible to speak of melody without rhythm, since the choice of notes which make up the melody will have a characteristic rhythm which defines how quickly they are played and how long each note is held for. Harmony is the pattern of *chords* played in a composition. A chord is a group of two or more notes sounded simultaneously. Finally, tone colour refers to the acoustic properties of a particular instrument. For a example, a note played on a guitar will sound different to the same note played on a piano.

When arranged from lowest pitch to highest pitch, the distance between each of the 12 notes is known as a *semitone* (or half tone). A particular selection of notes arranged in rising order of pitch is known as a *scale*. When all 12 notes are selected this is known as the *chromatic scale* (Figure 2.1). In this case each note is a half tone away from its neighbouring note in the scale. Most western music is derived from the *major scale*.

What defines a scale is the distance between each of its successive notes. A major scale is defined according to the following pattern: *tone, tone, semitone, tone, tone, tone, semitone*, where one tone equals two semitones. What this means is that the distance between the first, second, fourth, fifth and sixth pair of notes is a tone, while the distance between the third and seventh pair of notes is a semitone. For example, consider the C major scale (that is, the major scale starting on C). According to the major scale pattern, the next note of the scale is a tone away. As can be seen from Figure 2.1, increasing C by a semitone brings us to $C\sharp$. Raising it by yet another semitone (to get to the full tone) gives us a D. Thus, the second note in the C major scale is a D. The third note, which is found by raising the second note by a full tone will be an E. The fourth, a semitone higher than the third is an F. The fifth, a tone higher than the fourth is a G. The sixth, a tone higher than fifth is an A. This is because the notes wrap around — A follows $G\sharp/A\flat$. The seventh, a tone higher than the sixth is a B. Finally, the eighth note (which is one full octave higher than the first note) is a semitone higher than the seventh and is, like the first note, also a C. The pattern between the successive pairs of notes of a scale define the kind of scale it is. As was just illustrated the major scale is defined by a particular pattern. When given a starting note, such as C, the rest of the scale can be derived, given the pattern.

A	$A\sharp/B\flat$	B	C	$C\sharp/D\flat$	D	$D\sharp/E\flat$	E	F	$F\sharp/G\flat$	G	$G\sharp/A\flat$

Figure 2.1: One octave of the chromatic scale. The symbols $\sharp$ and $\flat$ (sharp and flat) modify a note by either raising the note by a semitone or lowering it by a semitone, respectively. Thus, $A\sharp$ and $B\flat$ are the same note (they are said to be *enharmonic*). Note that $B\sharp$ is C and that $E\sharp$ is F (conversely, $C\flat$ is B and $F\flat$ is E).

C	D	E	F	G	A	B	C

Figure 2.2: One octave of the C major scale

Music is arranged into *measures*. Each measure is comprised of multiple *beats*, measurements of musical time. If there are eight measures in a musical piece, and each measure contains eight beats, there will be $8 \times 8 = 64$ beats in that piece.

The next section discusses the interrelated history of music and mathematics as well as computational approaches to music.

2.2.2 A Brief History Music and Mathematics

The idea that mathematics and music are intertwined dates back to the Pythagoreans whose concept of the ubiquity of numbers is expressed in their assertion that either all things are numbers, all things are furnished with numbers or that all things behave like numbers [Diaz-Jerez 2000] [Papadopoulos and Wiggins 1999]. It was in fact Pythagoras who discovered the numerical ratios of the musical intervals [Diaz-Jerez 2000], that is the distances between pair of notes [Ammer 1995].

The earliest known application of algorithmic method to music composition is attributed to Guido D'Arezzo [Diaz-Jerez 2000], the Italian medieval music theorist, who invented staff notation as well as solmization, the mnemonic device of assigning syllables to each note (do—re—mi—fa—sol—la—ti - originally *ut*—re—mi—fa—sol—la—ti). What D'Arezzo did was to set out a 2 octave tessitura (a range of notes) and lay 3 iterations of the vowels "a—e—i—o—u" below the notes as follows:

Γ	A	B	C	D	E	F	G	a	b	c	d	e	f	g	a
a	e	i	o	u	a	e	i	o	u	a	e	i	o	u	a

The capital Greek gamma, Γ, is from the set of notes encompassing the octave below middle C. The upper case Latin letters represent the octave encompassing middle C, and the lower case Latin letters represent the octave above middle C. A text was then taken from which the vowels were extracted. The corresponding note values were then used in the composition of a melody. Since each vowel corresponds to 3 different pitch values, 4 in the case of "a", the composer then chose the note values which conformed the most to the stylistic norms of the time. Although this is a simple example of algorithmic composition, it does give some insight into the sheer size of the scope of musical composition: assuming for now that each vowel can take on only 3 values, a text with only 15 vowels, corresponding to a 15-note melody has 3^{15}, over 14 million, possible melodies which it can take on. There have been other examples throughout history of algorithmic composition, such as Mozart's Musical Dice Game [Diaz-Jerez 2000], but the birth and growth of computing in the twentieth century has allowed for much more powerful ways of using algorithms to compose music, given that the numerous tedious calculations required can now be handled by a computer.

Although the early enthusiasm regarding computational methods for musical composition was met with frustration [Conklin 2003], there have been numerous approaches to the task. The following methods have all at some point been tried:

- Grammars [Baroni and Jacoboni 1978]
- Genetic algorithms [Horner and Goldberg 1991]
- Chaos Theory [Little 1993]
- Cellular Automata [Miranda 1993]
- L-Systems [Nelson 1996]

For a more comprehensive list refer to Diaz-Jerez [2000]. Although a classification of the various techniques is somewhat moot, as many are equivalent, it does give insight into the multitude of approaches which have been tried. Of particular relevance to this research are mathematical models, since Markov Decision Processes, (MDPs) were used in this research to model the musical environment. Markov decision processes are discussed further in Section 2.3.3

A similar mathematical model, the Markov chain has been used in the past for musical composition. Markov chains, which share the Markov property (discussed in Section 2.3.3) with Markov Decision Processes, were first used for musical composition by Lejaren Hiller and Leonard Isaacson in their 1957 *Illiac Suite* [Ames 1989]. Two years later composer Iannis Xenakis used Markov chains in creating what he called *Stochastic Music*. Further, *M* and *jam factory* [Papadopoulos and Wiggins 1999], two automated composition music programs used Markov chains in their working. These are the first two programs of this kind which met any sort of commercial success. Reinforcement learning, a form of machine learning, has also been directly used in musical composition. A discussion of research into this is delayed until Section 2.5. Reinforcement learning itself is discussed in the next section.

2.3 Reinforcement Learning

2.3.1 Introduction

Reinforcement learning is a form of unsupervised machine learning wherein the learning agent interacts with the environment in order to achieve a goal [Sutton and Barto 1998]. This computational approach to learning via interaction relies on the idea that the learning agent must select actions in an uncertain environment in order to estimate how good those actions are; learning by doing is one of the core ideas of reinforcement learning. Taking actions which do not only look to maximize immediate reward, but rather maximize future rewards is an important feature of reinforcement learning. The agent seeks to maximize future rewards by directing itself to states (configurations of its environment) from which more reward can be gained; that is, it tries to direct itself to states with high *value* (a discussion on states and actions is presented in Section 2.3.2. The meaning of the term "value" is explained in Section 2.3.4). These two ideas, that the agent must interact with its environment and the idea of delayed rewards, distinguish reinforcement learning from other branches of machine learning.

The learner behaves according to a **policy** — a mapping from state-action pairs to probabilities of selecting each action. A policy may be deterministic in which case the policy is a mapping from each state to a corresponding action. A policy is denoted by the letter π.

To maximize the cumulative discounted reward it receives (which is the goal of learning), the learning agent attempts to ascertain the optimal policy — this is the policy which maximizes the expected cumulative discounted reward from a given state. The cumulative discounted reward at time t is known as the **return**, R_t. This is shown in Equation 2.1.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad 0 \leq \gamma \leq 1 \tag{2.1}$$

Note here that this is the actual return received by the learning agent. If the environment's dynamics were known, this return could be calculated a priori. What the learning agent attempts to uncover is a policy which will direct the agent in such a way that R_t is maximized. Of course, this problem is non-trivial as the dynamics of the environment can be (and usually are) complex, unknown and laden with uncertainty.

In the above equation, γ is the *discount rate* which is between 0 and 1, is used in order to ensure that the sum in 2.1 has a finite value. The closer γ is to 1 the more far-sighted the agent becomes, weighing future rewards more heavily. If γ is 0 then the agent concerns itself only with immediate reward, not taking into account any future rewards received.

2.3.2 States and Actions

The environment with which the agent interacts can be in one of a great many configurations. A particular configuration of the environment is known as a **state**. The agent can affect the configuration of the environment. An agent affects the state by performing an **action** - this changes the configuration of the environment, the state (although it may be the case that the chosen action does not change the state).

The agent interacts with the environment by choosing *actions* which cause the environment to *transition* from one *state* to another. The action an agent chooses can lead to several different outcomes; i.e. a given choice of action in a particular state may lead to one of several resulting states. The different resulting states from the choice of action and their respective probabilities of occurring (the *transition probabilities*) are one of the defining characteristics of the environment and are discussed further in the next section. The agent chooses its actions based on a *policy*; a mapping from states to actions. The goal of reinforcement learning is to find an optimal policy with which the agent can navigate its environment.

The state of the environment is perceived by an agent as a state signal. For example, the current positions of all the pieces on a checkers board could constitute a state signal for a checkers playing agent. In a musical setting, the state signal could be the current measure and the current beat within that measure. The actions available in the checkers setting are all the legal moves available to the agent, given the current configuration of the board. In the musical setting, the actions correspond to the range of notes available to the agent, one of which the agent must choose.

The next section discusses what is ideally required of a state signal (Section 2.3.3.1) as well as the formal framework within which the decision making process is carried out (Section 2.3.3.2).

2.3.3 The Markov Property and Markov Decision Processes

2.3.3.1 The Markov Property

One key difference between the two examples given in Section 2.3.2, is that in the checkers case, all relevant information is preserved by the state signal. In the checkers example, where the state signal was the position of all the pieces on board, it does not matter how we got to the current board configuration. The moves which led up to that state are irrelevant; what is relevant is only the current configuration, on which the agent bases its next move. In the music example, where the state signal was the current time step, it is not at all clear that this is enough information upon which to take the next action. The state signal does not include the previous notes which have been played. The previous notes played give a musical context upon which the choice of next note depends.

The question is then, how many previous notes do we give a learning agent in order for it to have enough musical context? Computational limitations aside, it is not at all clear, as it is with checkers, what the state signal should be in order to preserve all relevant information. The state signal in the checkers example has the "independence of path" property [Sutton and Barto 1998]. Such states signals and environments are said to be Markov, or have the Markov property. The state signal in the music example, does not, as the next note to be played depends on the sequence of notes which led up to it.

If the environment has the Markov property then the probability of the next state being s' and the next reward being r depends only on the previous state s and action a, as shown by the probability shown in 2.2. In the case where the Markov property does not hold, we have that this probability depends on the sequence of all past events, as shown by the probability shown in 2.3

$$\mathcal{P}r\left\{s_{t+1} = s', r_{t+1} = r \middle| s_t, a_t\right\} \tag{2.2}$$

$$\mathcal{P}r\left\{s_{t+1} = s', r_{t+1} = r \middle| s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\right\} \tag{2.3}$$

[Sutton and Barto 1998]

The next section discusses Markov decision processes, in which states are assumed to be Markov.

2.3.3.2 Markov Decision Processes

A Markov decision process (MDP) is a mathematical formulation of an environment in terms of states, actions and transition probabilities. The theory behind reinforcement learning is firmly rooted in having the learning agent perform in a Markov decision process. Ideally, the formulation of the agent's environment should be a Markov decision process, although in practice this is often not the case. In such cases the environment is approximated as a Markov decision process.

Formally an MDP is a tuple (S, A, T, γ, D, R), where S is the set of states, A is the set of actions, $T = P_{SA}$ is the set of state transition probabilities, $\gamma \in [0, 1)$ is a discount factor (discussed further in the next section), D is the initial state distribution and R is the reward function. In the case where the reward function is unknown, MDP\R is used to denote an MDP without a given reward function.

The transition probabilities mentioned above define the probability of getting to a resulting state s' from a current state s given that a particular action a has been taken. This is shown in Equation 2.4.

$$\mathcal{P}^a_{ss'} = \mathcal{P}r\left\{s_{t+1} = s' \middle| s_t = s, a_t = a\right\} \tag{2.4}$$

[Sutton and Barto 1998]

Further, states are associated with rewards. These rewards direct learning; certain states have a positive reward and are thus desirable states for the agent to be in, whilst others may carry a negative reward to convey that they are poor states. States may also have a reward of 0. Since taking an action can lead to one of several resulting states, the notion of an *expected* reward, given a particular action in a particular state, is needed to estimate the value of the next reward. This is shown in Equation 2.5 [Sutton and Barto 1998].

$$\mathcal{R}^a_{ss'} = E\left\{r_{t+1} \middle| s_t = s, a_t = a, s_{t+1} = s'\right\} \tag{2.5}$$

The most important aspects of the MDP are given by $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$.

The objective of the learning agent is to navigate the MDP such that it maximizes the cumulative reward it receives. In the reinforcement learning setting, the *value* of a state under a particular policy is more important than the state's immediate reward. The value of a state takes into account not only the immediate reward from that state, but also the expected future rewards resulting from being in that state, under a particular policy. The goal of learning then, is to maximize not the immediate *reward*, but rather long term *return*. The long term return is dependent upon the policy which the agent is using as its basis for action decisions. Thus, the goal of learning can be said to be to find a policy which maximizes the return of the learning agent. A policy is associated with a value function which is a mapping from a state to a number. This number is the value of the state, under a particular policy. Under a different policy, the value may (and likely will) be different, since the action selection rule from that state onwards will differ according to the different policy. The following section discusses the value function in more detail.

2.3.4 The Value Function And Action Value Function

The value function for a policy π in a certain MDP is formally defined as follows:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \tag{2.6}$$

where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is the return (that is, the cumulative discounted reward), following time t. The value function is an estimate of how good it is to be in a particular state; it takes into account the expected immediate reward from a state and the expected future rewards which will be received if the agent follows policy π from then on. The value function is associated with a policy, π since that policy dictates the choice of actions which leads the agent to the states it will visit and therefore the rewards it will receive. The expectation is taken since the environment may be stochastic - an action taken may with differing probabilities lead to different states, each with its own immediate reward. Further, the policy π may itself be stochastic; π may be a mapping from states to probabilities of selecting actions, as opposed to being deterministic where π is a direct mapping from a state to an action. The weighted average of these states is thus taken. The value of a state depends on the weighted average of the values of its successor states, and similarly the values of the successor states depend on the expected values of their successor states and so forth. This recursive relationship between the value of states and of their successor states is expressed in Equation 2.7 and is known as the *Bellman Equation*.

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right] \tag{2.7}$$

It is the job of the learning agent to estimate the value function of a given policy, as it is typically intractable to calculate it exactly. This process is known as *policy evaluation* [Sutton and Barto 1998]. The purpose of evaluating a policy is so the agent has some way of knowing "how good" it is to be in a particular state. Once this is known, it can base its action selection on the relative values of each action choice in a given state, s. For this reason an action value function is used and evaluated (Equation 2.8).

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \tag{2.8}$$

The action-value function associates to each state the value of each action from that state, under a given policy. A class of algorithms called Generalised Policy Iteration (GPI) is used to both evaluate a given policy, and thereafter improving upon it, by using an action selection rule which will with high probability choose the action with the highest value.

2.3.5 Exploration/Exploitation Dilemma

The two distinguishing features of this type of reinforcement learning, delayed rewards and learning from interaction, introduce challenges; namely the dilemma between exploiting a known good action and exploring other actions which may yield higher rewards. Since the learning agent can only estimate how good an action is by trying it, it might try an action (once or several times) and find that the action chosen was a very poor one. In order to gain the knowledge that this is indeed a poor action, the agent must take an action which results in a poor outcome. Suppose now that the agent was faced with two choices in a particular situation. The first choice has been tried before (once or several times), and is known to lead to good future rewards. The second action, which it has not tried, may lead the learning agent to situations where even greater rewards are obtained, but may also lead it to poorer rewards. How then should the agent behave? Should it *exploit* its current knowledge and select the action it knows will lead it to high future rewards, or should it *explore* its environment and try find an even better action? To get lots of reward it must exploit its experience, but to gain that experience it must explore its environment. This dilemma between exploration and exploitation is characteristic of reinforcement learning and the exploration rate is a parameter which the designer must tune. How this is done depends on the problem itself and is usually found by experimentation [Sutton and Barto 1998].

The goal of the learning agent is to maximize its return, which depends on the reward function. By crafting the reward function appropriately and by using one of the reinforcement learning methods which seek to maximize the total return, the agent will learn its task. In order to maximize its return, the agent needs to have some estimate of the *value* of a particular state. The estimated value of a state gives the agent some notion of how good the state is, in terms of the task set out for it. The following section discusses Sarsa, a reinforcement learning algorithm which can be configured to make the learning agent more prone to exploring its environment or to exploiting its current knowledge.

2.3.6 Sarsa

The Sarsa algorithm falls into the class of GPI algorithms. What this means is that two procedures are in play; the first is policy evaluation and the second is policy improvement. The Sarsa algorithm is shown in Figure 2.3.

Figure 2.3: The Sarsa Control Algorithm

```
1: Initialize $Q(s, a)$ arbitrarily
2: **loop** (for each episode):
3:     Initialize $s$
4:     Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
5:     **repeat** (for each step of episode):
6:         Take action $a$, observe $r$, $s'$
7:         Choose action $a'$ from $s'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
8:         $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
9:         $s \leftarrow s'$; $a \leftarrow a'$;
10:    **until** $s$ is terminal
11: **end loop**
```

The goal of this algorithm is to find the optimal action value function. This is usually not possible and we have to settle for a close approximation of this function. After the action value function is initialized arbitrarily (see line one of Figure 2.3), there are two loops which control the flow of this algorithm. The outer loop simply iterates through the number of episodes which the implementer of the algorithm has defined. An episode is simply one run through the environment, from start state to terminal state. The second loop controls the behaviour intra-episode. That is to say, at each state the following happens: an action is selected according to some action-selection rule (in this case ϵ-greedy) and a reward is observed which is used to better the estimate of the action value of the state action pair which led to the reward. An ϵ-greedy selection mechanism is one in which ϵ is the fraction of the time where a random action is selected and $(1 - \epsilon)$ is the fraction of the time where the action with the highest value is selected, where $0 < \epsilon < 1$. ϵ is used to control the trade off between exploration and exploitation (see Section 2.3.5).

Line eight of the algorithm is where the estimate of the state action pair is modified. The modification is done by updating the estimate of the state value pair towards its latest observed value. The previous estimate is moved partially towards the difference between the old and new estimate.

2.3.7 Q-learning

Another algorithm which falls into the class of GPI algorithms is Q-learning. The goal of the algorithm is the same as that of Sarsa — to find the optimal action value function — but the approach taken is slightly different. Whereas Sarsa is an on-policy algorithm, Q-learning is off-policy. What this means is that Q-learning updates one policy while following another. The policy it updates is the optimal policy. The policy it follows, however, is the ϵ-greedy policy. Thus, the action value function learned by Q-learning is that of the optimal policy. In other words, otherwise "good" states do not get punished for random actions having been taken from them. Whereas Sarsa returns an action value function which has learned that random moves may be taken, and this affects the state, Q-learning only updates the optimal action. The Q-learning algorithm is presented in 2.4.

Figure 2.4: Q-learning

```
1:  Initialize $Q(s,a)$ arbitrarily
2:  **loop** (for each episode):
3:      Initialize $s$
4:      **repeat** (for each step of episode):
5:          Choose action $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
6:          Take action $a$, observe $r$, $s'$
7:          $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
8:          $s \leftarrow s'$
9:      **until** $s$ is terminal
10: **end loop**
```

In this research it was decided to use Q-learning over Sarsa, since Sarsa incorporates the random action, taken according to the ϵ-greedy algorithm into its calculation of the q-table. Q-learning was chosen since it was not desired to penalize a good state for a random, potentially bad action taken from it.

2.3.8 Softmax Action Selection

Another kind of action selection rule which can be used is *softmax* action selection. Like ϵ-greedy action selection, softmax is used to select the optimal action only part of the time. Unlike ϵ-greedy the suboptimal actions are not all equally likely to be chosen. The likelihood of an action being selected is relative to its value. The action with the highest probability of being selected is the one with the highest value. The second most likely action to be selected is the one with the second highest value, and so on. Using the Higgs, or Boltzmann, distribution is a common way to implement softmax [Sutton and Barto 1998]. This distribution is shown in 2.9.

$$\frac{e^{Q_{t-1}(a)/\tau}}{\Sigma_b e^{Q_{t-1}(b)/\tau}} \tag{2.9}$$

The Higgs, or Boltzmann, distribution is often used to implement softmax action selection. The probability presented in 2.9 represents the probability of selecting action a at time t. τ is a positive parameter called the *temperature*. Higher temperatures cause actions selection to be more equi-probable. As $\tau \to 0$ softmax becomes the same as greedy action selection.

2.3.9 Eligibility Traces

Eligibility traces are used in reinforcement learning in combination with learning algorithms such as Sarsa and Q-learning in order for learning agents to learn more efficiently [Sutton and Barto 1998]. What these traces do is provide a mechanism for rewards to propagate to previous states more efficiently. That is, instead of only the state which led to the reward being received being eligible for an update, several states which led to the reward are now eligible for update too. How many of the previous states are eligible is controlled by the parameter λ. The higher λ is, the more states become eligible for update. Eligibility traces were combined with Q-learning in this research. The modified Q-learning algorithm is presented in 2.5

Figure 2.5: Q-learning

```
1:  Initialize $Q(s,a)$ arbitrarily and set $e(s,a) = 0$, for all $s$, $a$
2:  **loop** (for each episode):
3:      Initialize $s$, $a$
4:      **repeat** (for each step of episode):
5:          Take action $a$, observe $r$, $s'$
6:          Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
7:          $a^* \leftarrow \arg\max_b Q(s', b)$ (if $a'$ ties for max, then $a^* \leftarrow a'$)
8:          $\lambda \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
9:          $e(s,a) \leftarrow e(s,a) + 1$
10:         **for** for all $s, a$ **do**
11:             $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
12:             **if** $a' = a^*$ **then**
13:                 $e(s,a) \leftarrow \gamma\lambda e(s,a)$
14:             **else**
15:                 $e(s,a) \leftarrow 0$
16:             **end if**
17:         **end for**
18:         $s \leftarrow s'$; $a \leftarrow a'$
19:     **until** $s$ is terminal
20: **end loop**
```

Clearly this algorithm, and the ones presented in Sections 2.3.6 and 2.3.7 rely on there being a reward function; all algorithms which solve the reinforcement learning problem rely on there being a reward function. This reward function is external to the learning agent; it is part of the environment but the implementer of the algorithm must still specify this reward function. It is the way in which the goal of the task is conveyed to the agent. There are times, however, when it is difficult or impossible to manually specify the reward function [Abbeel and Ng 2004]. The next section discusses a way in which to recover an unknown reward function and thereafter to learn based on it.

2.4 Apprenticeship Learning via Inverse Reinforcement Learning

Inverse reinforcement learning is the task of deriving a reward function from observed behaviour. The inverse reinforcement learning problem was characterized in Ng and Russell [2000] as follows: **Given** 1) measurements of an agent's behaviour over time, in a variety of circumstances, 2) if needed, measurements of the sensory inputs to the agent; 3) if available, a model of the environment.
Determine the reward function being optimized.

Once the reward function is learned it can be used for creating computational models for human and animal learning; this has applications in theoretical biology, econometrics and other fields [Ng and Russell 2000]. Another motivation for finding a reward function is the case of apprenticeship learning, where the ultimate goal is to find a policy which a learning agent can use to map states to actions. The observed behaviour in this setting is the behaviour of an expert. The learning agent uses the behaviour of the expert to ascertain a reward function. The learning agent then uses this reward function to find a policy which it can use to intelligently take actions in a particular domain. The goal in the apprenticeship learning case is to find a policy which the learning agent can use to behave successfully in its domain. By using inverse reinforcement learning the agent learns the reward function first which is the most succinct, robust and transferable definition of the task [Abbeel and Ng 2004]. Once a reward function is found, any suitable algorithm which solves the reinforcement learning problem may be used to find a policy.

There are various approaches to apprenticeship learning via inverse reinforcement learning. The approach taken by Abbeel and Ng [2004] was used due to its prevalence in the literature. Abbeel and Ng [2004] describes two methods for the inverse reinforcement learning step: the projection method and max-margin method. Note that "inverse reinforcement learning step" refers to the step of the algorithm which uncovers a reward function. Both have the same convergence guarantees (discussed further in Section 2.4.4). However, the projection method was shown to be slightly faster and is simpler to implement, as it does not require the use of a quadratic program solver. A further discussion of the differences between the max-margin and projection methods is deferred until Section 2.4.4 where the apprenticeship learning via inverse reinforcement learning algorithm is presented. The following section discusses the target of the inverse reinforcement learning step, the expert reward function.

2.4.1 Expert Reward Function

Shown in Equation 2.10 is the scalar result of the dot product of a set of weights, w and the feature vector of a state, $\phi(s)$. The inverse reinforcement learning step finds a weight vector which is used in determining a reward function.

$$R(s) = w \cdot \phi(s) \tag{2.10}$$

The feature vector, ϕ, contains the state variables which provide information about the environment's current configuration. If the agent is in state s_k, then the features of the state will be $\phi(s_k)$. The weight vector specifies how each of these state attributes must be traded off.

In this setting, it is assumed that we have access to expert trajectories. A **trajectory** is a walk through the state space — that is, a set of states. An **expert trajectory** is a walk through the state space performed by the expert. It is also assumed that the expert's reward function is made up of an optimal weight vector, w^* as in Equation 2.11.

$$R^*(s) = w^* \cdot \phi(s) \tag{2.11}$$

We hope to uncover a weight vector which guides a learning agent to perform as well as the expert, under the expert's unknown reward function $R^*(s)$. While the algorithm makes no guarantee that we uncover the expert reward function after it terminates, we are guaranteed to find a reward function from which we may find a policy via which the learning agent behaves similarly to the expert. In order to get a more precise understanding of what we mean by "behaving similarly to the expert," we must introduce the concept of **feature expectations** which are discussed in the following section.

2.4.2 Feature Expectations

The expected value of a policy π may be written as follows:

$$E_{s_0 \sim D}[V^{\pi}(s_0)] = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi\right] \tag{2.12}$$

$$= E\left[\sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t) \middle| \pi\right] \tag{2.13}$$

$$= w \cdot E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \middle| \pi\right] \tag{2.14}$$

Where s_0 is drawn from D, the set of starting states. The step from 2.12 to 2.13 follows from Equation 2.10. Equation 2.14 may be rewritten as:

$$E_{s_0 \sim D}[V^{\pi}(s_0)] = w \cdot \mu(\pi) \tag{2.15}$$

where

$$\mu(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \middle| \pi\right] \tag{2.16}$$

$\mu(\pi)$ is the vector of **feature expectations** of a particular policy. It is defined as the expected discounted accumulated feature value vector [Abbeel and Ng 2004]. It represents a discounted sum of the feature vectors which are expected to be seen when following a particular policy. In order to ascertain the feature expectations, given a policy, Monte Carlo methods may be used to sample trajectories, or if the dynamics of the environment are known and it is tractable, they may be computed [Abbeel and Ng 2004].

These feature expectations may be seen as aggregating the behaviour of an expert under a policy. $\mu(\pi)$ is the expected value of a policy without the weight vector. As the weight vector dictates how an agent is to trade off the various state attributes, the feature expectations can be seen as aggregating the agent's walk through the state space, removing any indication of "how good" that walk was. We wish to find a weight vector which induce an agent's walk so that it is close the the expert's feature expectations.

2.4.3 Expert Feature Expectations

The apprenticeship learning algorithm relies on knowing the *expert*'s feature expectations, μ_E. In practice an estimate for the expert's feature expectations, $\hat{\mu}_E$ is found empirically, given access to expert trajectories, which we assume we have. An expert trajectory is the expert's path through the state space: $\{s_0, s_1, \ldots\}$. Suppose there are m trajectories through the state space, then we have: $\left\{s_0^{(i)}, s_1^{(i)}, \ldots\right\}_{i=1}^{m}$. The empirical estimate for μ_E is given by Equation 2.17.

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{2.17}$$

As shown in the following section which presents the apprenticeship learning via inverse reinforcement learning algorithm in its entirety, the expert feature expectations are used both in finding a weight vector as well as in determining the stop condition of the algorithm.

2.4.4 Apprenticeship Learning via Inverse Reinforcement Learning: The Algorithm

Once the expert feature expectations have been estimated, the apprenticeship learning algorithm attempts to find a policy, $\tilde{\pi}$, such that $\tilde{\pi}$ induces feature expectations close to $\hat{\mu}_E$ [Abbeel and Ng 2004].

The apprenticeship learning algorithm for doing so follows:

```
 1: Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and
    set $i = 1$.
 2: Set $w^{(1)} = \mu_E - \mu^{(0)}$ and $\bar{\mu}^{(0)} = \mu^{(0)}$
 3: Set $t^{(1)} = ||w^{(1)}||_2$
 4: if $t^{(1)} \leq \epsilon$ then
 5:     terminate
 6: else
 7:     loop (while $t^{(i)} > \epsilon$):
 8:         Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using $R = (w^{(i)})^T \phi$
 9:         Compute $\mu^{(i)} = \mu(\pi^{(i)})$
10:         Set $i = i + 1$
11:         Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu^{(i-1)} - \bar{\mu}^{(i-2)})} (\mu^{(i-1)} - \bar{\mu}^{(i-2)})$
12:         Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
13:         Set $t^{(i)} = ||w^{(i)}||_2$
14:     end loop
15: end if
```

[Abbeel and Ng 2004]

Step 11 computes the orthogonal projection of μ_E onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$. The above algorithm finds a policy such that the agent's performance under it is close to that of the expert's under the unknown expert reward function, $R^*(s)$. Such a policy $\tilde{\pi}$ will have $||\mu(\tilde{\pi}) - \mu_E||_2 \leq \epsilon$, where ϵ is an arbitrarily small positive number. This means that the feature expectations which are induced by such a policy will be arbitrarily close to the feature expectations of the expert.

The goal is to find such a policy - one inducing feature expectations close to μ_E. Thus, before the algorithm begins, the expert's feature expectations must be estimated. This is done as explained in Section 2.4.3; that is, $\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)})$.

Once the expert feature expectations, μ_E, have been estimated the algorithm begins. The first step of the algorithm is to generate a random policy $\pi^{(0)}$ and to compute or estimate its feature expectations, $\mu^{(0)} = \mu(\pi^{(0)})$. The feature expectations $\mu^{(0)}$ can be computed according to the equation $\mu(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \middle| \pi \right]$ if the dynamics of the environment are known. If the dynamics are not known or it is intractable to calculate the feature expectations this way, runs through the MDP (trajectories) may be generated using this policy. From this new set of trajectories $\mu^{(0)}$ can be estimated in the same way that the expert feature expectations were estimated: by averaging the discounted sums of the features of the states which were seen in each of the trajectories. That is $\hat{\mu}^{(0)} = \frac{1}{k} \sum_{i=1}^{k} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)})$, if there were k trajectories generated by policy $\pi^{(0)}$.

The next step of the algorithm returns the weight vector, which characterizes the reward function. The weight vector is set as the component-wise difference between the expert feature expectations, μ_E, and the random policy's feature expectations, $\mu^{(0)}$ (or its estimate $\hat{\mu}^{(0)}$). I.e.: $w^{(1)} = \mu_E - \mu^{(0)}$. It is important to note that each step of the apprenticeship learning algorithm returns a weight vector which characterizes a reward function. From this reward function a policy is learned. The algorithm does not make any guarantees regarding on which iteration the "best" weight vector is returned. In this research policies generated from the entire run of the algorithm were considered, not just the final policy. In addition to this, $\bar{\mu}^{(0)}$ is set equal to $\mu^{(0)}$. $\bar{\mu}^{(0)}$ will be used in the next iteration of the algorithm. Following this step, t is returned. t is the 2-norm of the weight vector, w, which is used as the stopping condition for the algorithm, in the unlikely event that the random policy induces feature expectations close to those of the expert's.

Lines 7 to 14 contain the main loop of the algorithm. In it, the previous weights are used to solve the MDP and a new policy is returned (line 8). The new policy's feature expectations are calculated, and the counter is increased (lines 9 and 10). Lines 11 - 13 constitute the projection method, as described in Abbeel and Ng [2004]. This is the inverse reinforcement learning step, as the weight vector, which describes the reward function, is returned. As previously mentioned, the projection method replaces the max-margin method. The max-margin method is shown in Figure 2.6.

Compute $t^{(i)} = \max_{w:||w||_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.

Figure 2.6: The max-margin method

The max-margin version of the inverse reinforcement learning step involves both a maximization and a minimization. In the minimization, the minimum of the difference between the expert feature expectations and the feature expectations of all previous policies is found, under a certain weight vector. In the maximization, the weight vector which induces the maximum over this minimum is found. Since we would like to find a policy which generates feature expectations close to that of the expert's, the max-margin step can be seen as finding a weight vector which maximizes the minimum distance between the expert feature expectations and the feature expectations of all previous policies. If this distance is lower than some threshold, then we have induced at least one policy which is closer than some value ϵ to the expert feature expectations.

The projection method replaces this step by setting the weight vector to be the difference between the expert feature expectations and the orthogonal projection of the expert feature expectations onto the line through the previous projection and the feature expectations generated from the last returned policy. This removes the need for the algorithm to consider in each step all the previous policies which have been generated as is done in the max-margin version of the algorithm.

Both methods have the same convergence guarantees although the projection method was shown to be slightly faster [Abbeel and Ng 2004]. As the projection method removes the need for a quadratic program solver, it is simpler to implement. The projection method was used in this research for both these reasons. The run time of the algorithm depends on the number of features used; it grows linearithmically with the number of features [Abbeel and Ng 2004].

Apprenticeship learning is used when the reward function is unknown or difficult to manually specify. Once a reward function is calculated, reinforcement learning commences with that given reward function. The next section discusses work where reinforcement learning was used in a musical setting, both in the case where reward functions were manually specified and where they were extracted using apprenticeship learning via inverse reinforcement learning.

2.5 Reinforcement Learning and Music

Reinforcement learning depends on having a clearly defined reward function which the agent then uses to guide its learning. In a musical setting, it is not at all straightforward what the reward signal should be. In the paper *Reinforcement Learning for Live Musical Agents* Collins [2008] presents three ideas for musically meaningful reward signals.

The first idea is to match internal goals by the imposition of a rule set. This has been explored by Franklin and Manfredi [2002] who based the reinforcement signal on eight hand-written rules for jazz improvisation. They used actor-critic reinforcement learning using a non-linear recurrent neural network for note generation. The second idea Collins presents is to base the reward signal on the appreciation of fellow participants (other musicians with whom the learning agent is performing) and audience members. He suggests the use of tracking facial expressions and using physiological indicators such as monitoring galvanic skin response to gauge a listeners engagement, although these ambitious approaches have not yet been explored. A third reward signal he proposes is *memetic success*, that is, the taking up of the agent's musical ideas by others. This approach was explored by Collins using his music framework *Improvagent*, where the learning agent improvised with fellow musicians basing its reward on how much it influenced the position in the state space, given its choice of action; the effect of the action it took is measured by whether or not it influenced the current status quo of the musical piece (as played by the fellow musicians).

In another approach, the OMax system [Assayag *et al.* 2006] used reinforcement learning to weight links in a Factor Oracle (FO). A FO is a finite state automaton constructed incrementally in linear time and space. A musical sequence is used to build up the FO. Each symbol in the sequence corresponds to a state in the FO and reinforcement signals are used to weight the links between these states. This can be used for live musical interaction.

Another approach which has been used is to use apprenticeship learning via inverse reinforcement learning. As was described in the previous section, the apprenticeship learning algorithm can be used to extract a reward function given a set of trajectories. This algorithm was used by Acevedo *et al.* [2007] in creating a system capable of creating new pieces within a given style of music. The expert trajectories given to the learning algorithm were Bach's Chorales; the system then managed to create original melodies whose overall shape was characteristic of Bach's work. The research presented in this document differs from that presented in Acevedo *et al.* [2007] in three ways.

Firstly, the music given to the system as expert trajectories in this research was generated by the author, as opposed to being given Bach's Chorales as was done in the work presented in Acevedo *et al.* [2007].

Secondly, the state signal differed. In this research two different types of state signals were used. The first type of state signal was used for RA-0 and the second type was used for RA-1 and MA. A full discussion of these state signals is deferred until Sections 4.2.3, 5.2.1 and 6.2.2, but briefly, the first type of state signal gave only the current position within the piece and the second type gave only the last eight actions taken. In the Acevedo *et al.* [2007] case, the state signal was a tuple consisting of the position within the musical piece, the current pitch of the melody, the difference between the current pitch and the pitch of the previous state, the current chord type, the difference between the current chord type and the root of the previous chord and finally the status of the melody: whether it is resting, continuing to sound a previous note or starting a new note.

The third difference is in the action space. The actions available to the learning agents in this document were either which drum component to play next, or which note to play next. Again, a full discussion is deferred (see Sections 4.2.1 and 6.2.1). The actions in the research by Acevedo *et al.* [2007] denoted whether there was a change in the current portion that the musical piece was in, whether there was a change in the pitch of the melody, whether there was a change in the root of the chord being played, the resulting chord type from taking the action and finally, the status of the resulting musical state: whether this action is going to rest, hold or state a new note.

2.6 Conclusion

This chapter presented work which is relevant to this research. Section 2.2.1 presented some basic music theory, followed by Section 2.2.2 which described the interconnected history of music and mathematics. States and actions were introduced in Section 2.3.2 which led to Section 2.3.3 which described the Markov property as well as Markov decision processes (MDPs). Reinforcement learning and apprenticeship learning via inverse reinforcement learning were discussed in Sections 2.3 and 2.4 respectively. Section 2.5 then described work done by several authors in applying reinforcement learning to the task of musical composition.

The next chapter describes the methodology used in creating the learning agents used in this research.

Chapter 3

Research Methodology

3.1 Introduction

The previous chapter discussed literature relevant to this research. This chapter discusses the research aim, the stages of the research as well as the expert trajectories used by the learning agents. As discussed in Section 2.5, reinforcement learning has been used to create software agents capable of generating new music. The key issue surrounding this is the formulation of a musically meaningful reward function. This research used apprenticeship learning via inverse reinforcement learning to find such a reward function. The aim of this research was to provide a learning agent with expert trajectories from which it could generate its own musical pieces, which are musically similar to those expert musical pieces. Two drum-beat generating agents and one melody generating agent were created. The musical pieces given to these learning agents as expert trajectories were created by the author.

3.2 Research Aim

The aim of this research was to use reinforcement learning to generate new musical pieces given a set of expert musical pieces. As reinforcement learning requires a reward function for learning to commence and it is unclear how to manually specify one in the musical setting, apprenticeship learning via inverse reinforcement learning was used to uncover a reward function from expert behaviour. The expert behaviour used in this research corresponds to drum-beats and melodies created by the author.

The aim was to create musical agents capable of creating drum-beats as well as melodies. In both cases the agent was tasked with populating an 8×8 grid with actions. This 8×8 grid corresponded to the eight measures and eight beats (divisions) of the musical piece. This twofold aim of creating both a drum-beat generating agent as well as a melody generating agent was realised by the implementation of two rhythm agents (discussed in Chapters 4 and 5) as well as one melody agent (discussed in Chapter 6). The actions which the rhythm agents had at their disposal in populating the 8×8 grid corresponded to striking one or more components of a drum kit. The melody agent's actions corresponded to playing one of the notes available to it. The next section describes the stages of this research.

3.3 Research Stages

This research was broken up into two stages, corresponding to the two types of musical agents which were implemented. The first type of musical agent generated drum-beats, while the second generated melodies. In the first stage, two drum-beat generating agents were created — RA-0 and RA-1. In the second stage, one melody generating agent was created — MA.

Once each agent was created, several experiments were performed. The first of these experiments aimed to recreate a given expert trajectory. Further experiments demonstrate that the learning agents are capable of generating new pieces of music, although these pieces of music, as expected, are similar to the expert examples given to the learning agents. Details of the results of the experiments are given in the chapters corresponding to those learning agents. The following section discusses the expert trajectories used by the learning agents.

3.4 Expert Trajectories

Apprenticeship learning depends on having expert trajectories. These trajectories are used in order to generate a reward function which is then used by the learning agent to create a new policy derived from that reward function. From this new policy, new trajectories are created and the process is repeated - a new reward function is found and subsequently a new policy. The expert trajectories for each learning agent were created by the author. These were created in the form of expert drum-beats, for RA-0 and RA-1 and expert melodies, for MA. These expert trajectories were created as midi files and subsequently parsed into a form which the learning agents understood. 100 expert trajectories were created for the rhythm agents and 40 for the melody agent. Further implementation details regarding the expert trajectories are deferred until Sections 4.2.2 and 6.2.3 which discuss the rhythm agent's expert trajectories and the melody agent's expert trajectories, respectively.

3.5 Conclusion

This chapter discussed the research aim, research stages as well as the expert trajectories which were used by the learning agents. Further implementation details as well as the results of the experiments performed using each of the three learning agents is presented in the following three chapters.

Chapter 4

The First Drum-Beat Generating Agent - RA-0

4.1 Introduction

Rhythm Agent 0 — RA-0 — was the first of two drum-beat generating learning agents. In this first of two implementations of apprenticeship learning via inverse reinforcement learning for drum-beat generation, the learning agent received the previous four actions it had taken as its state history. In this context an action refers to striking one or two components of a drum kit. This chapter is broken up into an implementation and a results section. The implementation is presented in Section 4.2. Following this, Section 4.3 presents the results of the experiments run using RA-0.

4.2 Implementation

4.2.1 Action Space

The drums are played by striking components of a drum kit. A series of these strikes constitutes a drum-beat. In practice, one or more of these components may be struck simultaneously. In this research the strikes, or *actions*, available to the agent are limited to those shown in Figure 4.1 This figure shows which action number corresponds to which drum component. Actions 10 - 14 are composite actions; 2 drum components are played simultaneously. As a simple example, taking actions 1, 12, 0 in that order implies that first a bass drum is struck, followed by the simultaneous striking of the bass drum and crash cymbal and finally, in the last time step, no component is struck. This is known as a *rest*.

4.2.2 Rhythm Agents' Expert Trajectories

Both RA-0 and RA-1 used the same set of expert trajectories. The drum-beats which served as the trajectories were manually created by the author using a software drum machine (Hydrogen [hyd]). Each author-created drum-beat was saved as a midi file. A midi file contains information about music (in particular, information about events in a piece of music, such the timing of onsets of notes or drum hits). The trajectories were 8 measures long. Each measure contained 8 beats, yielding a total of 64 possible positions in which an action may be taken — where taking an action refers to playing one or more components of the drum kit. 100 expert trajectories were created. Since each drum-beat was comprised of 8 measures, and each measure was divided into 8 beats of equal length, each drum-beat was represented as an 8×8 matrix with the rows corresponding to the measures and the columns to the beats. When the drum-beats are played, they are played by traversing the matrix left to right, top to bottom. An

Drums(s)	**Action Number**
no hit	0
bass drum	1
snare	2
low tom	3
mid tom	4
high tom	5
closed hi-hat	6
open hi-hat	7
crash	8
ride	9
bass drum and snare	10
snare and crash	11
bass drum and crash	12
bass drum and closed hi-hat	13
bass drum and open hi-hat	14

Figure 4.1: Drum components and their action numbers

example of a drum-beat represented using this convention is shown in Figure 4.2. Figure 4.3 shows this same drum-beat represented in sheet music.

10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11

Figure 4.2: A drum-beat represented as a matrix.

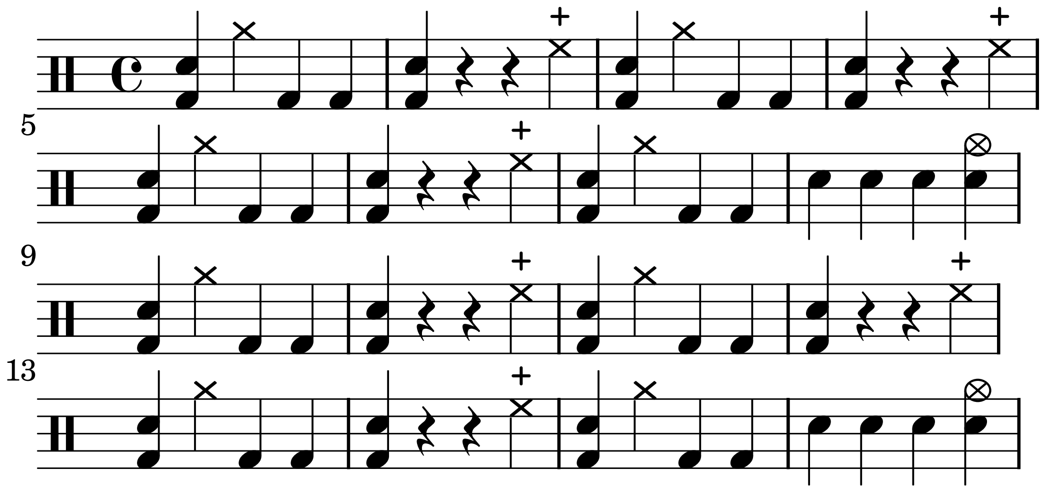

Figure 4.3: Sheet music corresponding to 4.2

As previously stated, 100 expert trajectories were created. These trajectories all exhibited similar characteristics; of the 8 measures which comprised these trajectories, measures 1 – 3 and 5 – 8 were identical. Further, the first four beats in each measure were the same. The second halves of measures 4 and 8 were identical to each other. These are called *fills*. The reason for this structure is because a drum-beat forms the backbone of a song and as such it is required that it is somewhat consistent. The fills allow for some variation in the drum-beat, but once they are completed the regular pattern is resumed. The 100 expert trajectories were split into 5 equal sets of twenty. Within each set the fills used were the same. Thus, there were five fill patterns used for all 100 expert trajectories. The fill patterns were:

- 2 2 2 11
- 1 0 1 0
- 0 0 12 12
- 1 1 1 11
- 14 5 12 0

The expert trajectories were grouped in this manner in the hopes that these similar characteristics would be learned by the learning agent; that the agent would begin with one of the expert trajectories, encounter a fill, and then resume playing a different expert trajectory from the same set. These expert trajectories can be heard at `http://aiml.cs.wits.ac.za/orry/c/expert_beats/`.

4.2.3 State Space

The following 4-tuple was given to the agent as a state signal: (x_0, x_1, x_2, x_3). Each x_i refers to a previous action taken, where x_0 was the most recent action taken and x_3 the least recent. Since the number of actions available to this learning agent was 15 and the history length was 4, this led to a state space of size 15^4. The next section discusses the feature vector, ϕ, which was used for RA-0.

The state space of RA-0 was the tuple consisting of the four previous actions taken. The start state in all cases was $(-1, -1, -1, -1)$. Once action a_1 is taken, it is put into the first position of the state vector with every other variable shifting to the right by one position. Consider the sequence of states starting from the start state in which actions a_1, a_2, a_3 a_4 are taken in that order:

- Start state: $(-1, -1, -1, -1)$. Action a_1 is now taken.
- State 2: $(a_1, -1, -1, -1)$. Action a_2 is now taken.
- State 3: $(a_2, a_1, -1, -1)$. Action a_3 is now taken.
- State 4: $(a_3, a_2, a_1, -1)$. Action a_4 is now taken.
- State 5: (a_4, a_3, a_2, a_1).

4.2.4 Feature Vector

As stated in the previous section, the state signal was composed of four state variables, $x_0 - x_3$. These four state variables were used to make up the feature vector, ϕ. Note that it is not in general the case that the state signal is the same as the feature vector, as it is in this research. The start state of the agent had action value -1 in all four state variables. The initial state signal was $(-1, -1, -1, -1)$. Since (x_0, x_1, x_2, x_3) can have 15^4 possible combinations, as $0 \leq x_i \leq 14$ (where $0 \leq i \leq 3$), ϕ is a vector of length 15^4 , where each element corresponds to a unique combination of the state signal.

Since the state space is discrete, and the feature vector is the same as the state signal, the reward function which is a linear combination of the features must lie in the span of the features. Put another way, each reward state is exactly represented by a unique feature vector and the feature vectors cover the entire state space. Thus, a reward function which maps states to the real numbers can be directly modelled by the features.

4.2.5 Expert Feature Expectations

As was explained in Section 2.4.3, the expert feature expectations vector, μ_E, is the sum of the discounted feature vectors which the expert observes when following his policy. Feature expectations average the expected features, given a policy. Since the expert's policy is unknown, μ_E must be estimated by $\hat{\mu}_E$. $\hat{\mu}_E$ is the expert's averaged discounted accumulated feature value vector, given a set of trajectories. This is shown in Equation 4.1. The features of each of the 64 states the expert encountered, $\phi(s)$, are added together, using a discount rate of γ - this is shown in Equation 4.1 as the inner sum. The outer sum runs through the m expert trajectories given to the learning agent.

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{63} \gamma^t \phi(s_t^{(i)}) \tag{4.1}$$

Equation 4.1: the expert feature expectations vector

Apprenticeship learning via inverse reinforcement learning (Section 2.4) was used to uncover the reward function using these expert trajectories.

4.2.6 The Reinforcement Learning Step

Once the reward function has been returned, Q-learning was used in the reinforcement learning step of the apprenticeship learning algorithm. Q-learning finds a policy by learning a state-action value function, Q, from which a policy can be derived. Greedy action selection and softmax were used to derive the policies which led to the expert trajectories. Q was stored in 15 tables, corresponding to the 15 actions available to the learning agent.

4.2.7 Calculating the New Feature Expectations

Once a new policy π was ascertained from the previous step, the new feature expectations $\mu(\pi)$ were calculated using Equation 4.2. Since the policy is deterministic, the drum-beat which the policy constructs is known exactly, and thus the feature expectations can be calculated without taking an expectation. Once the new feature expectations have been found, these are used in the projection method to create a new weight vector which in turn leads to a new reward function, and the process repeats until a stop condition has been met, or the process is manually stopped. Using this implementation, the results presented in the following section were ascertained.

$$\mu(\pi) = \sum_{t=0}^{63} \gamma^t \phi(s_t) \tag{4.2}$$

4.3 Results - Rhythm Agent 0

4.3.1 Introduction

This section provides the results of the experiments run using RA-0. Four experiments were run using this learning agent. These are presented in Sections 4.3.2 – 4.3.5. In Experiment 1.a (Section 4.3.2) a simple reward function was created manually by the author. An expert trajectory was then created using this reward function. The learning agent then learned the reward function from this trajectory and exhibited behaviour in line with the initial reward function. In Experiment 1.b (Section 4.3.2), one of the expert trajectories was given to the agent in the hope that it would find a reward function which it could use to exactly replicate the given expert trajectory. The learning agent was able to create a similar, but not necessarily identical trajectory to the given expert trajectory. Experiment 2 (Section 4.3.4) was then run in which 20 expert trajectories from the same set were given to the learning agent. These expert trajectories had the same fills in the second halves of measures four and eight, as explained in Section 4.2.2. Both greedy and softmax action selection were investigated. Finally, in Experiment 3 (Section 4.3.5), all 100 of the expert trajectories were given to the learning agent. Once again, the use of greedy and softmax action selection were investigated.

In all the experiments the following parameters were set: $(\gamma, \gamma_2, \epsilon, \epsilon_2, \alpha, \lambda)$. γ was the discount rate used in Q-learning. γ_2 was the discount rate used in calculating feature expectations. ϵ was the exploration rate used in Q-learning. α was the learning rate used in Q-learning. ϵ_2 was the stop condition used for apprenticeship learning. λ was the eligibility trace parameter. The start state of the learning agent was $(-1, -1, -1, -1)$. It was found that giving the agent a start state which included any of the action numbers affected learning adversely. Thus a start state which did not include any of the actions available to the agent was used. In addition to the parameters mentioned above, in experiments where softmax action selection was used to select the final policy, the temperature, τ, was set. Discounting was used in these tasks, despite their being episodic, since discounting gives some indication of placement within a piece; the more a reward or value is discounted, the further in the piece it appeared.

4.3.2 Experiment 1.a — Simple Reward Function

This section presents the results of the experiment in which a simple reward function was used in creating an expert trajectory which RA-0 used for learning. In order to create the expert trajectory, Q-learning was run using a manually created reward function. The reward function used was the following: in state $(1, 2, 3, 4)$ the agent was given a reward of $+100$. In every other state it was given a reward of -30. Once Q-learning had completed, the drum-beat presented in Figure 4.4 was returned. This was expected since in order to reach state $(1, 2, 3, 4)$, the agent would need to take the actions which made up the state variables in reverse order, as the most recent action taken was the first state variable.

The parameters for this experiment were set as follows:

- $\gamma = 0.9$
- $\gamma_2 = 1$
- $\epsilon = 0.1$
- $\epsilon_2 = 0.5$
- $\alpha = 0.1$
- $\lambda = 0.5$

Q-learning was configured to stop when it produced the same policy in 3000 consecutive episodes. The start state of the learning agent was $(-1, -1, -1, -1)$. This learning agent used greedy action selection both during learning and when picking a final policy once learning had completed.

4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1

Figure 4.4: The drum-beat generated using the manually created reward function. It can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp1a/4.4/`

The results of this experiment were very promising; within a matter of seconds the learning agent converged. It returned a policy which exactly replicated the expert trajectory; the expert trajectory itself was generated using the manually created reward function. The drum-beat which resulted from the manually created reward function is not particularly musically meaningful. The purpose of this experiment was simply to check whether it was possible for the learning agent to return a policy which was based on a reward function which it had no access to. The indirect access it had was in the form of the expert trajectory. Since the learning agent produced a trajectory which was identical to the one expert trajectory given to it, it is clear that it has in this simple case learned an appropriate reward function. The following section discusses the results of the experiment run using a single author-created expert trajectory.

4.3.3 Experiment 1.b — One Expert Trajectory

The expert trajectory used in this experiment is shown in Figure 4.5. In this expert trajectory, all measures besides measures 4 and 8 were identical. Measures 4 and 8 contained fills (these were explained in Section 4.2.2). This structure is typical of a drum-beat; since drums form the "backbone" of a song, they need to be somewhat consistent. It was hoped that in this experiment RA-0 would learn to replicate the expert trajectory.

The following parameters were used in this experiment:

- $\gamma = 0.9$
- $\gamma_2 = 0.9$
- $\epsilon = 0.2$
- $\epsilon_2 = 0.5$
- $\alpha = 0.1$
- $\lambda = 0$
- $\tau = 0.25$

Q-learning was configured to stop once it had produced the same policy 6000 times. It was found that turning off eligibility traces, by setting $\lambda = 0$ as well as setting the discount of the feature expectations to $\gamma_2 = 0.9$ produced better results than the parameter values used in the previous experiment. Using the previous parameter values led to the learning agent being in states which had not been observed in the expert trajectory.

10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11

Figure 4.5: The sole expert trajectory given to the learning agent in this experiment. It can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp1b/4.5/`

Both greedy action selection and softmax were used to create policies and ultimately trajectories. When greedy action selection was used, the learning agent chose the measures without the fills; it never chose to play a fill, as the states which would result from playing the fill are all of lower value than similar states in which no fill is played. This is due to the fact that the fills are not played as often as other patterns in the drum-beat. The result of using greedy action selection is shown in Figure 4.6.

10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6

Figure 4.6: The expert trajectory generated using the greedy policy. It can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp1b/4.6/`

In order for the learning agent to reach states which had some positive value, although not as high as the optimal states, softmax action selection was used to generate softmax trajectories. Of particular interest in this experiment is the state $(1, 1, 9, 10)$ of the trajectory shown in Figure 4.5. This state was observed 8 times in the trajectory. After 6 of these 8 states, action 10 was taken. After 2 of these 8 states, action 2 was taken. By using softmax action selection after learning had completed — and after some trial and error in setting the temperature, τ — the results shown in Figures 4.7a – 4.7d were obtained. As can be seen from the results, action 10 follows state $(1, 1, 9, 10)$ more often than action 2. Since softmax action selection was used in the playouts, the playouts were now not deterministic. This accounts for the variation in the results seen in Figures 4.7a – 4.7d. The learning agent did not necessarily play actions 2, 2, 2, 11 at the same location as was done in the expert trajectory. In order for the agent to learn to play the fills in the correct locations, the learning agent needed timing information from the state signal. This led to the changing of state variables which was done in RA-1, where the learning agent was given its current location in the trajectory as part of its state signal, in addition to a two action history. This implementation of RA-1 is discussed more comprehensively in Section 5.2.

10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11

(a)

10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11

(b)

10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6

(c)

10	9	1	1	2	2	2	11
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11

(d)

Figure 4.7: Softmax trajectories generated by RA-0 using one expert trajectory. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp1b/4.7/`

4.3.4 Experiment 2

Twenty expert trajectories were used in this experiment. Although they all differed, they had one similar characteristic: the second halves of measures 4 and 8 were the actions 2, 2, 2 and 11. In other words, the fills were all identical and positioned in the same location within the trajectory. Aside from the fills, each beat within a measure was identical to that same beat in every other measure. Several of the expert trajectories used in this experiment are shown in Figure 4.8.

All of the parameters besides ϵ_2 were set at values used for Experiment 1.b. ϵ_2 was set to 0.01. Since ϵ_2 was used in the stop condition of the apprenticeship learning algorithm, reducing it to 0.01 meant that the algorithm would take longer to converge and thus produce more output. When the program was run, after some time, it became clear that the t value would not drop below $\epsilon_2 = 0.01$ and thus that the algorithm would not terminate according to this condition, despite the convergence guarantees of the algorithm. Convergence is guaranteed only with a certain sample size of expert trajectories; here, the number of expert trajectories given is too low for the t value to drop below the threshold of 0.01. The program was then stopped manually with a t value equal to 0.01220671118499733. Since the apprenticeship learning algorithm does not make any guarantees regarding which returned policy is the "best" (that is, the last returned policy is not necessarily better than any other returned policy) [Abbeel and Ng 2004], the entire range of returned policies was investigated. At the end of each iteration of apprenticeship learning both the greedy policy and the softmax policy were returned. From these returned policies, trajectories were then created. A set of greedy trajectories and a set of softmax trajectories were thus created. There was variability both within each set, but more prominently, between the two sets. Of the many returned trajectories, only a few of them are shown in Figures 4.9 and 4.10. However, these trajectories serve as a good representation for the types of trajectories from each set.

1	4	13	4	1	0	1	3
1	4	13	4	1	0	1	3
1	4	13	4	1	0	1	3
1	4	13	4	2	2	2	11
1	4	13	4	1	0	1	3
1	4	13	4	1	0	1	3
1	4	13	4	1	0	1	3
1	4	13	4	2	2	2	11

(a)

1	9	14	4	1	0	1	3
1	9	14	4	1	0	1	3
1	9	14	4	1	0	1	3
1	9	14	4	2	2	2	11
1	9	14	4	1	0	1	3
1	9	14	4	1	0	1	3
1	9	14	4	1	0	1	3
1	9	14	4	2	2	2	11

(b)

1	0	14	2	1	0	1	3
1	0	14	2	1	0	1	3
1	0	14	2	1	0	1	3
1	0	14	2	2	2	2	11
1	0	14	2	1	0	1	3
1	0	14	2	1	0	1	3
1	0	14	2	1	0	1	3
1	0	14	2	2	2	2	11

(c)

1	0	12	2	1	0	1	3
1	0	12	2	1	0	1	3
1	0	12	2	1	0	1	3
1	0	12	2	2	2	2	11
1	0	12	2	1	0	1	3
1	0	12	2	1	0	1	3
1	0	12	2	1	0	1	3
1	0	12	2	2	2	2	11

(d)

Figure 4.8: Several of the expert trajectories used for Experiment 2 of RA-0. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/expert_beats/1/`

It is clear that the softmax trajectories are much more erratic than the greedy trajectories. This is due to the greedy trajectories finding the best states, and sticking to them. This leads to far more consistent drum-beats, but, unfortunately, the fills are not played as they are the result of the agent choosing sub-optimal actions. In the softmax trajectories, the pattern 2 2 2 11 is present in all the trajectories shown in Figure 4.10. Although it was not present in all the softmax trajectories, it was present in the majority. Here we have behaviour closer to what was hoped for: a generated drum-beat which "switches" its trajectory after encountering a fill. This is especially clear in the first three measures of the trajectory shown in Figure 4.10d. At first the pattern 10 9 1 1 10 0 0 6 is played in the first measure. This pattern comes directly from an expert trajectory. In the following measure, the fill is played followed by a change of trajectory in the third measure. Examples of this can be seen throughout the generated softmax trajectories. Unfortunately, the timing information is not retained; the fills are played seemingly at random, unlike their structured placement in the expert trajectories. These resulting trajectories show the need for timing information in order to produce more structured drum-beats.

12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0
12	0	12	0	12	0	12	0

(a)

12	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0
1	0	12	2	1	2	12	0

(b)

1	0	14	2	1	5	1	3
14	5	14	2	1	5	1	3
14	5	14	2	1	5	1	3
14	5	14	2	1	5	1	3
14	5	14	2	1	5	1	3
14	5	14	2	1	5	1	3
14	5	14	2	1	5	1	3
14	5	14	2	1	5	1	3

(c)

1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0
1	0	12	4	12	0	5	0

(d)

Figure 4.9: RA-0, Experiment 2 — trajectories generated using the greedy policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp2/4.9/`

1	4	13	4	2	2	2	11
12	0	12	11	12	0	12	2
0	12	12	2	12	11	12	0
0	12	2	0	12	12	0	0
12	12	2	12	11	2	2	2
2	2	2	2	2	11	10	9
1	1	10	0	0	6	10	9
1	1	2	2	2	2	2	11

(a)

12	9	14	11	1	0	12	2
1	0	9	12	1	8	2	0
1	2	9	12	1	8	0	2
1	0	4	12	14	4	1	2
2	2	11	10	9	1	1	10
0	0	6	10	9	1	1	10
0	0	6	10	9	1	1	10
0	0	6	10	9	1	1	2

(b)

1	0	1	2	2	2	11	12
2	1	2	2	2	2	2	11
1	9	14	4	1	0	1	3
1	8	1	2	2	2	11	10
9	1	1	2	2	2	11	12
9	14	11	2	2	2	11	14
5	14	2	1	0	4	12	14
4	1	2	1	0	0	12	1

(c)

10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11
12	2	12	11	12	0	0	12
2	1	2	2	2	11	12	2
1	2	2	2	11	12	0	12
0	1	0	12	2	1	2	12
0	1	6	1	3	12	0	12
2	1	2	2	2	2	2	2

(d)

Figure 4.10: RA-0, Experiment 2 — trajectories generated using the softmax policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp2/4.10/`

4.3.5 Experiment 3

All 100 expert trajectories were used in this experiment. As explained in Section, 4.2.2, the expert trajectories were divided into 5 sets. Each set had the same fill in the second half of measures 4 and 8. The parameters were the same as those used in the previous experiment. Unlike the previous experiment though, the t value dropped below the threshold of $\epsilon_2 = 0.01$ at which point the program terminated. Note that the number of expert trajectories used in this experiment was 100. When using 20 expert trajectories, as in the previous experiment, the t value did not drop below the threshold of $\epsilon_2 = 0.01$. Figure 4.11 shows several of the trajectories created using the greedy policy and Figure 4.12 shows several of the trajectories created using softmax. These results demonstrate the need for timing information to be incorporated into the state signal. While the learning agent does find "good states" it moves from them without the type of structure needed of a drum-beat — especially in the softmax case.

The trajectories which resulted from using greedy action selection are similar to those generated in the previous experiment. As was the case in the previous section, only a subset of the generated greedy trajectories are shown but again, they are demonstrative of the types of trajectories generated by the agent. As can be seen from Figures 4.11a and 4.11d, some of the generated trajectories have the type of periodicity desirable of a drum-beat. There is a consistent structure which provides the backbone of the song. However, the learning agent was not able to play a fill and then return to the drum-beat's structure. The trajectories shown in Figures 4.11b and 4.11c exhibit repeated patterns as well, but not patterns of eight beat length, as in the case of Figures 4.11a and 4.11d. For example, the trajectory presented in Figure 4.11b exhibits the five beat pattern 1 0 2 12 0 repeatedly.

The trajectories generated by the softmax policy are even more erratic. There does seem to be some repetition between the measures, as well as the playing of fills, but the beats are not consistent enough to form the basis of a song. The following section provides a discussion on the results of these experiments.

4.3.6 Discussion

The results show that this approach to apprenticeship learning via inverse reinforcement learning can work, but only to a limited extent. Experiments 1.a and 1.b show that this approach works in the case of a simple, manually generated reward function and one expert trajectory, respectively. However, once multiple trajectories are given to the learning agent, either the structure of the drum-beats is lost, as is the case with softmax trajectories, or the trajectories do not incorporate fills, as is typically the case with greedy trajectories. The modifications described in the following chapter, which led to the creation of RA-1, attempt to overcome these limitations.

4.4 Conclusion

The implementation of as well as results using RA-0 were presented in this chapter. Following a description of the implementation details in Section 4.2, the results of the experiments were presented in 4.3. The results show that this implementation of apprenticeship learning via inverse reinforcement learning to generating drum-beats can work in simple cases. However, this approach needs to be improved upon when presenting the learning agent with multiple expert trajectories. This improvement was implemented in the form of RA-1, which is the subject of the following chapter.

12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12
12	0	0	12	12	0	0	12

(a)

5	1	1	1	0	2	12	0
1	0	2	12	0	1	0	2
12	0	1	0	2	12	0	1
0	2	12	0	1	0	2	12
0	1	0	2	12	0	1	0
2	12	0	1	0	2	12	0
1	0	2	12	0	1	0	2
12	0	1	0	2	12	0	1

(b)

1	0	1	3	1	5	0	1
0	12	4	12	0	1	2	1
0	0	2	2	9	0	0	9
0	0	9	0	0	9	0	0
9	0	0	9	0	0	9	0
0	9	0	0	9	0	0	9
0	0	9	0	0	9	0	0
9	0	0	9	0	0	9	0

(c)

2	0	0	6	1	0	1	1
2	0	1	12	6	9	1	1
2	0	1	12	6	9	1	1
2	0	1	12	6	9	1	1
2	0	1	12	6	9	1	1
2	0	1	12	6	9	1	1
2	0	1	12	6	9	1	1
2	0	1	12	6	9	1	1

(d)

Figure 4.11: RA-0, Experiment 3 — trajectories generated using the greedy policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp3/4.11/`

0	0	0	12	1	1	5	9
0	8	2	12	0	1	2	0
1	0	2	12	0	4	5	1
0	0	2	0	2	1	1	2
1	14	5	12	0	3	0	3
12	3	0	3	0	14	5	12
0	6	1	0	1	3	12	9
14	11	1	0	0	1	0	1

(a)

12	9	14	11	1	0	12	2
1	0	9	12	1	8	2	0
1	2	9	12	1	8	0	2
1	0	4	12	14	4	1	2
2	2	11	10	9	1	1	10
0	0	6	10	9	1	1	10
0	0	6	10	9	1	1	10
0	0	6	10	9	1	1	2

(b)

1	0	1	2	2	2	11	12
2	1	2	2	2	2	2	11
1	9	14	4	1	0	1	3
1	8	1	2	2	2	11	10
9	1	1	2	2	2	11	12
9	14	11	2	2	2	11	14
5	14	2	1	0	4	12	14
4	1	2	1	0	0	12	1

(c)

10	9	1	1	10	0	0	6
10	9	1	1	2	2	2	11
12	2	12	11	12	0	0	12
2	1	2	2	2	11	12	2
1	2	2	2	11	12	0	12
0	1	0	12	2	1	2	12
0	1	6	1	3	12	0	12
2	1	2	2	2	2	2	2

(d)

Figure 4.12: RA-0, Experiment 3 — trajectories generated using the softmax policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-0/exp3/4.12/`

Chapter 5

The Second Drum-Beat Generating Agent - RA-1

5.1 Introduction

Rhythm Agent 1 (RA-1) is the second drum playing agent which used the apprenticeship learning algorithm to generate drum-beats. It received a state signal composed of the last 2 actions it took as well as its current position in the piece — that is, its current measure and beat. Thus, the learning agent had both an absolute context from which to base its decisions as well as a relative context. The absolute context is conveyed in terms of the first two variables of the state signal and the relative context is conveyed via the last two variables of the state signal.

As with the previous chapter, this chapter is divided into two sections: Section 5.2 which describes the implementation of RA-1 and Section 5.3 which describes the results of the experiments performed using RA-1. Since the only changes between RA-1 and RA-0 have to do with the state space and feature vector, these make up the content of the two sections discussed in Section 5.2, the implementation section.

5.2 Implementation

5.2.1 State Space

The state signal received by RA-1 was the absolute time step within the song as well as the previous two actions taken. The state signal still comprised of 4 state variables, as it did in RA-0. However, in this case the first two variables refer to the current time step within the song. The following 4-tuple was given to the agent as state signal: (x_0, x_1, x_2, x_3). x_0 and x_1 refer to the measure and beat, respectively. x_2 refers to the most recent action taken and x_3 refers to the action taken before that. The state state in all cases was $(1, 1, -1, -1)$, conveying the information that the current measure is measure 1 and the current beat is beat 1. The previous actions taken were undefined; that is, no actions have previously been taken. Consider the sequence of states starting from the start state in which actions a_1, a_2, a_3 a_4 are taken in that order:

- Start state: $(1, 1, -1, -1)$. Action a_1 is now taken.
- State 2: $(1, 2, a_1, -1)$. Action a_2 is now taken.
- State 3: $(1, 3, a_2, a_1)$. Action a_3 is now taken.
- State 4: $(1, 4, a_3, a_2)$. Action a_4 is now taken.

- State 5: $(1, 5, a_4, a_3)$.

As a further example, consider the state $(3, 4, 0, 1)$; it conveys the information that the current position is measure 3, beat 4, and the previous action taken was 0 with the action before that being 1.

5.2.2 Feature Vector

As stated in the previous section, the state signal was composed of 4 state variables, $x_0 - x_3$. These 4 state variables were used to make up the feature vector, ϕ. Once again, it is important to note that it is not in general the case that the state signal is the same as the feature vector, as it is in this research. The initial state signal was $(1, 1, -1, -1)$. In the case of this state signal, (x_0, x_1, x_2, x_3), x_0 and x_1 can have 8 possible values each (8 measures and 8 beats within those measures), and x_2 and x_3 can have 15 possible values corresponding to the 15 different action choices. Thus, (x_0, x_1, x_2, x_3) can have $8^2 \times 15^2$ possible combinations. ϕ is therefore a vector of length $8^2 \times 15^2$, where each element corresponds to a unique combination of the state signal.

As was the case with RA-0, the state space is discrete and the feature vector is the same as the state signal. The reward function which is a linear combination of the features must lie in the span of the features. Put another way, each reward state is exactly represented by a unique feature vector and the feature vectors cover the entire state space. Thus, a reward function which maps states to the real numbers can be directly modelled by the features.

Aside from the changes in state space, RA-1 was implemented in the same manner as RA-0. The following section presents the results of the experiments run using RA-1.

5.3 Results - Rhythm Agent 1

5.3.1 Introduction

This section provides the results of the experiments using RA-1. Three experiments were run using this agent. These are presented in Sections 5.3.2 – 5.3.3. A discussion in Section 5.3.5 then follows. Once again, the parameters $(\gamma, \gamma_2, \epsilon, \epsilon_2, \alpha, \lambda)$ were set for each experiment. Additionally, τ was set when using softmax to generate trajectories.

5.3.2 Experiment 1 - One Expert Trajectory

This section presents the results in which one expert trajectory was given to the learning agent. As in the case of Experiment 1.b, the trajectory shown in Figure 4.5 was used as the sole expert trajectory. The following parameters were used in this experiment:

- $\gamma = 0.9$
- $\gamma_2 = 0.9$
- $\epsilon = 0.1$
- $\epsilon_2 = 1.1$
- $\alpha = 0.1$
- $\lambda = 0.0$

The learning agent converged within a matter of seconds. The greedy policy was then used to generate the returned trajectory which was identical to the expert trajectory used as input. This experiment demonstrates the ability of the new state space to exactly recreate a given expert trajectory. Softmax was not used to generate a trajectory as the purpose of this experiment was to test whether this agent could exactly replicate a given expert trajectory.

5.3.3 Experiment 2 - 20 Expert Trajectories

This section presents the results of the experiment run with 20 expert trajectories given to the learning agent. The set of 20 expert trajectories all had the same fill in their fourth and eighth measures. The fill used was 2 2 2 11. Figure 5.1 shows several of the greedy trajectories and Figure 5.2 shows several of the softmax trajectories.

It can be seen in the greedy trajectories that the learning agent has learned to play the fills in the correct locations — the fills are played in the second halves of measures 4 and 8. The learning agent has clearly learned some other features from the expert trajectories; for example the agents proclivity for taking actions 1, 10, 12 and 14 in the first beats of measures can be attributed to this feature of the expert drum-beats. This was purposefully done by the author, since these actions incorporate playing the kick drum (see figure 4.1), and this is a common way to start measures.

The softmax generated trajectories are much more erratic than the greedy trajectories. These trajectories exhibit neither the periodicity desired of a drum-beat nor the ability to play the fills in the correct locations. It appears that using greedy action selection to generate the trajectories is better in this scenario. The greedy trajectories all exhibit the fills in the locations where they are expected to be; this is in contrast with the softmax trajectories. Further, there appears to be more of the type of periodicity desirable in a drum-beat in the greedy trajectories. This can be seen most clearly in figure 5.1b where the first beat of every measure is action 1, corresponding to a bass-drum. Further examples of this type of periodicity, where the columns in the matrices are equal to one another are more prevalent in the greedy case than in the softmax case.

10	9	14	2	12	0	0	6
10	9	1	2	12	0	4	12
1	4	13	4	1	0	1	0
12	2	12	2	2	2	2	11
10	9	14	2	1	0	1	0
1	8	0	2	12	0	4	12
12	0	12	0	1	2	0	6
10	9	1	2	2	2	2	11

(a)

1	0	1	2	1	0	1	3
1	0	12	0	1	0	1	0
1	0	1	1	10	0	1	3
1	0	1	1	2	2	2	11
1	0	1	2	1	0	1	3
1	0	12	0	1	0	1	0
1	0	1	2	1	0	1	3
1	0	1	1	2	2	2	11

(b)

12	0	12	0	1	2	0	12
1	4	1	2	1	2	0	6
10	9	1	2	1	0	4	12
1	4	1	2	2	2	2	11
12	0	12	0	1	2	0	12
1	4	1	2	1	2	0	12
1	4	1	1	10	0	4	12
1	4	1	2	2	2	2	11

(c)

14	4	1	1	10	0	4	12
12	0	1	1	10	0	1	3
1	8	0	2	12	0	5	0
1	9	14	2	2	2	2	11
14	4	1	1	10	0	4	12
12	0	1	1	10	0	1	3
1	8	0	2	12	0	5	0
1	8	2	0	2	2	2	11

(d)

Figure 5.1: RA-1, Experiment 2 — trajectories generated using the greedy policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-1/exp2/5.1/`

1	0	12	5	8	1	11	5
9	11	5	8	9	10	12	12
3	13	4	11	5	2	7	7
2	14	11	12	0	2	0	5
10	14	9	5	11	11	1	11
5	11	14	9	3	14	9	4
2	5	14	7	10	14	1	6
13	1	10	7	4	12	13	1

(a)

10	9	14	4	3	11	5	10
14	14	2	4	0	8	5	4
13	3	10	13	10	2	9	13
2	13	11	9	0	0	5	12
14	14	5	10	1	5	14	6
5	12	14	9	2	4	14	12
9	7	9	13	8	2	14	7
5	0	1	2	2	2	2	11

(b)

12	0	12	2	1	6	4	11
9	13	3	10	13	13	7	9
7	8	13	4	11	1	0	13
8	12	4	9	14	5	14	11
3	10	9	7	11	6	9	6
12	11	13	9	3	12	11	12
6	9	11	8	8	11	14	7
13	3	9	14	12	2	14	4

(c)

1	0	14	11	1	6	12	1
14	13	7	11	10	7	13	4
8	6	8	8	7	14	4	11
14	11	6	12	11	1	13	10
2	7	6	13	12	9	1	4
7	14	7	3	14	1	4	13
11	13	9	7	11	11	2	13
9	9	9	13	12	9	4	12

(d)

Figure 5.2: RA-1, Experiment 2 — trajectories generated using the softmax policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-1/exp2/5.2/`

5.3.4 Experiment 3 - 100 Expert Trajectories

This section presents the results of the experiment run with 100 expert trajectories given to the learning agent. This is the same set of 100 expert trajectories used for Experiment 3 of RA-0. As was explained in Section 4.2.2, this set of 100 trajectories was comprised of 20 subsets. Each trajectory in a particular subset contained the same fill, at the same location in the trajectory. At each iteration of the apprenticeship learning algorithm, a softmax and a greedy trajectory were generated.

Several of the softmax trajectories are shown in Figure 5.4. As was the case in Experiment 2, the softmax trajectories are extremely erratic; they do not appear to be any better than random. Several of the greedy trajectories are presented in Figure 5.3. The greedy trajectories do play the fills at the correct locations, and there does appear to be some structure to the playing — as with the greedy trajectories generated in the previous experiment, the learning agent often plays the same action in the same beat of different measures.

12	0	2	0	2	0	5	0
3	0	2	0	1	0	3	0
3	0	2	0	2	0	14	0
3	0	2	0	2	2	2	11
1	1	2	0	2	0	3	0
13	0	2	0	2	0	3	0
3	0	2	0	1	0	5	0
14	0	2	0	2	2	2	11

(a)

3	0	7	0	12	0	3	0
5	0	13	0	2	0	5	0
5	0	3	0	12	0	3	0
5	0	3	0	14	5	12	0
12	0	14	0	12	0	5	0
5	9	0	1	2	1	0	1
1	1	2	1	0	1	0	1
1	1	0	1	1	1	1	11

(b)

14	0	12	0	9	1	0	1
1	1	0	1	12	1	0	1
1	1	2	1	0	1	0	1
1	1	0	8	0	0	12	12
1	2	0	1	2	1	0	1
1	1	0	2	1	1	2	1
2	0	3	0	12	0	3	0
3	0	12	0	14	5	12	0

(c)

1	1	0	1	12	1	0	1
1	1	2	1	0	1	0	1
1	8	0	2	12	0	5	0
1	1	0	8	0	0	12	12
1	2	0	1	2	1	0	1
1	1	0	2	1	1	2	1
2	0	3	0	12	0	3	0
3	0	12	0	14	5	12	0

(d)

Figure 5.3: RA-1, Experiment 3 — trajectories generated using the greedy policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-1/exp3/5.3/`

7	9	3	0	1	0	4	12
3	7	2	6	2	6	12	4
14	2	0	14	3	6	2	7
11	11	6	5	10	8	7	13
7	0	14	11	3	1	4	6
5	1	3	2	5	0	12	13
6	7	3	7	14	13	2	6
9	1	0	10	3	9	11	11

(a)

4	12	14	9	5	11	4	12
6	6	1	14	1	5	4	2
12	0	7	11	10	3	2	8
14	1	6	11	1	10	3	7
7	4	6	1	2	4	14	13
9	5	0	4	3	1	3	12
3	6	6	9	2	11	6	12
4	14	13	0	12	12	11	7

(b)

6	2	3	10	3	3	10	6
2	4	3	1	14	4	2	13
6	4	13	3	5	10	5	8
1	3	8	11	2	2	1	12
6	5	7	1	3	2	8	13
1	6	9	13	8	12	10	1
11	2	13	9	9	5	4	11
4	4	12	3	0	12	2	9

(c)

13	3	5	13	5	4	5	3
14	7	7	5	3	14	7	11
6	6	4	9	1	12	11	6
7	5	9	14	8	5	8	12
14	10	11	6	2	13	9	3
8	12	12	11	9	2	12	6
4	12	13	11	2	7	1	13
2	7	3	6	8	4	11	8

(d)

Figure 5.4: RA-1, Experiment 3 — trajectories generated using the softmax policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/RA-1/exp3/5.4/`

5.3.5 Discussion

The results of the experiments performed using RA-1 show that it is an improvement over RA-0 in two ways. Firstly, Experiment 1 shows that unlike RA-0, RA-1 is able to exactly replicate an expert trajectory, when only one is given to is as input. Secondly, Experiments 2 and 3 show that, when using the greedy policy, the learning agent is able to consistently play the fills in the correct locations in the generated trajectories. This is an improvement over RA-0, as when the greedy trajectories were taken in that case, the learning agent was not able to play fills.

Although it is hard to judge "how good" the results are, given the nature of the domain, several characteristics of the generated trajectories demonstrate that RA-1 has learned, to a certain degree, to generate coherent drum-beats in unexpected ways. The greedy trajectories generated in running Experiments 2 and 3 demonstrate what is meant by coherent. Firstly, the learning agent plays fills in the correct locations. Secondly, there is an element of repetition in the drum beats — the agent often chooses the same action in the same beat of different measures. RA-1 has managed to learn to play somewhat consistently.

5.4 Conclusion

The implementation of as well as results using RA-1 were presented in this chapter. Following a description of the implementation details in Section 5.2, the results of the experiments were presented in Section 5.3. The following chapter presents the work done relating to the MA, the melody generating agent.

Chapter 6

Melody Agent - MA

6.1 Introduction

Melody Agent (MA) is the melody generating implementation of the apprenticeship learning algorithm. Like RA-0 and RA-1, this learning agent relies on a set of expert trajectories which were created by the author. Like RA-0, the state signal of MA is comprised of the last 4 actions taken. In this case the last 4 actions correspond to the last 4 notes the agent played, rather than the last 4 drum components struck.

The only differences between MA and RA-0 are the expert trajectories used and the state and action spaces. These differences are discussed in the following section, Section 6.2. Following this, Section 6.3 then discusses the results of the experiments run with MA.

6.2 Implementation

6.2.1 Action Space

Each cell in Figure 6.1 contains a value which corresponds to a choice of action. Each action corresponds to a particular note. The range of values that the cells can take on is from 0 to 25 which correspond to two full octaves starting and ending with the note D. Additionally, an action was provided which corresponded to no note being played was provided (action 0). The full range of actions and their note values is shown in Figure 6.2.

1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0
1	3	10	10	10	8	8	8
0	15	15	0	13	13	0	0
0	18	18	0	17	17	0	0
0	20	20	0	18	18	0	0

(a) An expert melody as series of action numbers

$D3$	$E3$	$F\sharp3$	$F\sharp3$	$F\sharp3$	$E3$	$E3$	$E3$
–	$B3$	$B3$	–	$A3$	$A3$	–	–
$D3$	$E3$	$F\sharp3$	$F\sharp3$	$F\sharp3$	$E3$	$E3$	$E3$
–	$D4$	$D4$	–	$C\sharp4$	$C\sharp4$	–	–
D	E	$B3$	$B3$	$B3$	$A3$	$A3$	$A3$
–	$E4$	$E4$	–	$D4$	$D4$	–	–
–	$G4$	$G4$	–	$F\sharp4$	$F\sharp4$	–	–
–	$A4$	$A4$	–	$G4$	$G4$	–	–

(b) An expert melody as a series of notes

Figure 6.1: Example expert melody in the key of D major. Figure 6.1a shows the melody as a series of action numbers. Figure 6.1b shows the melody as a series of notes.

Note	**Action Number**
rest	0
$D3$	1
$D\sharp 3/E\flat 3$	2
$E3$	3
$F3$	4
$F\sharp 3/G\flat 3$	5
$G3$	6
$G\sharp 3/A\flat 3$	7
$A3$	8
$A\sharp 3/B\flat 3$	9
$B3$	10
$C4$	11
$C\sharp 4/D\flat 4$	12
$D4$	13
$D\sharp 4/E\flat 4$	14
$E4$	15
$F4$	16
$F\sharp 4/G\flat 4$	17
$G4$	18
$G\sharp 4/A\flat 4$	19
$A4$	20
$A\sharp 4/B\flat 4$	21
$B4$	22
$C5$	23
$C\sharp 5/D\flat 5$	24
$D5$	25

Figure 6.2: Notes and their action numbers

The number after the note refers to the octave in which the note belongs. For example $D3$ is a full octave (12 semitones) below $D4$ (it is lower in pitch). Action 0 corresponds to a rest, where no note is played. The next section discusses the state space of the learning agent.

6.2.2 State Space

As was the case with RA-0, the state signal received by this agent were the previous 4 actions taken: (x_0, x_1, x_2, x_3). Each x_i refers to a previous action taken, where x_0 was the most recent action taken and x_3 the least recent. Since the number of actions available to this learning agent was 26 and the history length was 4, this led to a state space of size 26^4. An example expert melody - one in the key of D major - is shown in Figure 6.1

The feature vector ϕ was implemented in the same way as it was for RA-0. For a discussion on this, refer to 4.2.4. The only difference being that MA's ϕ had 26^4 elements as opposed to RA-1's 15^4. This is due to the fact that MA had 26 actions available to it as opposed to RA-0's 15.

6.2.3 Melody Agent Expert Trajectories

The melodies used as expert trajectories for MA were saved as midi files. These expert melodies were, like the drum-beats, structured so that there were 8 beats in each of the 8 measures. Two sets of expert trajectories were written. Both sets spanned two octaves. The first set was written in the key of D major and the second set in the key of E♭ major. Each set contained 20 expert trajectories for a total of 40 expert trajectories. The notes of D major are shown in Figure 6.3. The notes of E♭ major are shown in Figure 6.4.

D	E	$F\sharp$	G	A	B	$C\sharp$	D

Figure 6.3: The notes of the D major scale

$E\flat$	F	G	$A\flat$	$B\flat$	C	D	$E\flat$

Figure 6.4: The notes of the E♭ major

Since only a subset of all the 26 available actions (25 which correspond to notes and 1 which corresponds to a rest where no note is played) are used, one might ask why all these action choices were made available to the learning agent. The reason is to test whether or not the agent was capable of learning to use only the subset of the notes which the expert used and still create original musical pieces.

As was the case with the drum-beat expert trajectories, the rows represent measures and the columns represent beats within the measure. The value in each cell corresponds to the action taken at that particular measure and beat. Each action number represents the value of the note played at that time step. The choice of notes which make up a given melody correspond to action choices made, either by the expert or the learning agent.

6.3 Results - Melody Agent

6.3.1 Introduction

This section provides the results of the experiments run using MA. Three experiments were run and are presented in Sections 6.3.2 – 6.3.4. In the first experiment (Section 6.3.2) As was the case with the RA-0 and RA-1, the parameters $(\gamma, \gamma_2, \epsilon, \epsilon_2, \alpha, \lambda)$ were set in each experiment.

6.3.2 MA Experiment 1 — 1 Expert Trajectory

This section provides the results of the experiment using MA in which one expert trajectory was given to the learning agent. The expert trajectory used in the experiment is the one shown in Figure 6.1. Both greedy trajectories and softmax trajectories were generated by the agent. Figure 6.5 shows the greedy trajectories and Figure 6.6 shows the softmax trajectories. The greedy trajectories only contain notes from the single expert trajectory given to the agent, unlike the softmax trajectories which contain notes outside of those used in the expert trajectories. These notes are shown in red. The results of this experiment show that when using the greedy policy for creating trajectories, the learning agent is able to recombine a single expert trajectory without deviating from the notes used in that trajectory. The learning agent is not able to exactly replicate the expert trajectory. It is conjectured that using a state space like the one used in RA-1 would allow the learning agent to do so. The use of the state space used in RA-1 was not investigated as, unlike drum-beats, it is not important in this case for actions to appear in the same beat that they did in the expert trajectories. What is important is that the learning agent be able to generate notes which are consistently in one key. Since softmax will occasionally choose sub-optimal actions, as expected, the softmax trajectories occasionally contain notes not found in the expert trajectory. The following section presents the results where the learning agent was run using 20 expert trajectories.

1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0

(a)

1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0
1	3	10	10	10	8	8	8
0	0	1	3	10	10	10	8
8	8	0	0	1	3	10	10
10	8	8	8	0	0	1	3
10	10	10	8	8	8	0	0
1	3	10	10	10	8	8	8

(b)

1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0
1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0
1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0
1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0

(c)

1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	15
15	0	13	13	0	12	12	0
0	1	3	5	5	5	3	3
3	0	10	10	0	8	8	0
15	15	0	13	13	0	12	12
0	0	1	3	5	5	5	3
3	3	0	10	10	0	8	8

(d)

Figure 6.5: MA, Experiment 1 — trajectories generated using the greedy policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/MA/exp1/6.5/`

1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0

(a)

1	3	5	5	5	3	3	3
0	13	13	0	12	12	0	0
1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	0
1	3	10	10	10	8	8	8
0	0	1	3	5	5	5	3
3	3	0	13	13	0	12	12
0	0	1	3	10	10	10	8

(b)

1	3	5	5	5	3	3	3
0	13	13	0	12	2	12	13
19	1	20	22	4	20	10	24
21	20	13	16	17	5	13	14
17	18	1	13	22	7	7	5
23	12	18	3	7	13	15	11
8	3	17	7	17	12	24	16
22	19	19	10	8	6	2	22

(c)

1	3	5	5	5	3	3	3
0	10	10	0	8	8	0	15
16	17	0	9	6	19	8	22
7	10	4	14	3	23	18	21
5	24	7	8	4	1	24	17
19	12	13	18	14	0	23	22
6	10	18	15	1	0	6	21
11	14	18	13	20	7	18	13

(d)

Figure 6.6: MA, Experiment 1 — trajectories generated using the softmax policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/MA/exp1/6.6/`

6.3.3 MA Experiment 2 — 20 D Major Expert Trajectories

This section provides the results of using 20 D major expert trajectories with MA. Once again, greedy trajectories as well as softmax trajectories were generated. Several of the greedy trajectories are shown in Figure 6.7 and several of the softmax trajectories are shown in Figure 6.8. The greedy trajectories contain only notes from D major. The learning agent has managed to recombine the expert trajectories in new ways without deviating from the key of D major. Only occasionally did a softmax trajectory contain notes outside of the key of D major. These notes are highlighted in red, in Figure 6.8. The results of using softmax action selection are considered successful, as only rarely did these trajectories deviate from the key of D major. The results of this experiment show that the learning agent has learned to generate new melodies within the key of D major, without being explicitly told that it should only play notes in this key. The following section provides results of the experiment run using 40 expert trajectories; 20 in the key of D major, and 20 in the key of E♭ major. The purpose of this experiment was to test whether the learning agent could generate new melodies which began in either key, and stayed in that key.

13	1	8	6	10	8	5	1
1	0	0	8	10	0	3	0
1	0	1	1	0	0	8	10
0	3	0	1	0	1	1	0
0	8	10	0	3	0	1	0
1	1	0	0	8	10	0	3
0	1	0	1	1	0	0	8
10	0	3	0	1	0	1	1

(a)

1	3	5	3	1	0	0	3
5	0	10	0	8	0	6	5
3	1	0	0	3	5	0	10
0	8	0	6	5	3	1	0
0	3	5	0	10	0	8	0
6	5	3	1	0	0	3	5
0	10	0	8	0	6	5	3
1	0	0	3	5	0	10	0

(b)

15	12	8	10	12	13	0	1
13	8	10	12	13	0	1	13
8	10	12	13	0	1	13	8
10	12	13	0	1	13	8	10
12	13	0	1	13	8	10	12
13	0	1	13	8	10	12	13
0	1	13	8	10	12	13	0
1	13	8	10	12	13	0	1

(c)

0	0	0	18	18	0	17	18
18	20	20	13	10	13	8	8
10	18	17	15	12	13	15	8
13	12	13	1	15	13	12	10
8	6	5	6	5	3	13	1
1	3	5	0	10	6	8	6
6	5	3	13	1	1	3	5
0	10	6	8	6	6	5	3

(d)

Figure 6.7: MA, Experiment 2 — trajectories generated using the greedy policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/MA/exp2/6.7/`

6.3.4 MA Experiment 3 — 40 Mixed Expert Trajectories

The purpose of this experiment was to test whether MA could learn about key signatures when given expert trajectories in 2 different key signatures. 40 expert trajectories were given to MA. 20 of these trajectories were in the key of D major. The other 20 were in the key of E♭ major. Once again both greedy trajectories and softmax trajectories were generated at every iteration of the apprenticeship learning algorithm. It was hoped that the learning agent would generate new trajectories which would not deviate from a certain key. That is, if the trajectory began in the key of D major, then it was hoped that the learning agent would continue to play notes in D major for the rest of the trajectory.

Figure 6.9 shows 4 of the greedy trajectories generated by MA. The trajectories shown in Figures 6.9a and 6.9b contain only notes from the D major scale while the trajectories shown in Figures 6.9a and 6.9a contain only notes from the E♭ major scale. These results demonstrate that the learning agent had learned about key signatures. The learning agent created trajectories in which its choice of notes was consistent. However, in the softmax trajectories — as expected — the trajectories are more erratic. For example, the trajectory shown in Figure 6.10a contains the actions 1, 2, 3, 4, 5 and 6, corresponding the the notes $D3$, $D\sharp 3/E\flat 3$, $E3$, $F3$ $F\sharp 3/G\flat 3$ and $G3$, corresponding to notes of both the D major scale as well as the E♭ major scale.

0	0	0	18	18	0	17	18
18	20	17	16	18	24	15	5
9	23	22	23	24	17	24	13
19	10	6	3	23	14	23	11
24	23	15	18	21	21	11	15
15	0	2	4	23	22	15	12
8	10	12	13	0	15	0	12
10	0	8	1	6	1	5	13

(a)

5	1	0	1	0	0	12	13
0	15	0	12	10	0	8	1
6	5	3	1	1	3	3	5
0	8	0	1	1	0	3	0
10	8	5	1	1	0	3	0
1	0	0	1	18	18	1	17
17	1	1	3	3	5	0	18
0	13	15	1	15	10	0	18

(b)

0	0	0	18	18	0	17	18
18	20	20	0	10	0	8	8
0	15	15	0	13	0	8	0
1	1	0	3	0	10	8	6
8	6	5	3	13	1	0	5
0	1	3	5	0	18	0	17
18	18	20	20	0	18	18	20
20	13	10	13	8	13	6	13

(c)

15	12	8	10	12	13	12	13
1	15	13	12	12	13	12	5
1	13	8	10	10	18	17	15
13	1	10	8	13	12	6	1
13	1	10	8	13	12	13	15
12	13	15	12	13	15	12	13
15	13	1	10	8	13	12	13
10	8	6	10	8	5	1	1

(d)

Figure 6.8: MA, Experiment 2 — trajectories generated using the softmax policy. The cells in red are notes not in the key of D major. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/MA/exp2/6.8/`

6.3.5 Discussion

The results of the experiments run using MA show that the learning agent is capable to taking in one expert trajectory and recombining it without deviating from the notes used in that expert trajectory — this is demonstrated by the results of Experiment 1 (Section 6.3.2). The results of the following experiment, presented in Section 6.3.3, show that the learning agent is capable of recombining multiple expert trajectories without deviating from the notes used in those trajectories. Finally, Section 6.3.4 presents the results of Experiment 3, in which multiple expert trajectories in 2 key signatures are presented to the learning agent. The learning agent has learned to recombine the expert trajectories given to it without deviating from the key, within the trajectory. MA has learned to generate melodies in a certain key.

The previous observations all pertain to the greedy trajectories. The softmax trajectories were not as successful; in all experiments they contained notes outside of the intended key signatures.

6.4 Conclusion

This section discussed the implementation of MA as well as the results of the experiments performed using it. It was implemented in much the same way as RA- with the differences being the expert trajectories given to it and its state space.

1	0	0	1	3	5	0	5
0	1	3	5	0	5	0	1
3	5	0	5	0	1	3	5
0	5	0	1	3	5	0	5
0	1	3	5	0	5	0	1
3	5	0	5	0	1	3	5
0	5	0	1	3	5	0	5
0	1	3	5	0	5	0	1

(a)

1	3	5	3	1	1	0	5
0	5	3	0	3	0	10	8
6	0	0	1	15	10	0	18
0	13	15	1	15	10	0	18
0	13	15	1	15	10	0	18
0	13	15	1	15	10	0	18
0	13	15	1	15	10	0	18
0	13	15	1	15	10	0	18

(b)

2	4	6	6	6	1	1	9
11	1	4	1	2	1	2	2
1	1	11	9	1	6	1	1
9	11	1	4	1	2	1	2
2	1	1	11	9	1	6	1
1	9	11	1	4	1	2	1
2	2	1	1	11	9	1	6
1	1	9	11	1	4	1	2

(c)

2	1	6	1	6	4	1	4
1	11	9	6	2	2	4	4
6	1	6	4	1	4	1	11
9	6	2	2	4	4	6	1
6	4	1	4	1	11	9	6
2	2	4	4	6	1	6	4
1	4	1	11	9	6	2	2
4	4	6	1	6	4	1	4

(d)

Figure 6.9: MA, Experiment 3 — trajectories generated using the greedy policy. The trajectories shown in Figures 6.9a and 6.9b are in the key of D major while the trajectories shown in Figures 6.9c and 6.9d are in the key of E♭ major. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/MA/exp3/6.9/`

14	18	20	13	4	8	15	17
9	3	17	11	4	19	11	7
14	24	24	14	21	16	7	11
12	2	14	1	23	18	23	18
20	6	5	10	24	19	12	18
11	19	1	6	20	19	17	23
2	6	19	7	21	14	21	6
23	11	3	24	13	2	23	23

(a)

0	0	0	18	18	0	17	17
0	0	0	20	20	0	10	0
24	2	18	14	15	10	8	22
17	16	5	10	15	11	8	15
14	3	24	9	3	11	12	13
14	20	18	13	1	19	5	4
13	20	12	18	10	12	23	11
5	5	5	13	17	18	23	3

(b)

13	17	7	14	15	1	18	21
3	24	5	14	10	14	9	12
13	12	11	9	19	13	15	24
14	8	20	11	4	13	11	11
13	3	15	12	10	21	6	18
12	15	12	11	6	17	15	11
18	24	20	6	16	16	3	20
17	13	17	22	1	10	22	10

(c)

13	1	8	6	10	13	24	23
2	5	23	14	24	7	12	16
23	7	8	12	15	8	8	19
5	12	17	10	22	17	12	21
14	9	18	0	24	12	19	11
16	13	16	13	15	9	21	19
10	3	5	22	22	14	8	13
3	7	9	20	11	11	16	3

(d)

Figure 6.10: MA, Experiment 3 — trajectories generated using the softmax policy. These can be heard at `http://aiml.cs.wits.ac.za/orry/c/MA/exp3/6.10/`

Chapter 7

Conclusion

Creating a music generating agent is confounded by not being able to easily and manually identify a way to reward "good music". The notion of "good music" is itself so subjective that any meaningful application of the term would have to be viewed within a certain context. Another way to approach the task is to positively reward generated music which has similar features to expert music presented to it, thus eliminating the need to manually specify a reward function which has the ability to discern between good and bad music. In this research a reward function was built up from expert behaviour; the idea being that the expert has some notion of what he considers to be good music and the music he creates are artefacts of this notion. Thus, what the expert considers good music will be rewarded.

Apprenticeship learning via inverse reinforcement learning shows potential as a way to algorithmically generate music. The results presented here show that it is possible to use this algorithm to create new music based on expert music. The approach taken in this research can be improved upon in many ways. The most obvious way to extend this research is to present the learning agent with more expert trajectories. As more trajectories are added, the learning agent will have a greater base from which to extract the rewards which guide the creation of new trajectories. Another way the research can be extended is by providing a deeper note resolution. Currently, the learning agent could only play eighth notes; this can be extended so that the agent is able to play sixteenth notes, allowing for more variety in the type of music it can handle. Further, the agents could be extended to play triplets, quintuplets and even septuplets. Currently, the melody agent and rhythm agent are standalone agents. An interesting approach could be to allow the learning agents to learn in a way that promotes collaborative music generation; for example, if it is seen that whenever a bass drum is played, a certain note is played, provide a mechanism for the learning agent to take that into account. There are many approaches one could take to implement a learning agent which makes use of reinforcement learning for music generation. This research presents one such approach which has been shown to generate new music in unexpected ways.

Appendix A

Sheet Music

A.1 Introduction

Presented in sections A.2, A.3 and A.4 is the sheet music corresponding to the results of the experiments run with RA-0, RA-1 and MA, respectively.

A.2 RA-0

A.2.1 Experiment 1a

This section presents the sheet music corresponding to the results of RA-0, experiment 1a (presented in 4.3.2).

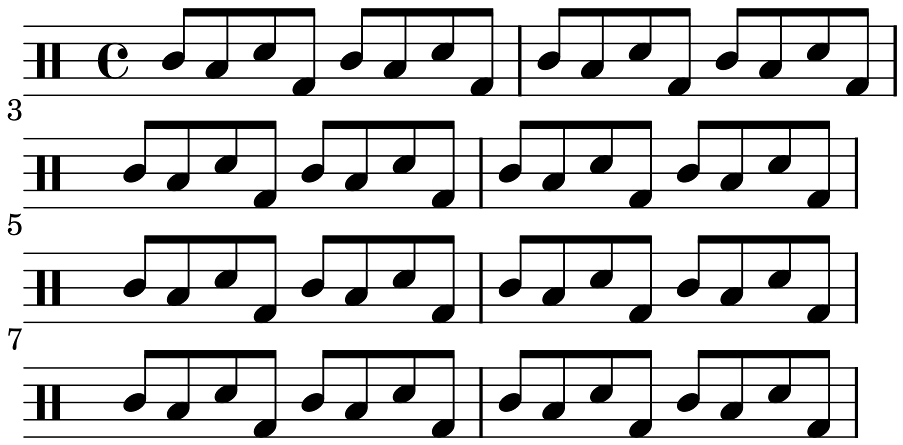

Figure A.1: Sheet music corresponding to Figure 4.4.

A.2.2 Experiment 1b

This section presents the sheet music corresponding to the results of RA-0, experiment 1b (presented in 4.3.2).

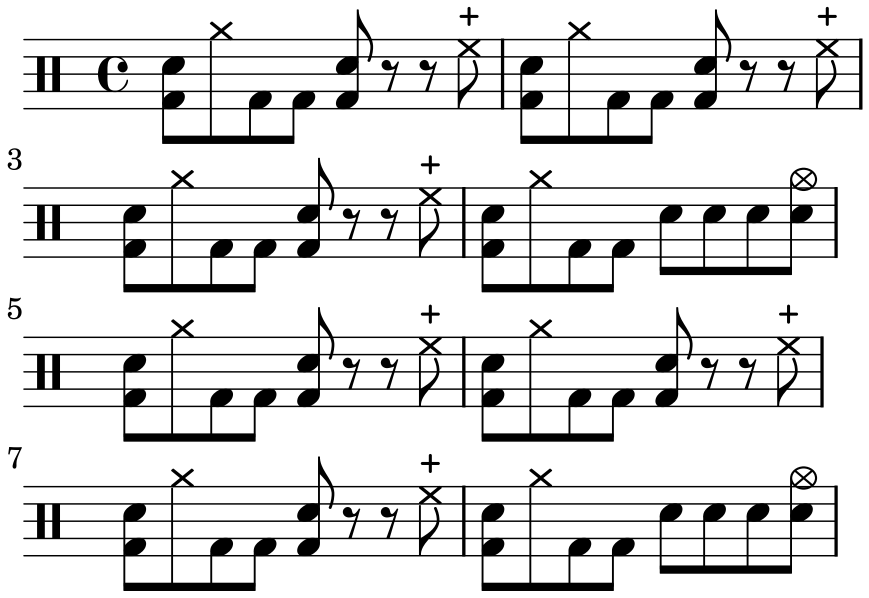

Figure A.2: Sheet music corresponding to Figure 4.5

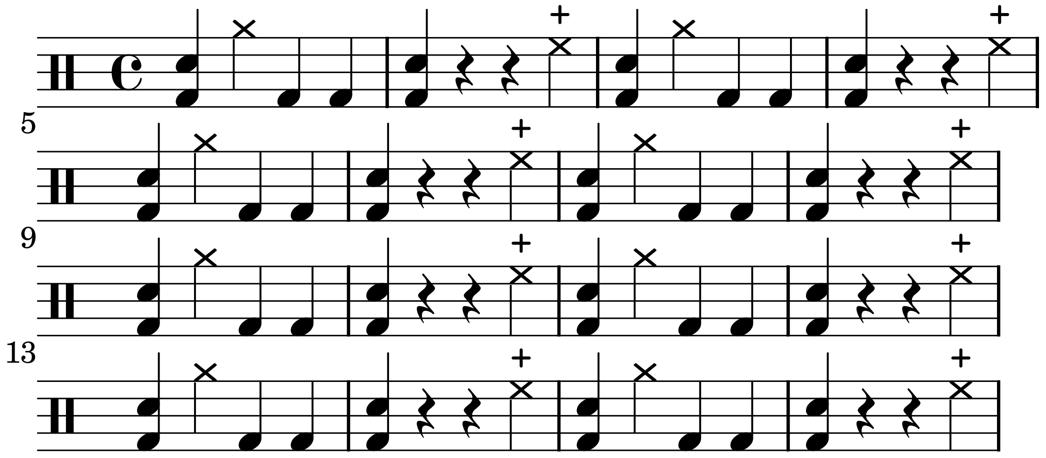

Figure A.3: Sheet music corresponding to Figure 4.6

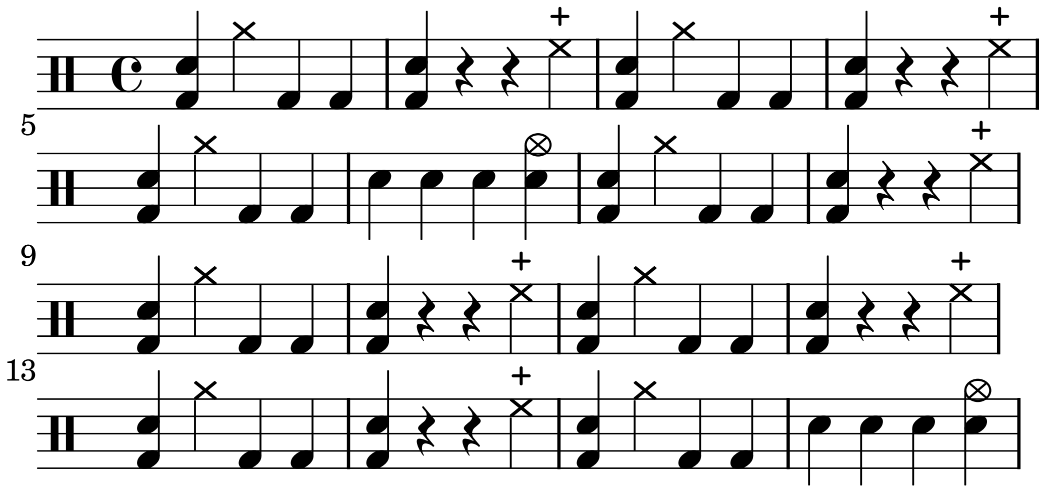

Figure A.4: Sheet music corresponding to Figure 4.7a

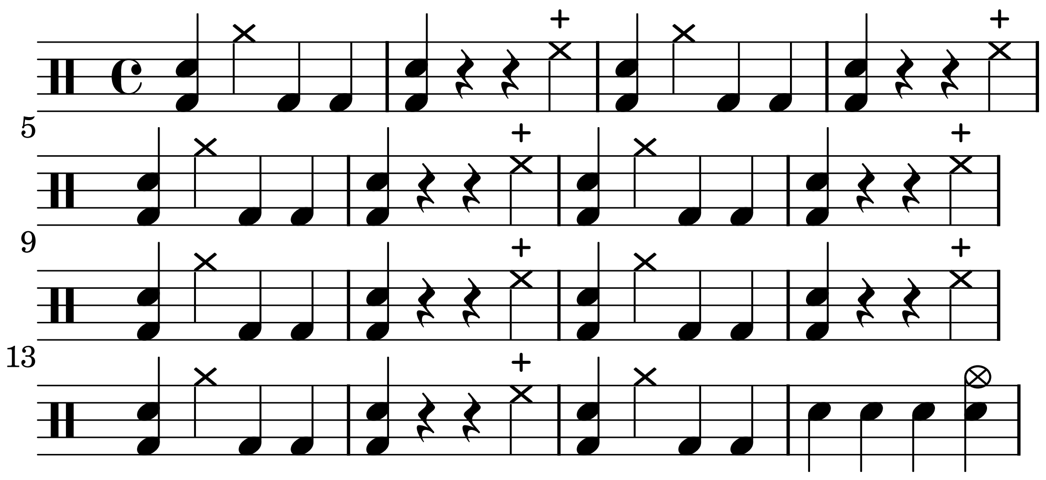

Figure A.5: Sheet music corresponding to Figure 4.7b

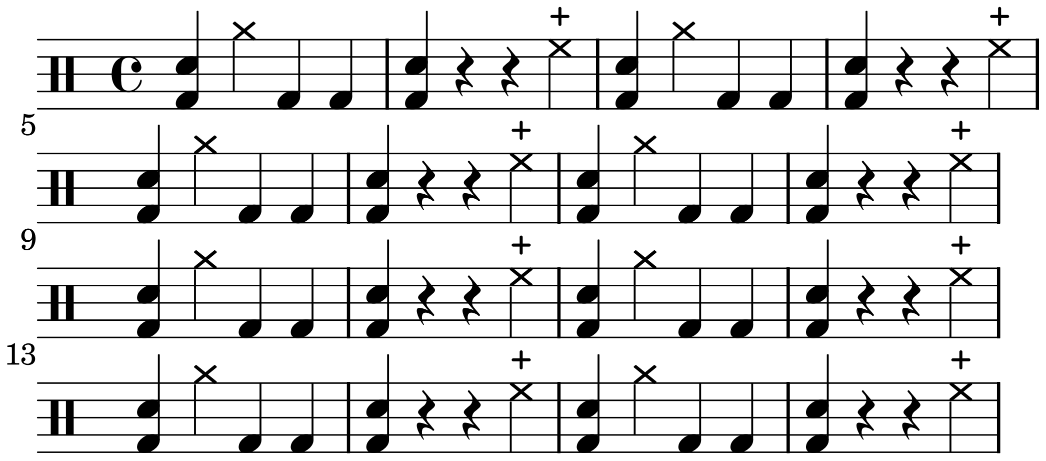

Figure A.6: Sheet music corresponding to Figure 4.7c

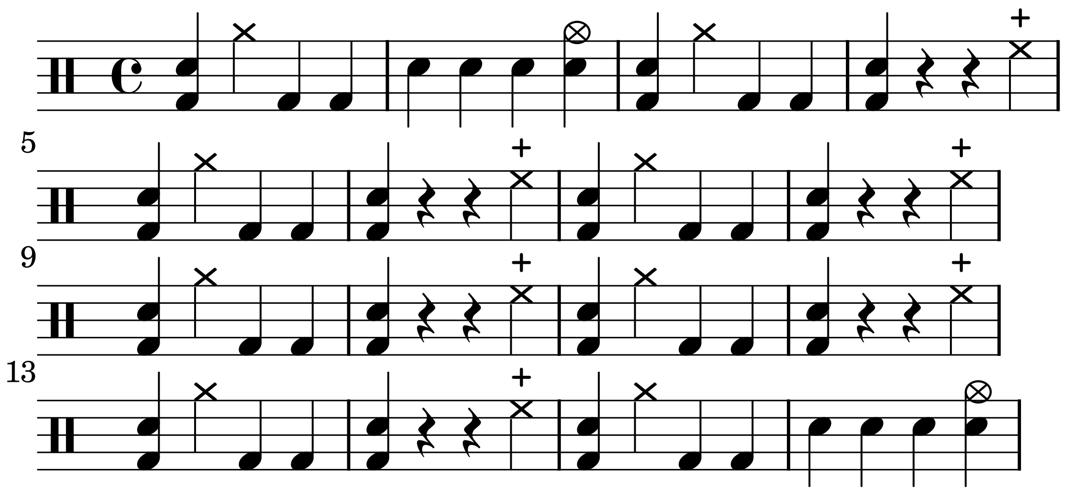

Figure A.7: Sheet music corresponding to Figure 4.7d

A.2.3 Experiment 2

This section presents the sheet music corresponding to the results of RA-0, experiment 2 (presented in 4.3.4).

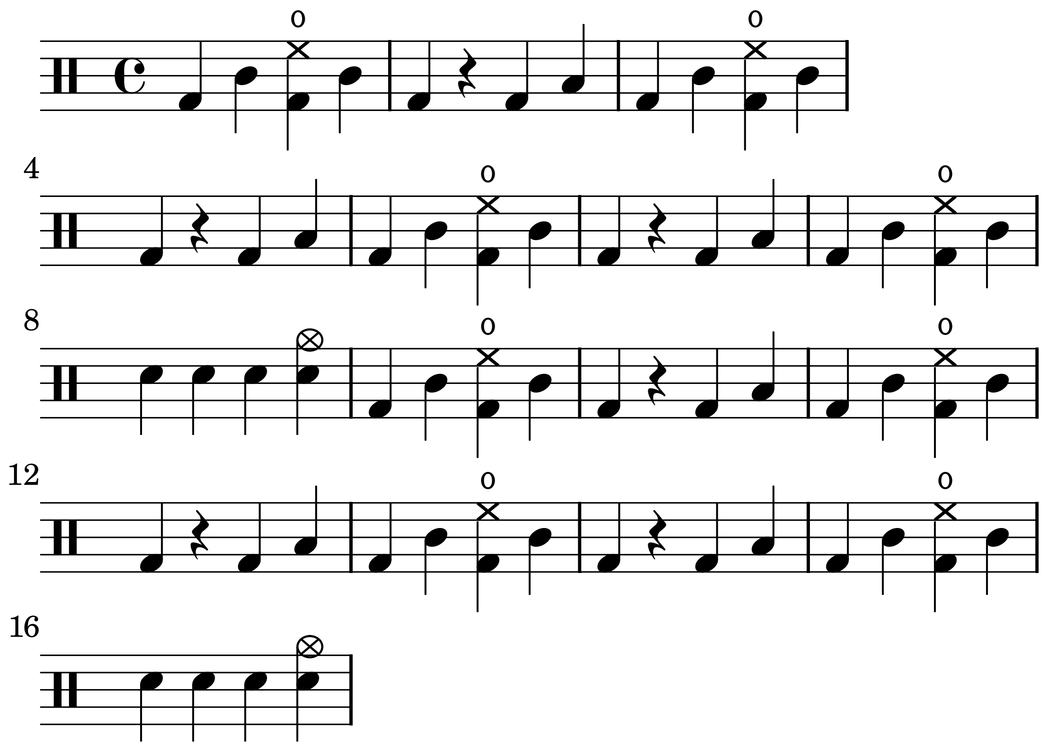

Figure A.8: Sheet music corresponding to Figure 4.8a

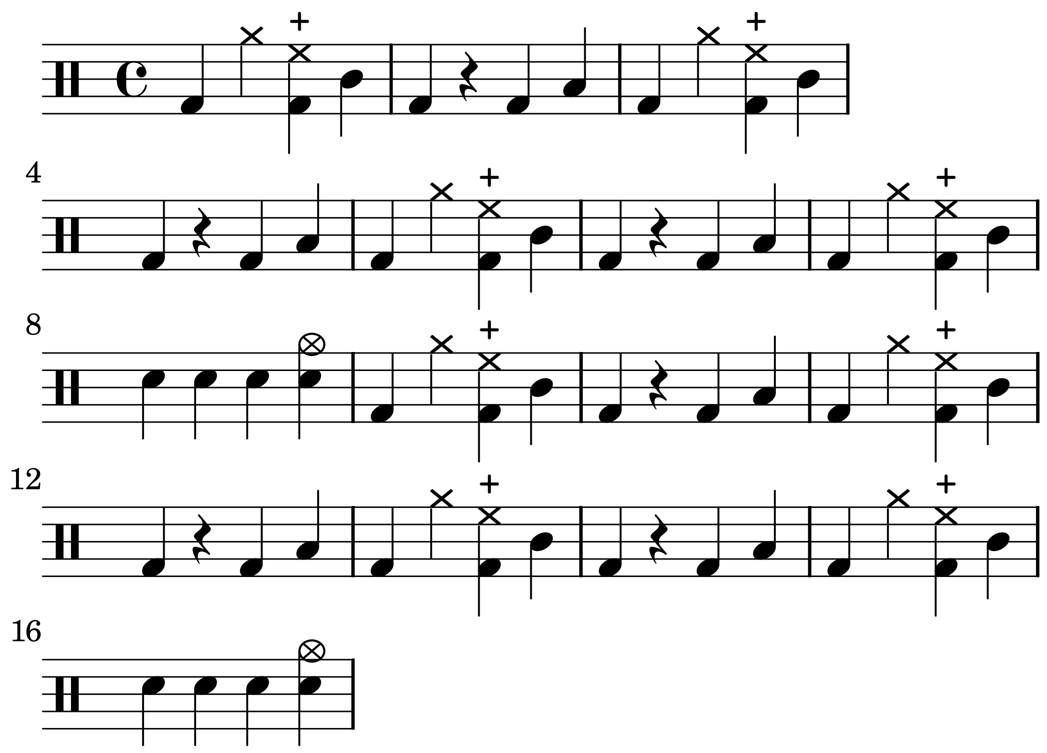

Figure A.9: Sheet music corresponding to Figure 4.8b

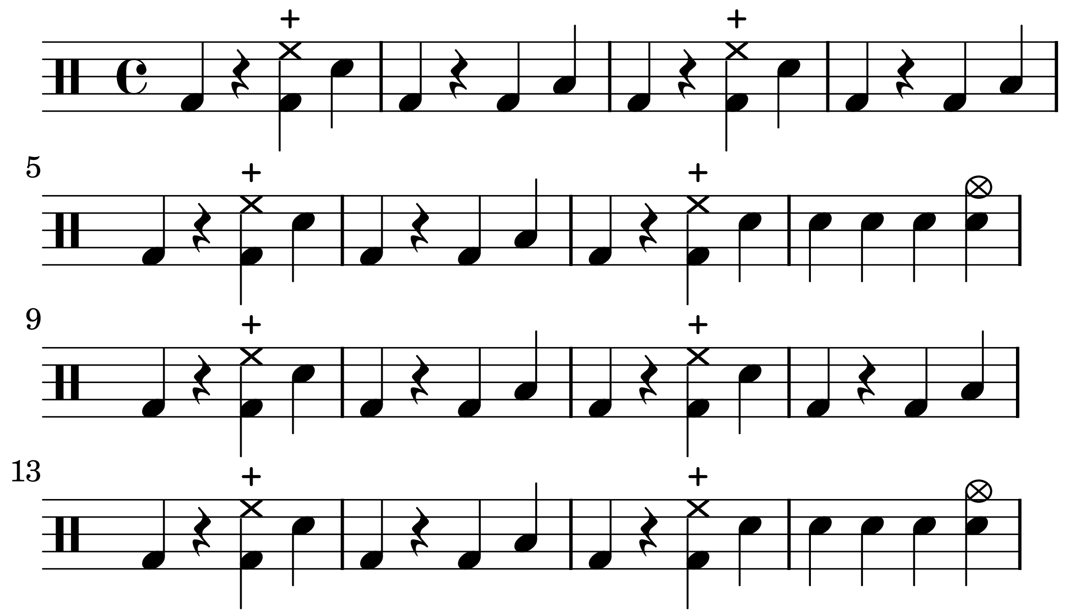

Figure A.10: Sheet music corresponding to Figure 4.8c

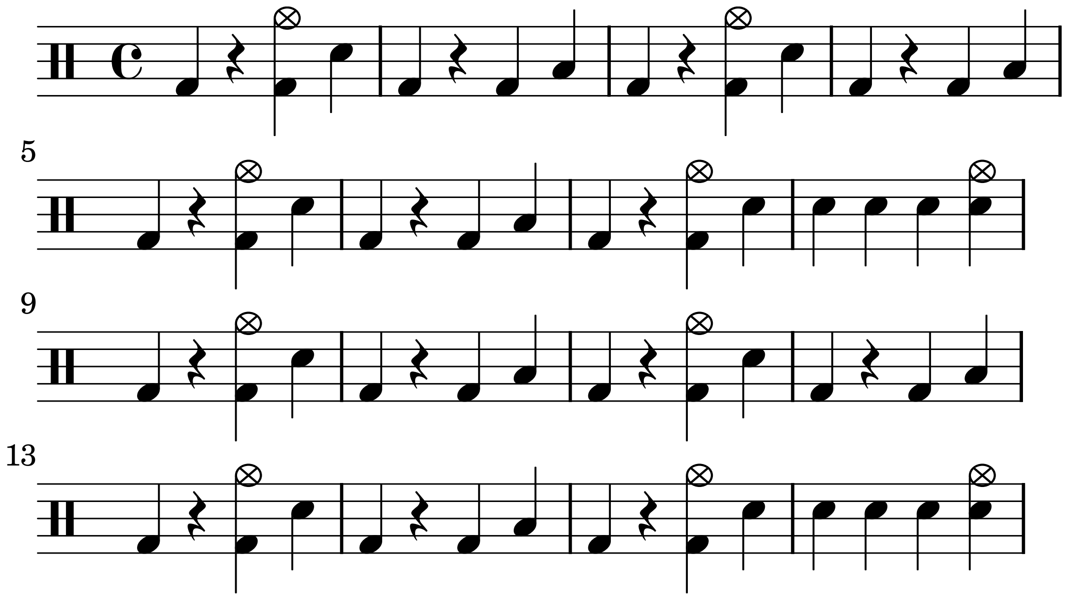

Figure A.11: Sheet music corresponding to Figure 4.8d

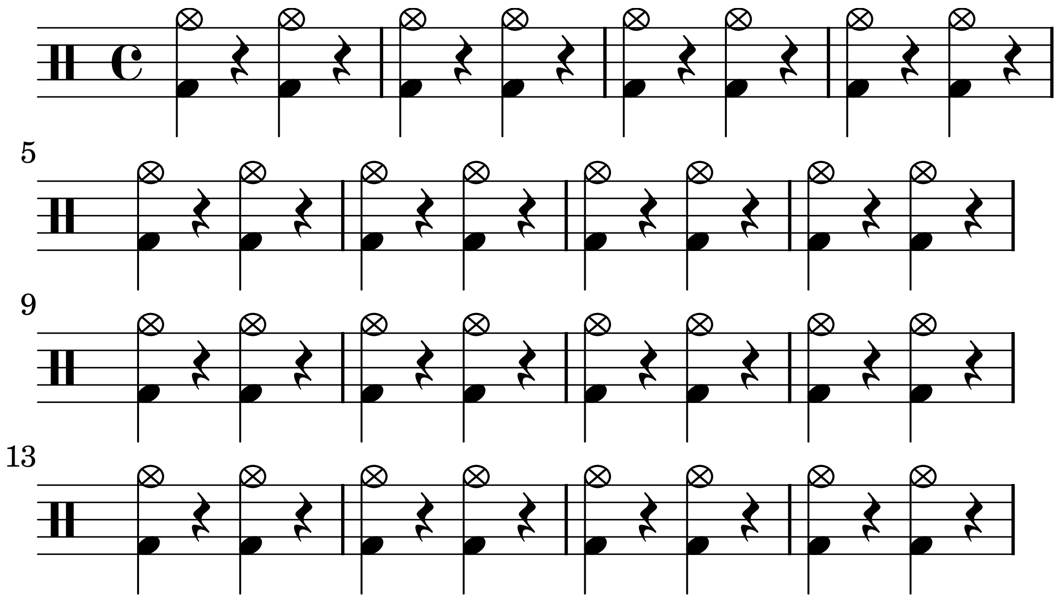

Figure A.12: Sheet music corresponding to Figure 4.9a

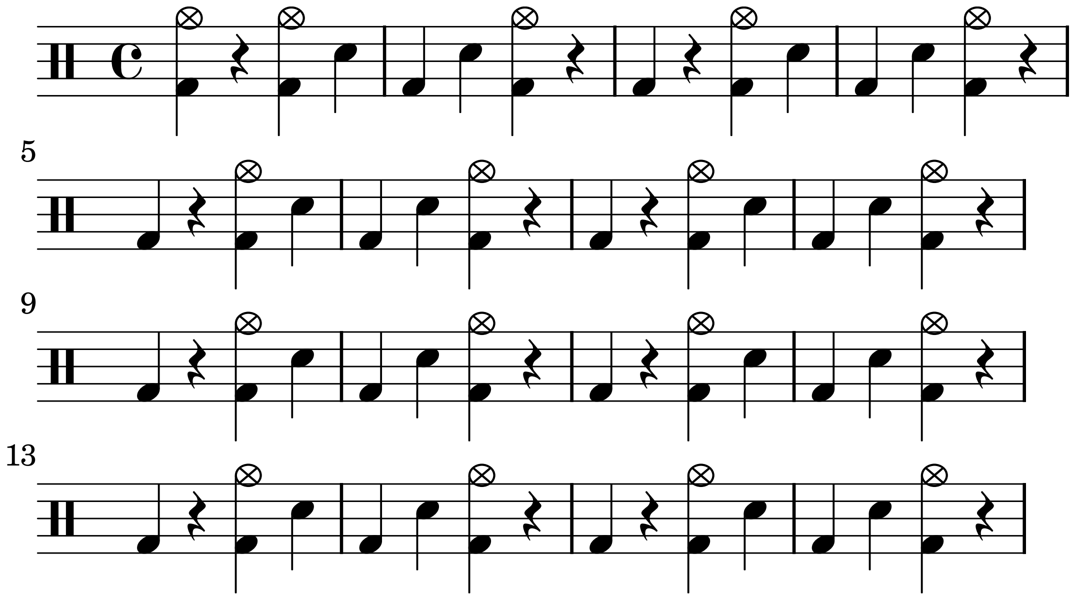

Figure A.13: Sheet music corresponding to Figure 4.9b

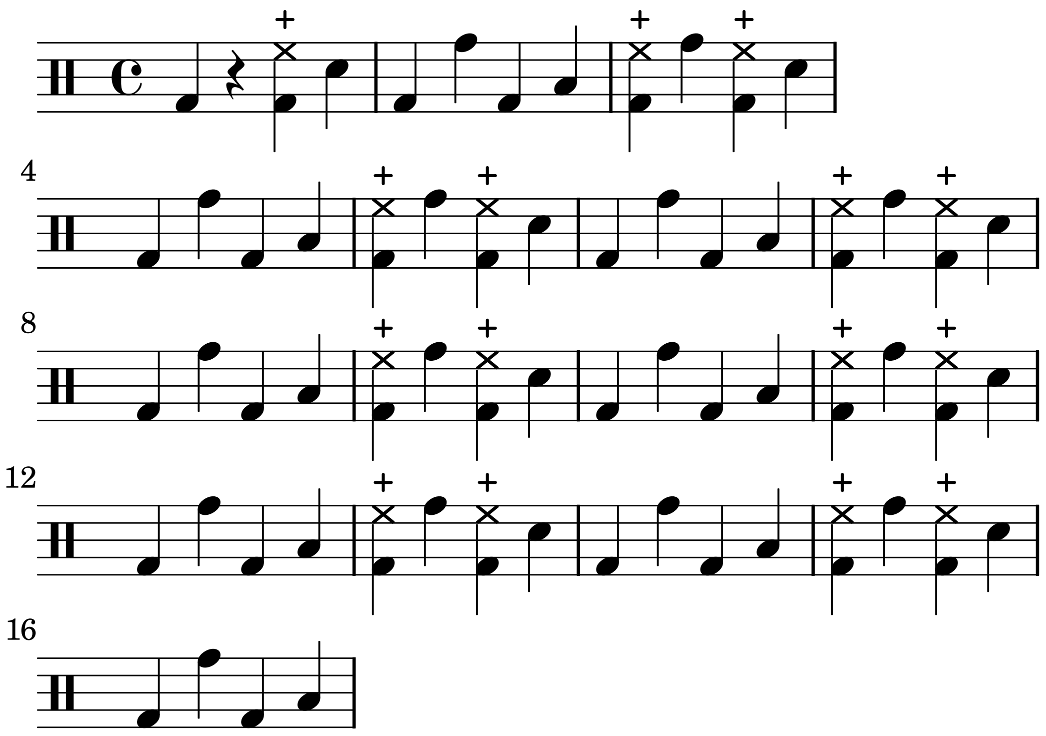

Figure A.14: Sheet music corresponding to Figure 4.9c

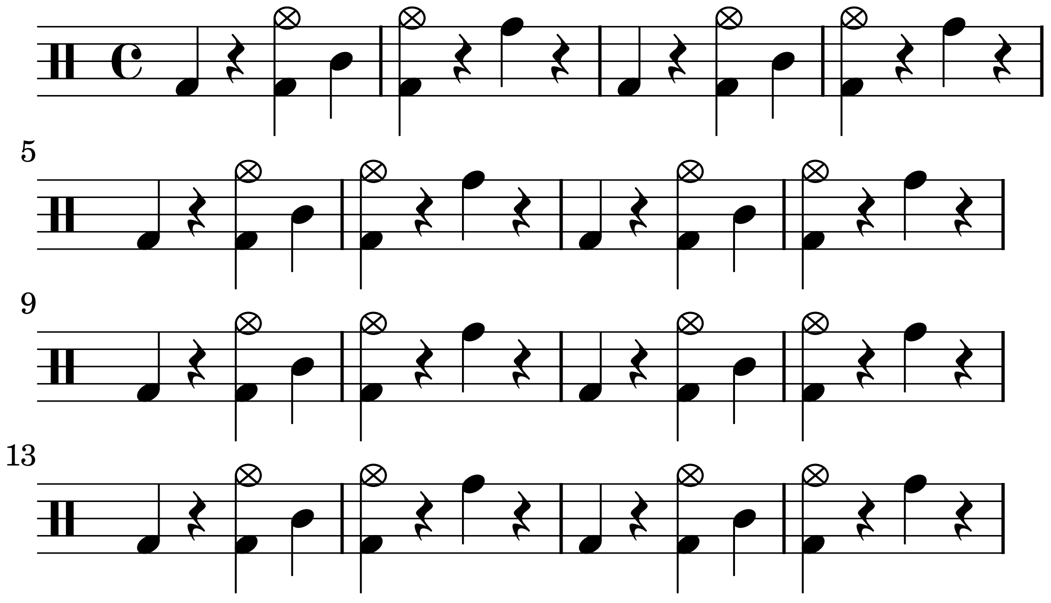
Figure A.15: Sheet music corresponding to Figure 4.9d

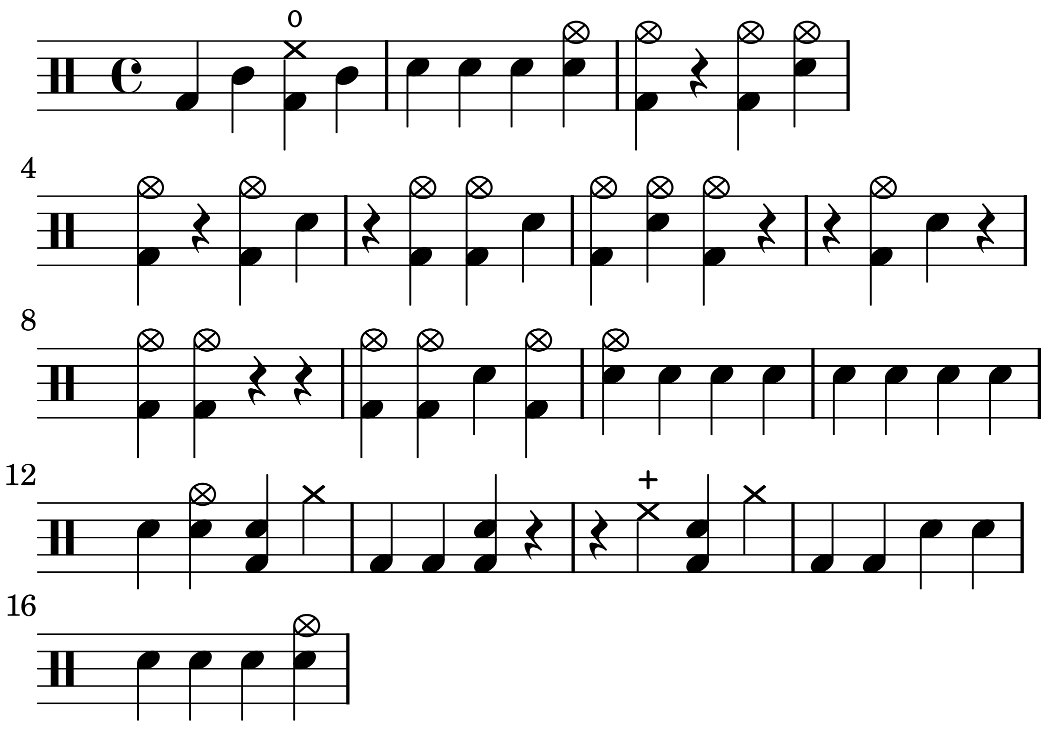

Figure A.16: Sheet music corresponding to Figure 4.10a

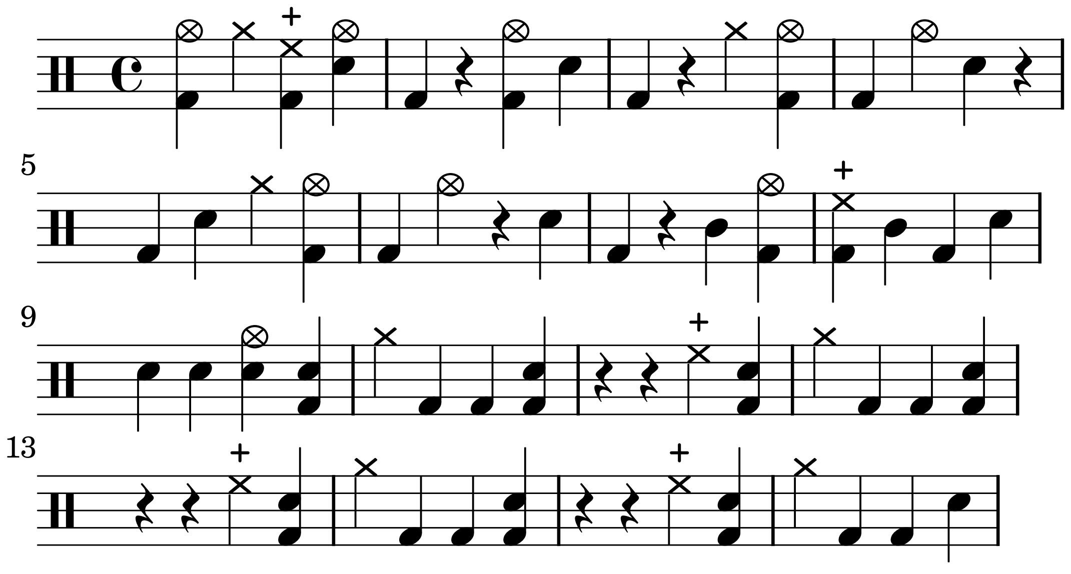

Figure A.17: Sheet music corresponding to Figure 4.10b

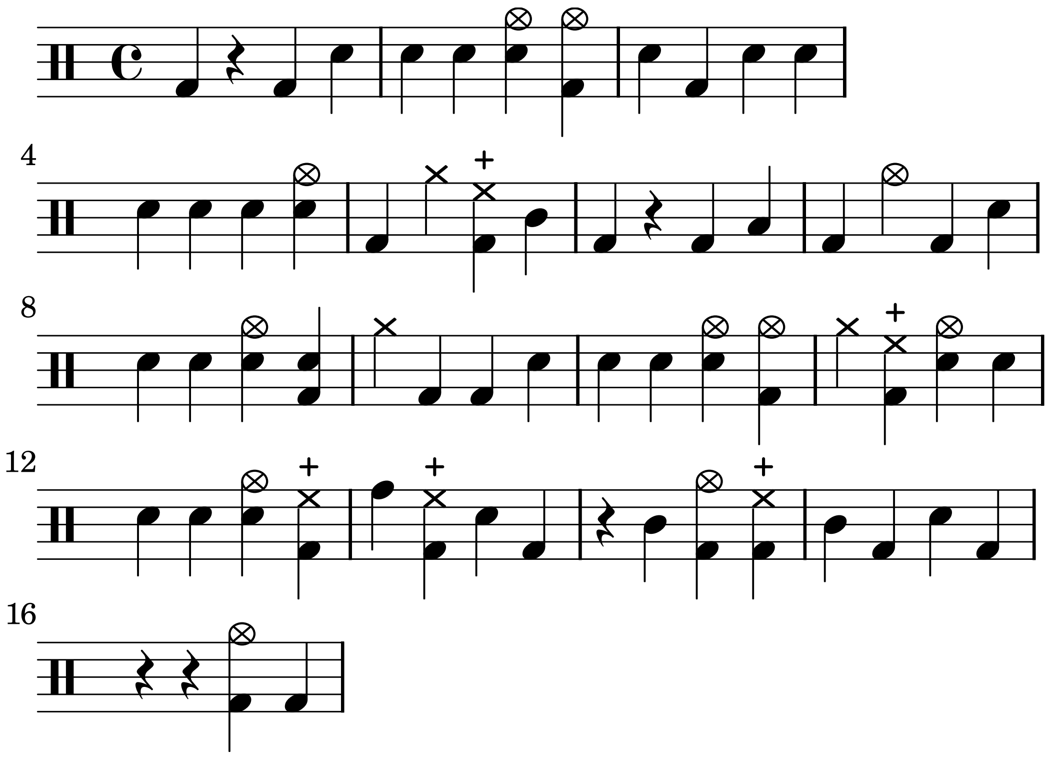

Figure A.18: Sheet music corresponding to Figure 4.10c

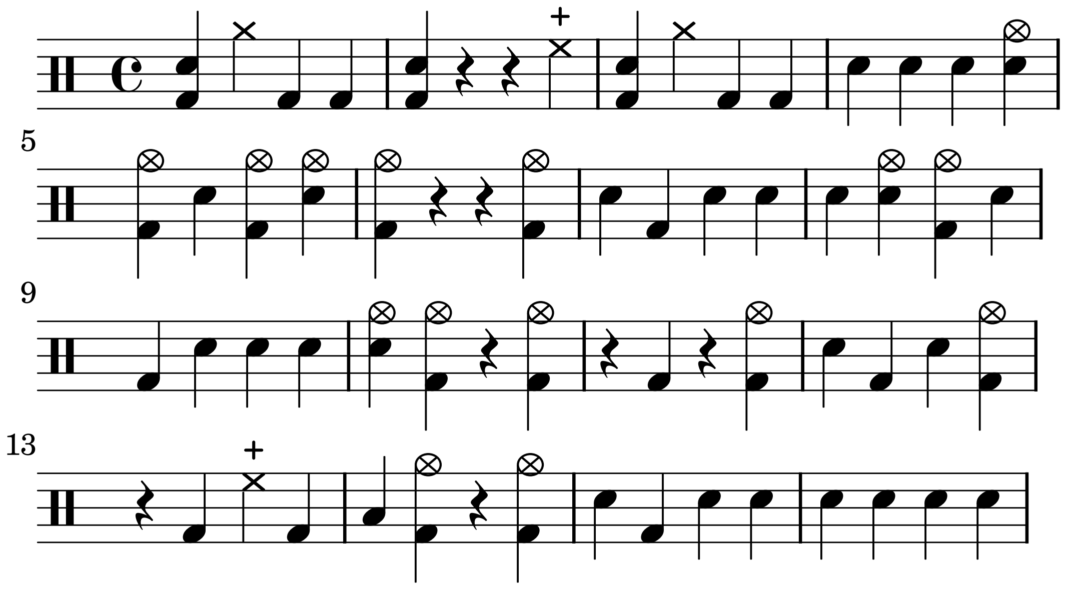

Figure A.19: Sheet music corresponding to Figure 4.10d

A.2.4 Experiment 3

This section presents the sheet music corresponding to the results of RA-0, experiment 3 (presented in 4.3.5).

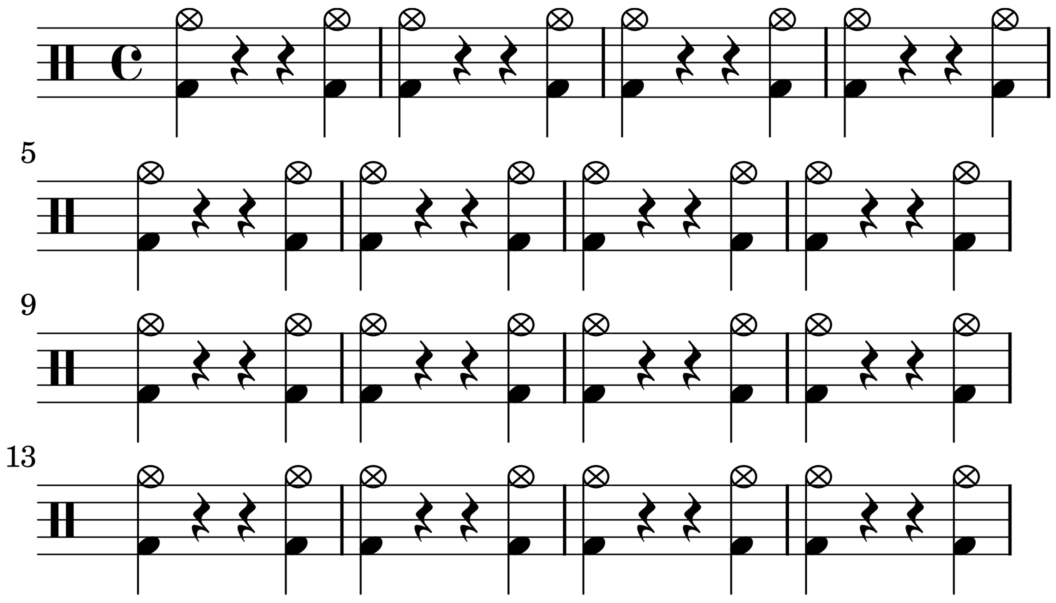

Figure A.20: Sheet music corresponding to Figure 4.11a

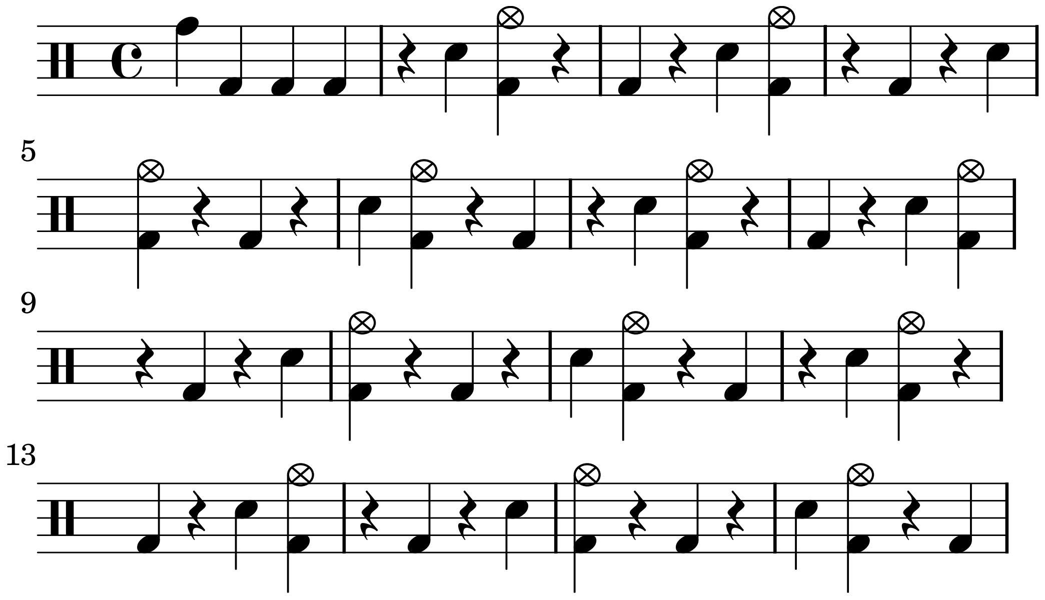

Figure A.21: Sheet music corresponding to Figure 4.11b

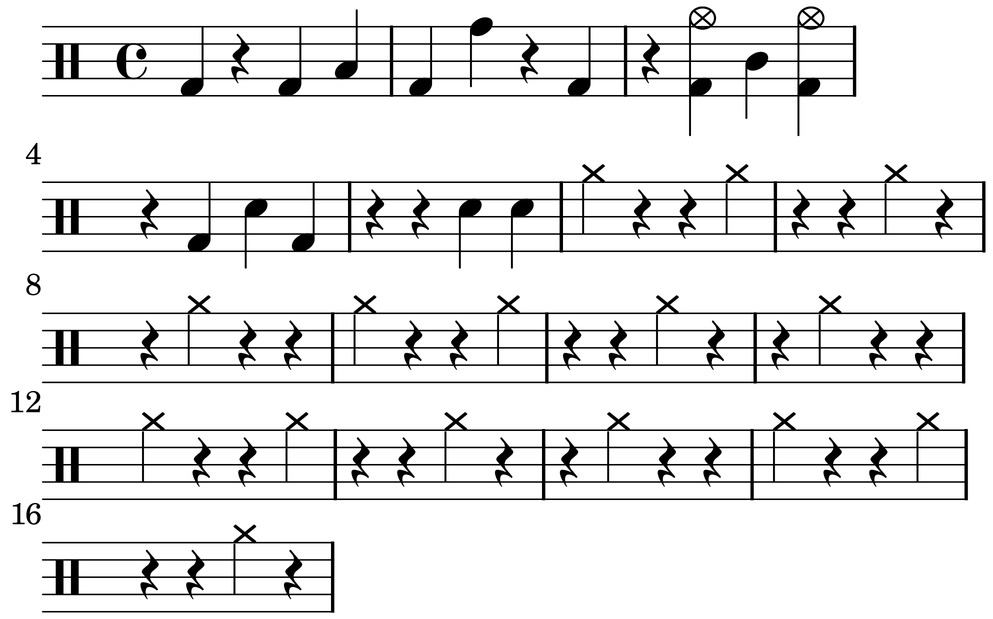

Figure A.22: Sheet music corresponding to Figure 4.11c

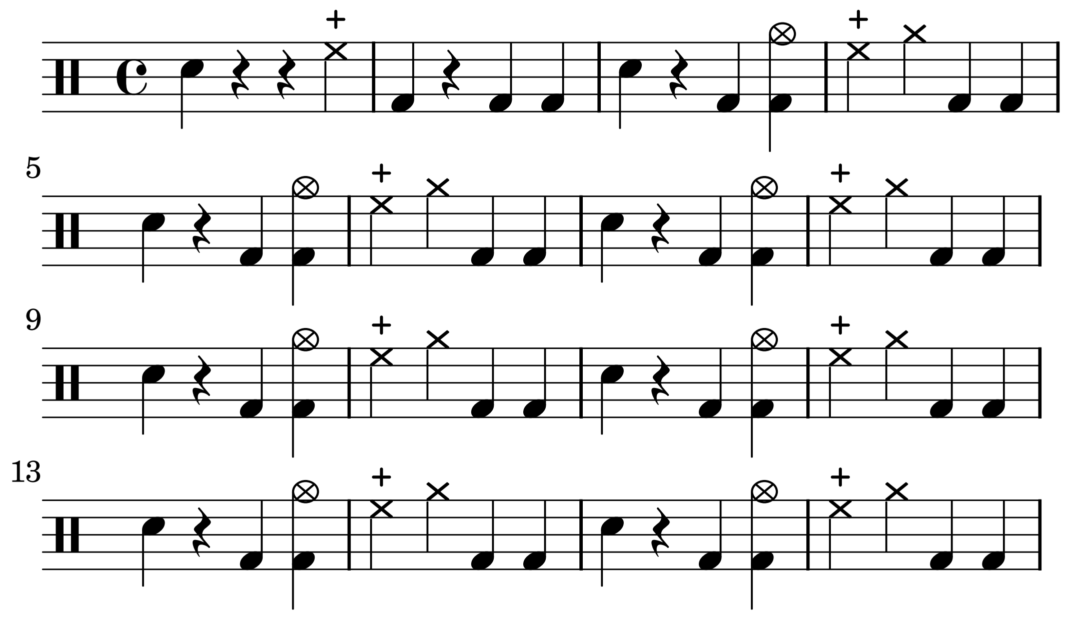

Figure A.23: Sheet music corresponding to Figure 4.11d

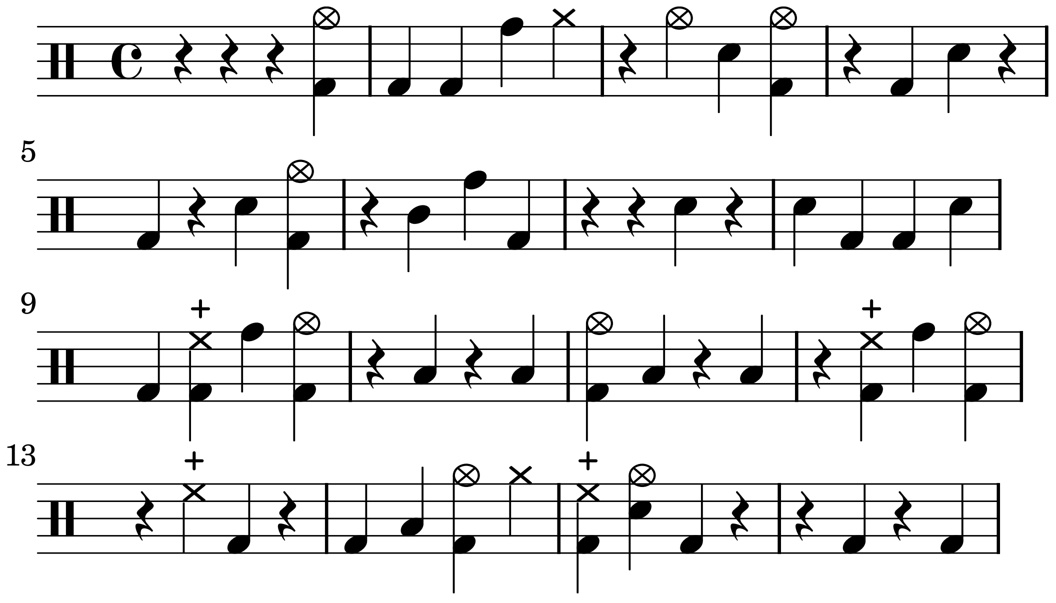
Figure A.24: Sheet music corresponding to Figure 5.4a

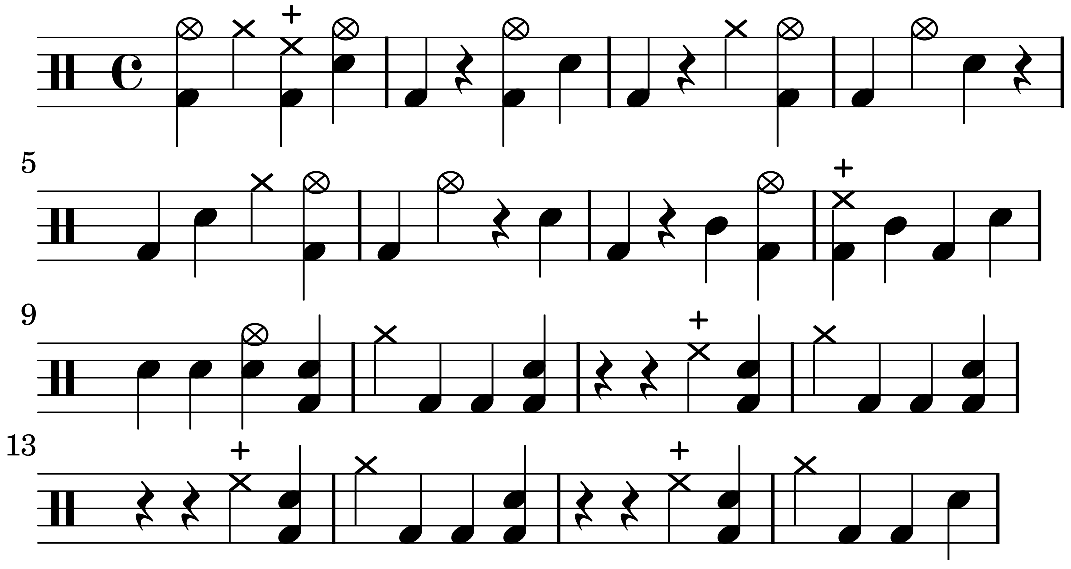
Figure A.25: Sheet music corresponding to Figure 5.4b

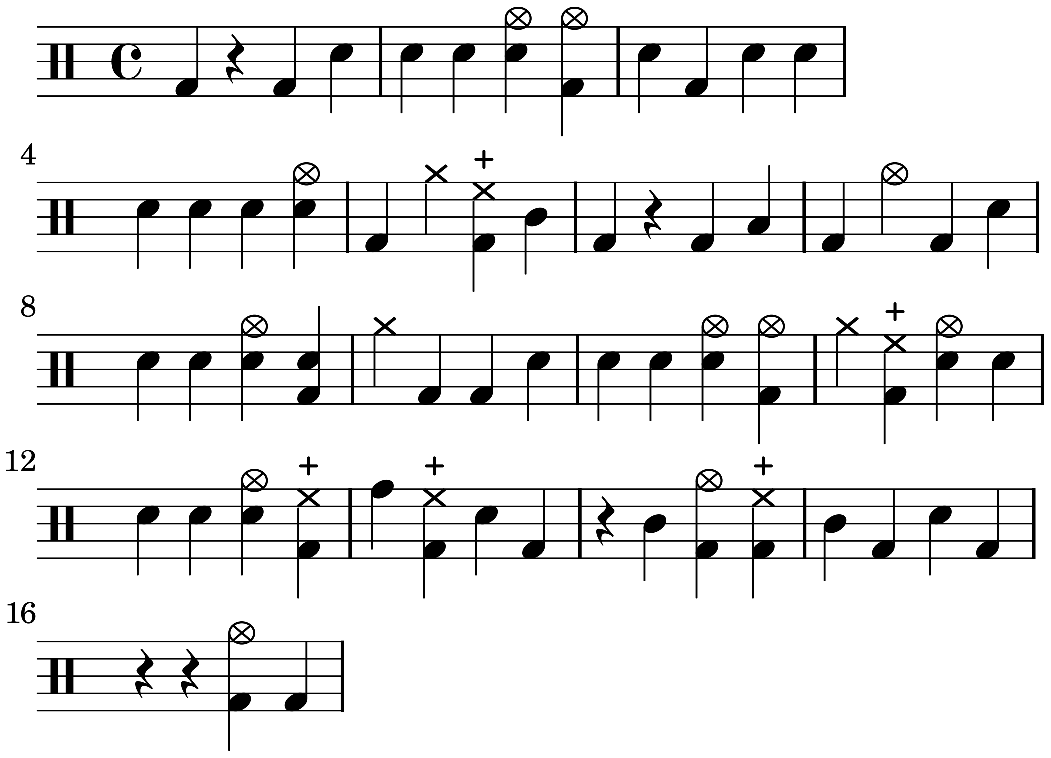

Figure A.26: Sheet music corresponding to Figure 5.4c

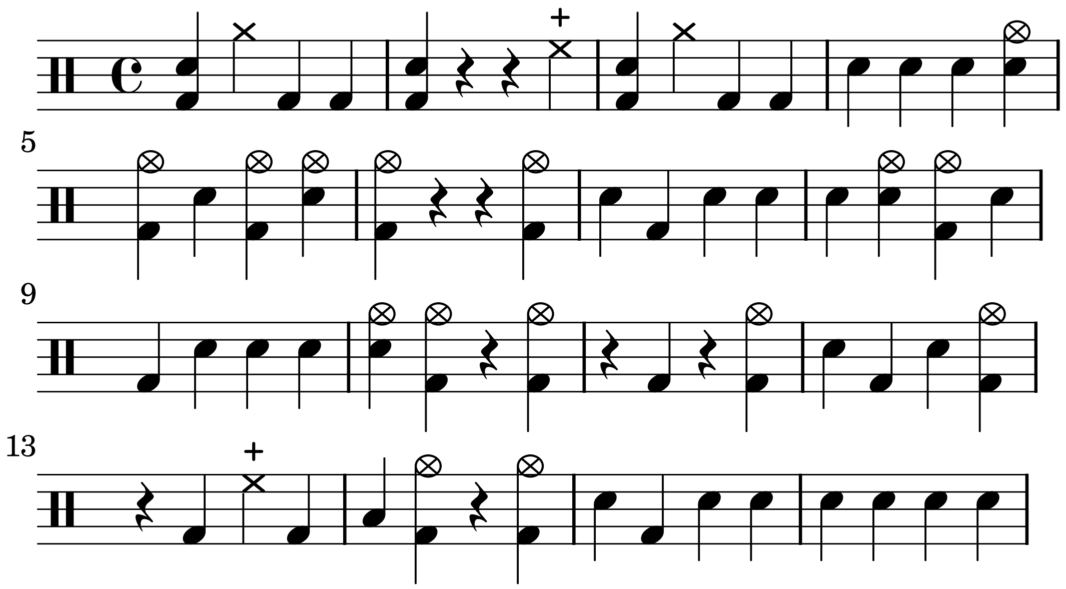

Figure A.27: Sheet music corresponding to Figure 5.4d

A.3 RA-1

A.3.1 Experiment 2

This section presents the sheet music corresponding to the results of RA-1, experiment 2 (presented in 5.3.3).

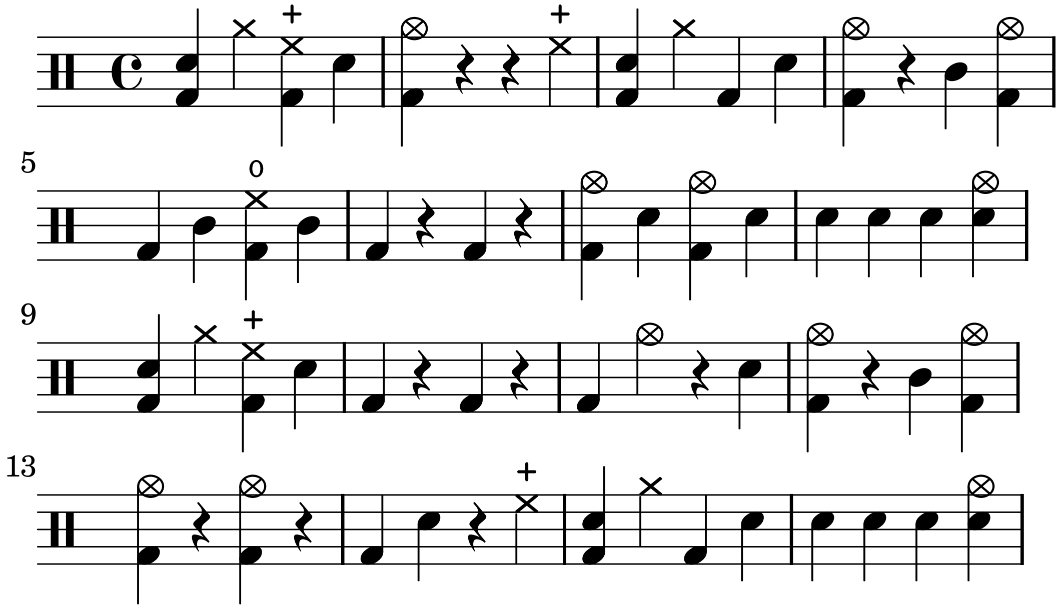

Figure A.28: Sheet music corresponding to Figure 5.1a

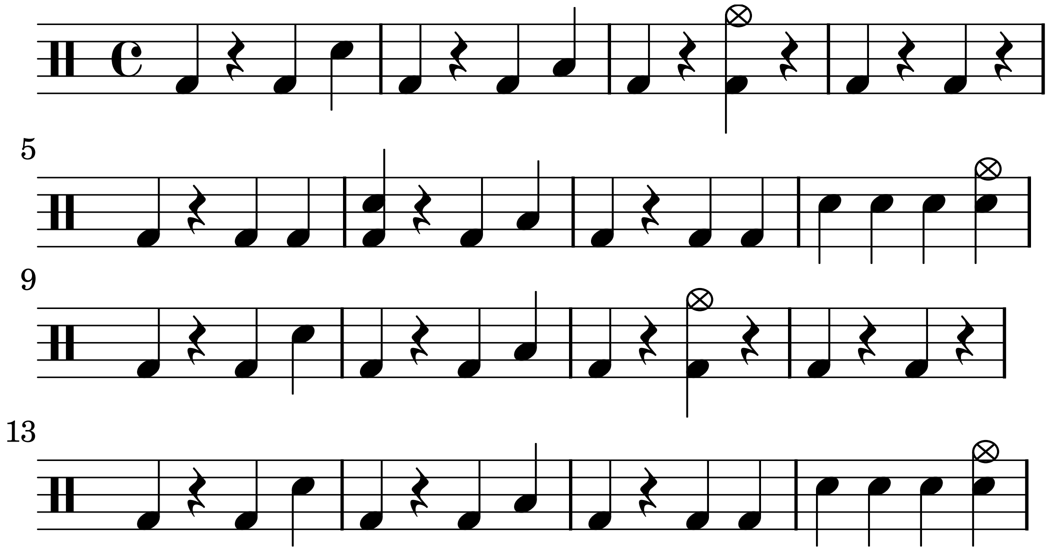

Figure A.29: Sheet music corresponding to Figure 5.1b

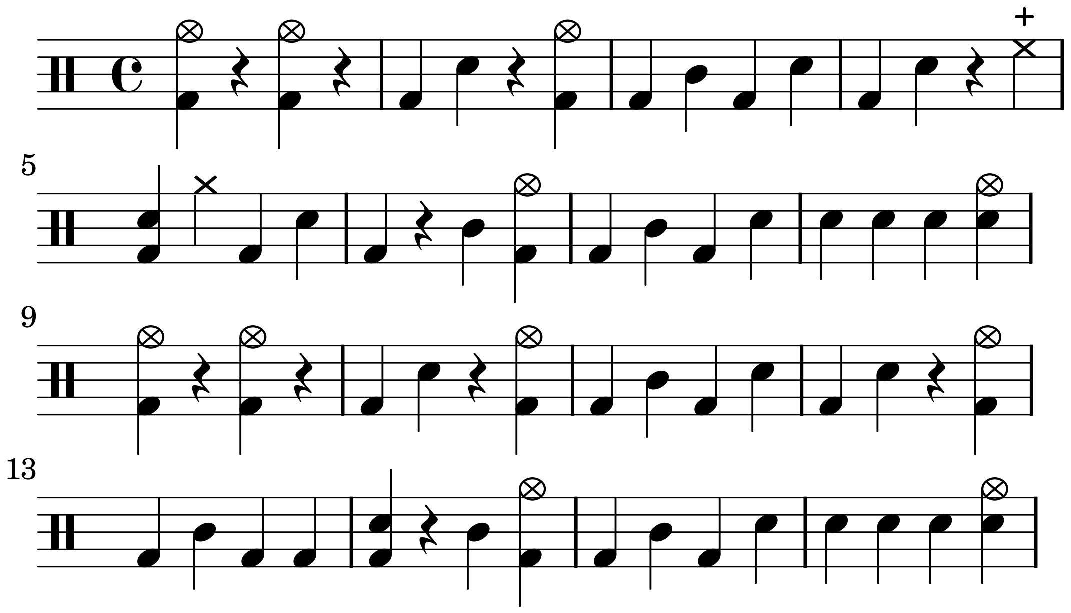
Figure A.30: Sheet music corresponding to Figure 5.1c

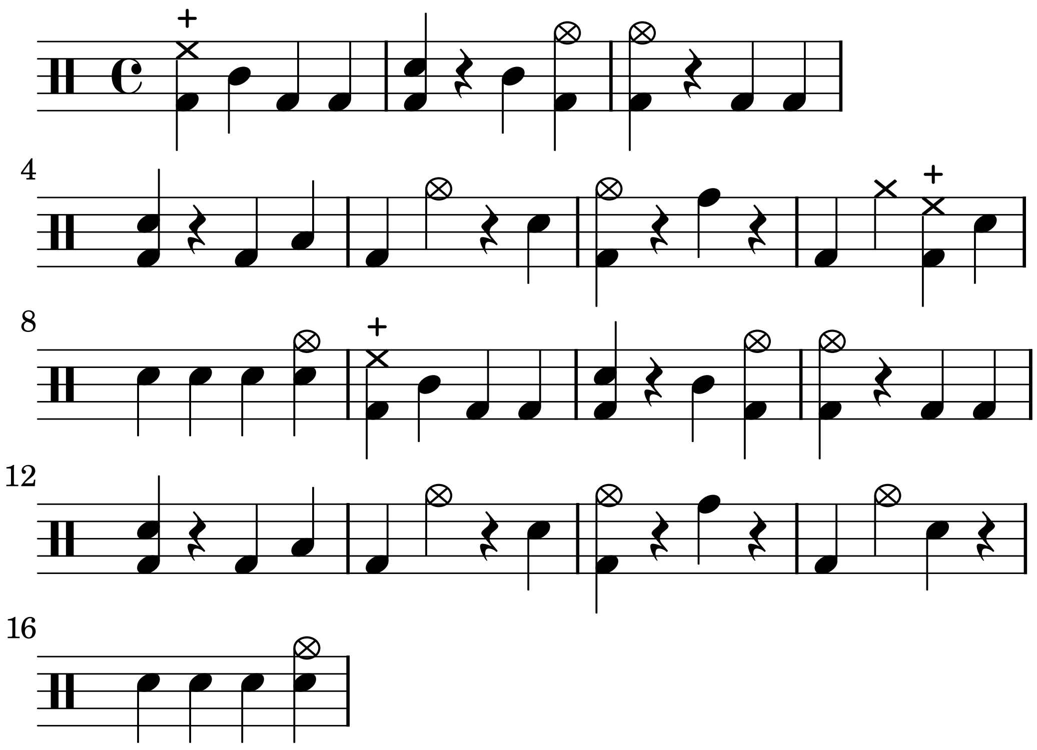
Figure A.31: Sheet music corresponding to Figure 5.1d

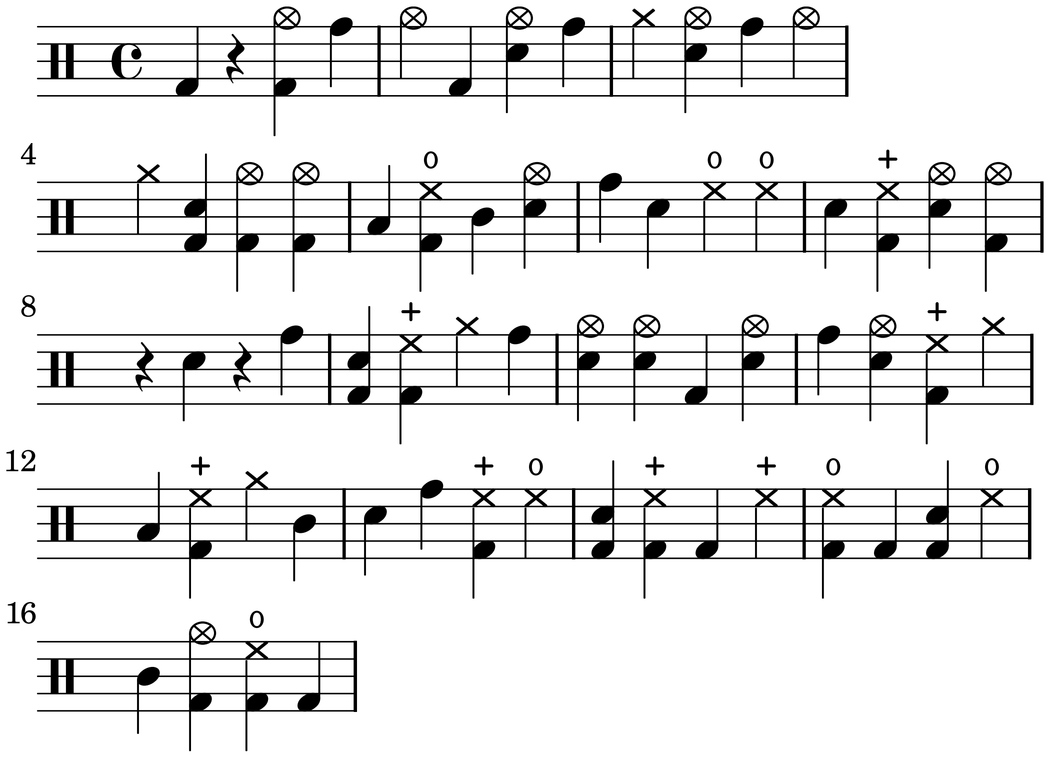

Figure A.32: Sheet music corresponding to Figure 5.2a

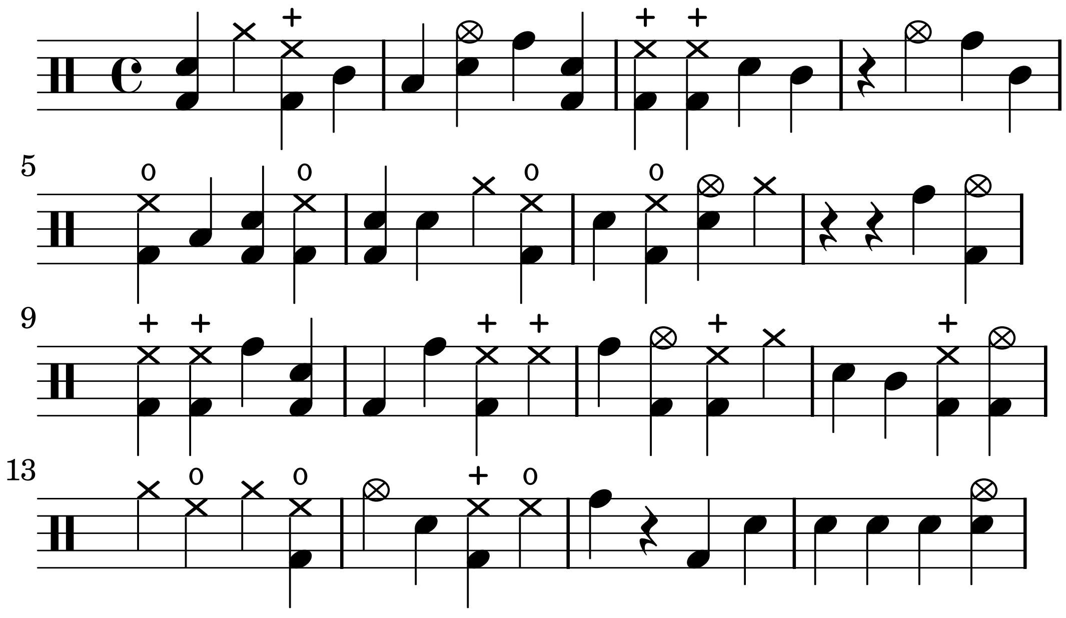

Figure A.33: Sheet music corresponding to Figure 5.2b

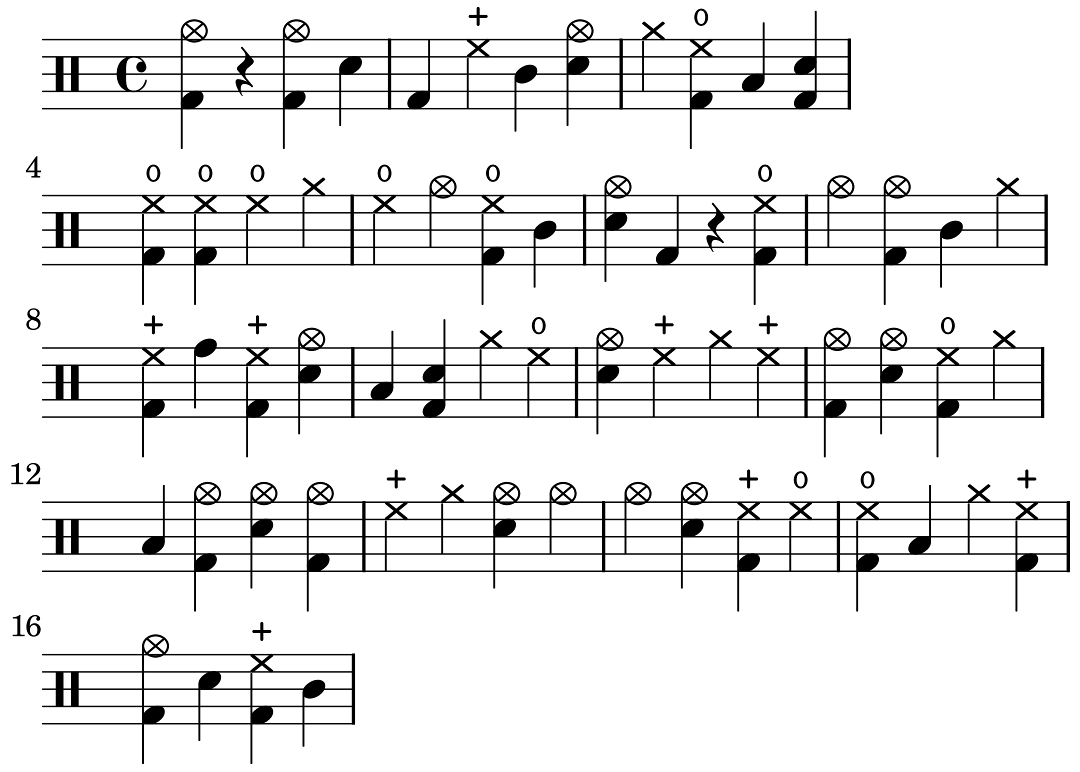

Figure A.34: Sheet music corresponding to Figure 5.2c

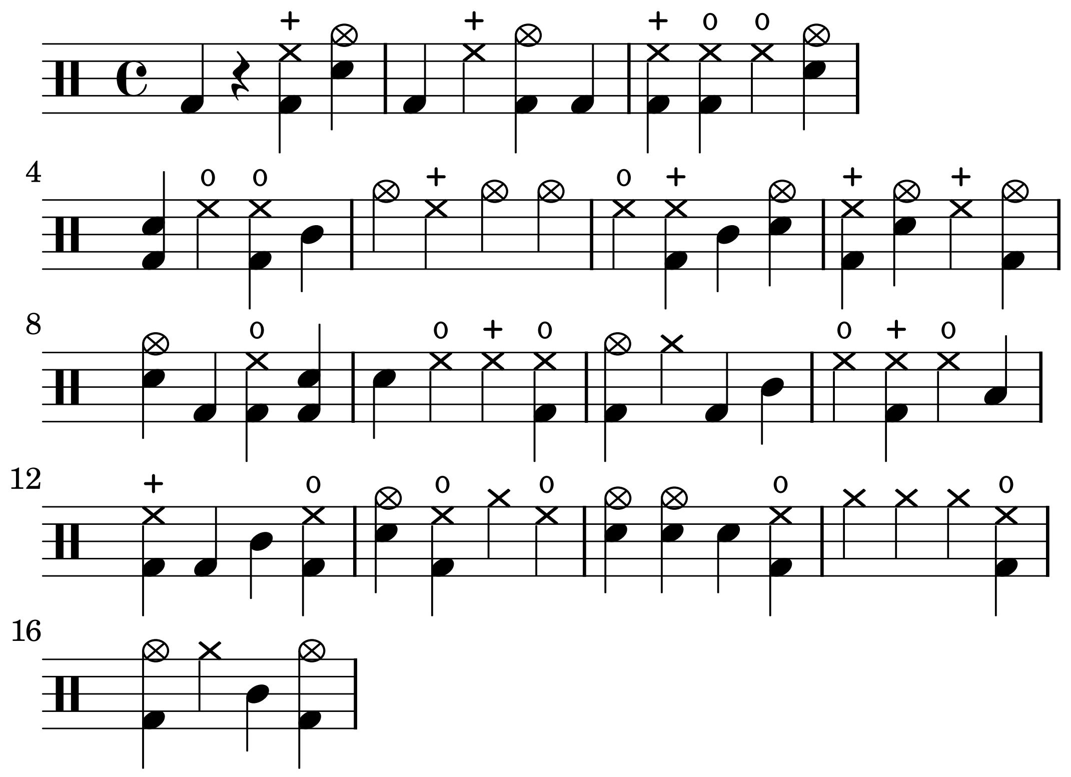

Figure A.35: Sheet music corresponding to Figure 5.2d

A.3.2 Experiment 3

This section presents the sheet music corresponding to the results of RA-1, experiment 3 (presented in 5.3.4).

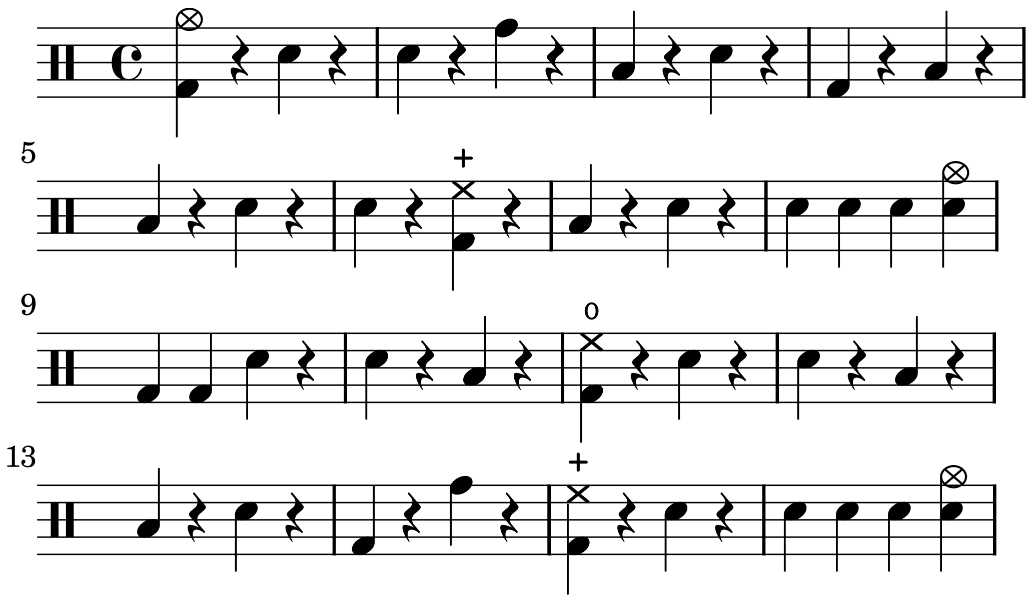

Figure A.36: Sheet music corresponding to Figure 5.3a

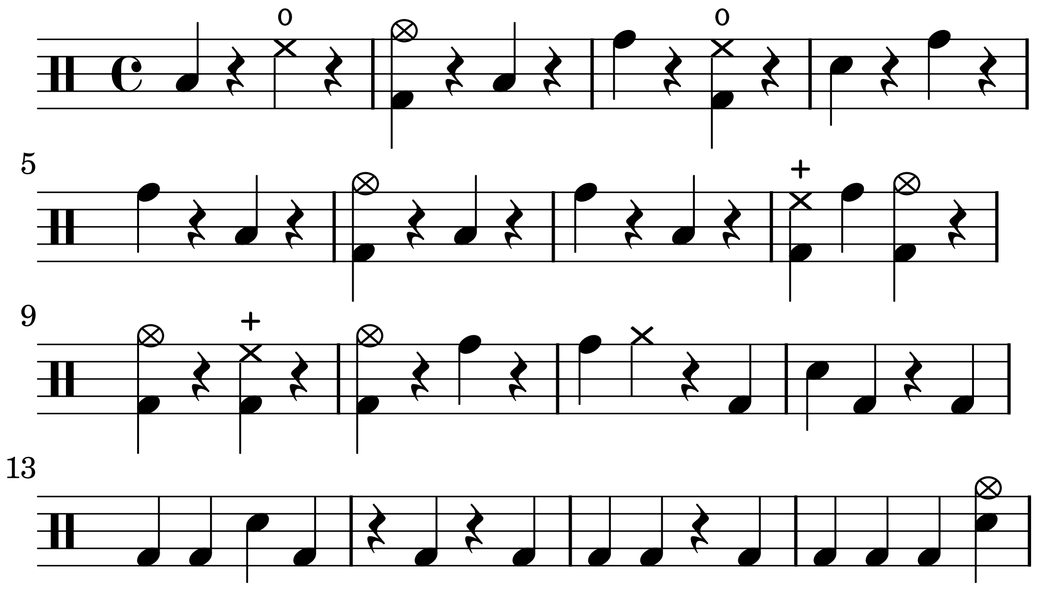

Figure A.37: Sheet music corresponding to Figure 5.3b

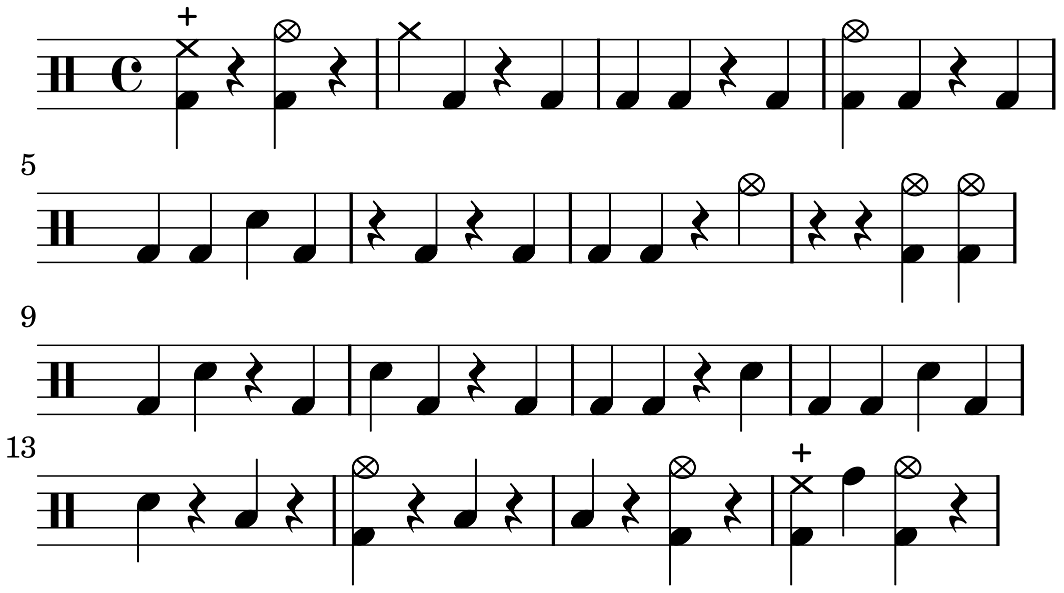

Figure A.38: Sheet music corresponding to Figure 5.3c

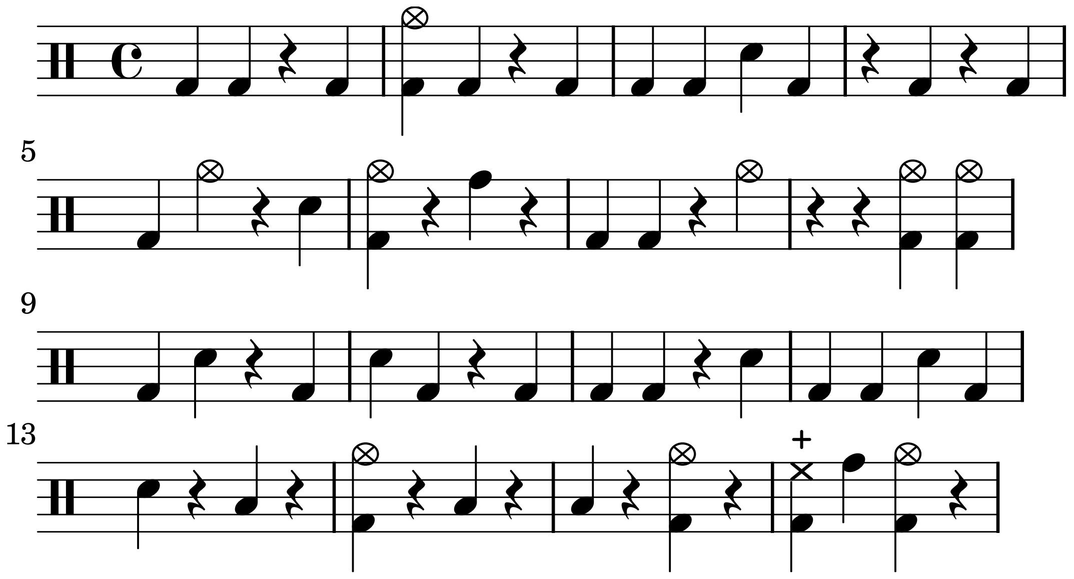

Figure A.39: Sheet music corresponding to Figure 5.3d

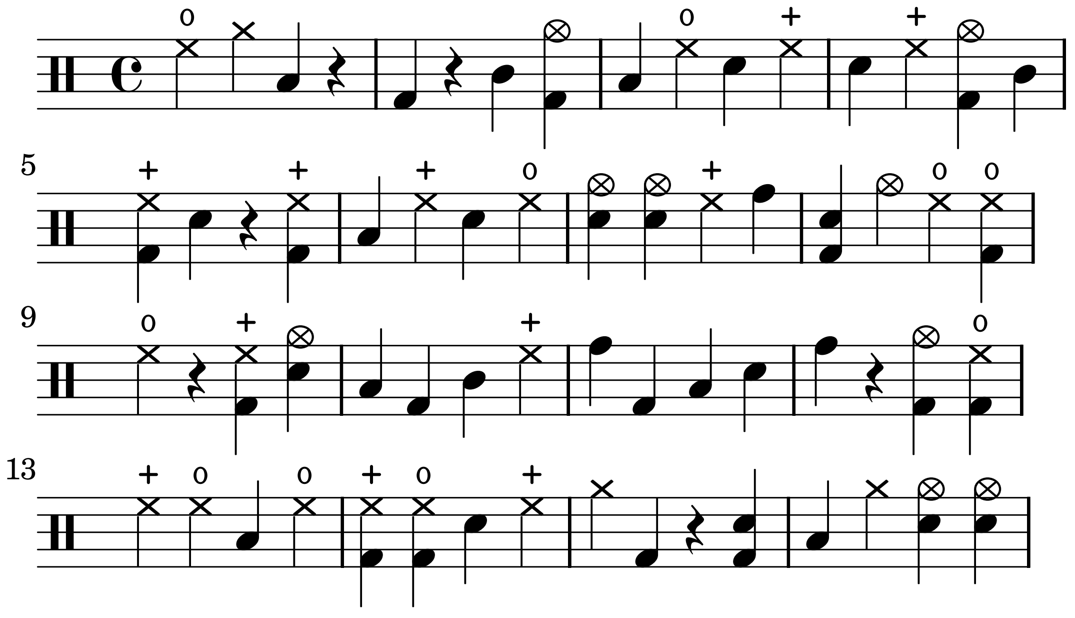

Figure A.40: Sheet music corresponding to Figure 5.4a

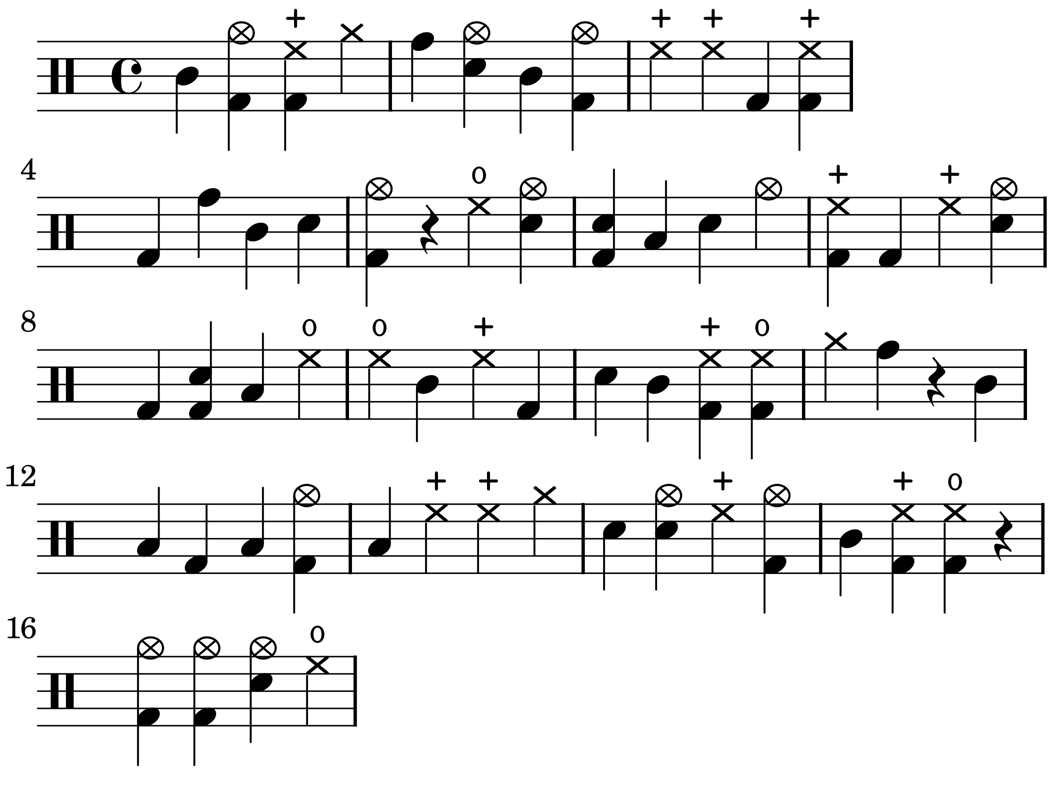

Figure A.41: Sheet music corresponding to Figure 5.4b

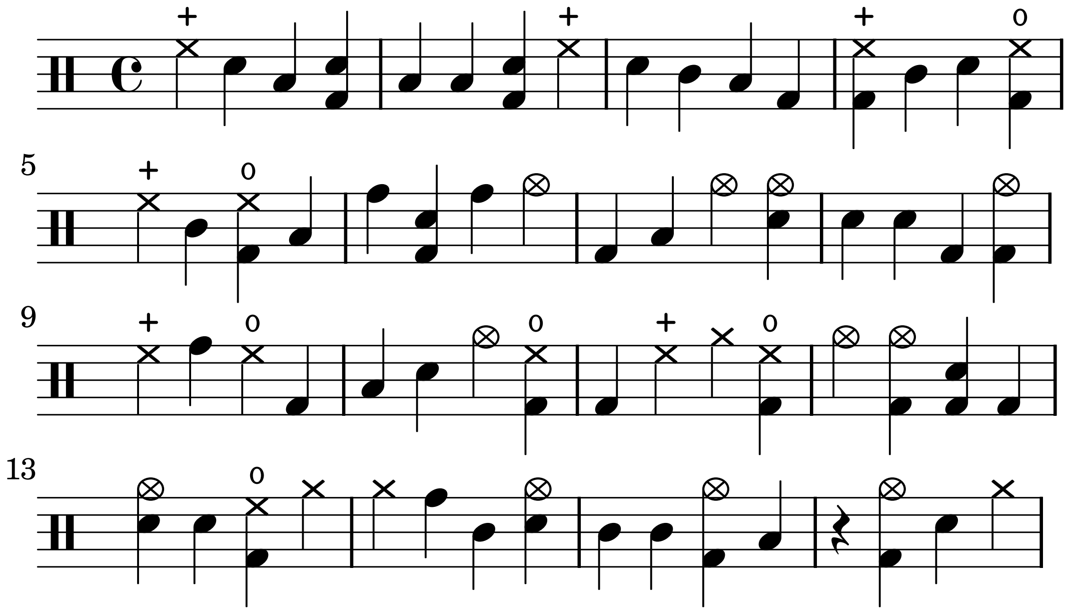

Figure A.42: Sheet music corresponding to Figure 5.4c

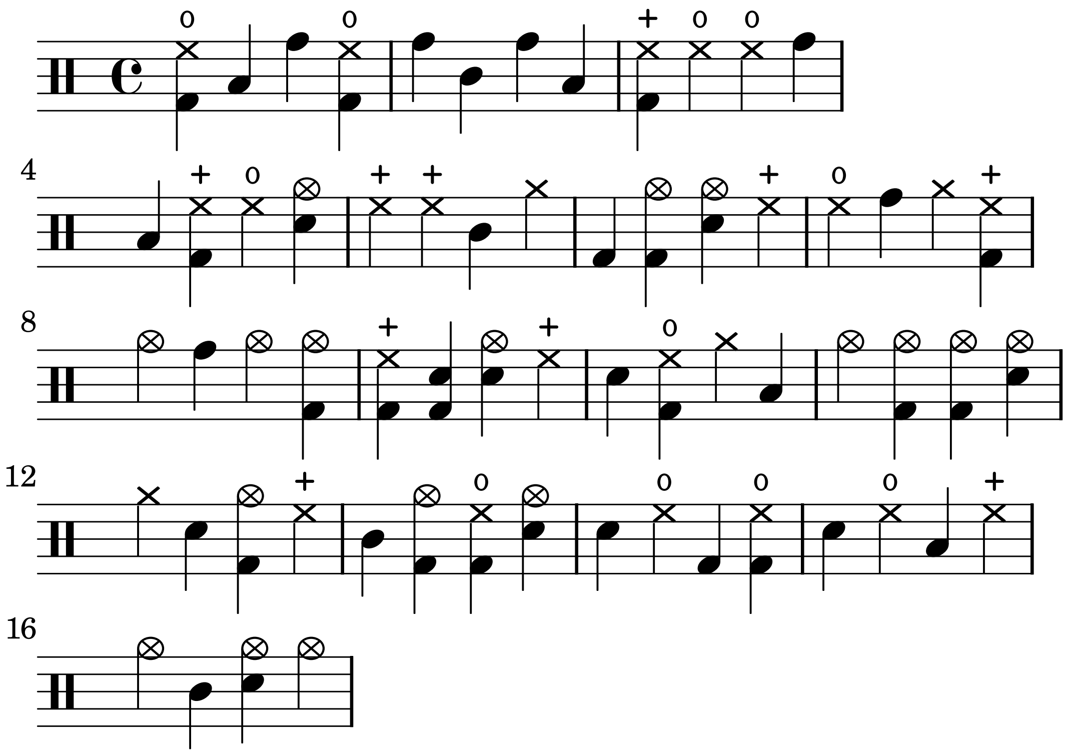

Figure A.43: Sheet music corresponding to Figure 5.4d

A.4 MA

A.4.1 Experiment 1

This section presents the sheet music corresponding to the results of MA, experiment 1 (presented in 6.3.2).

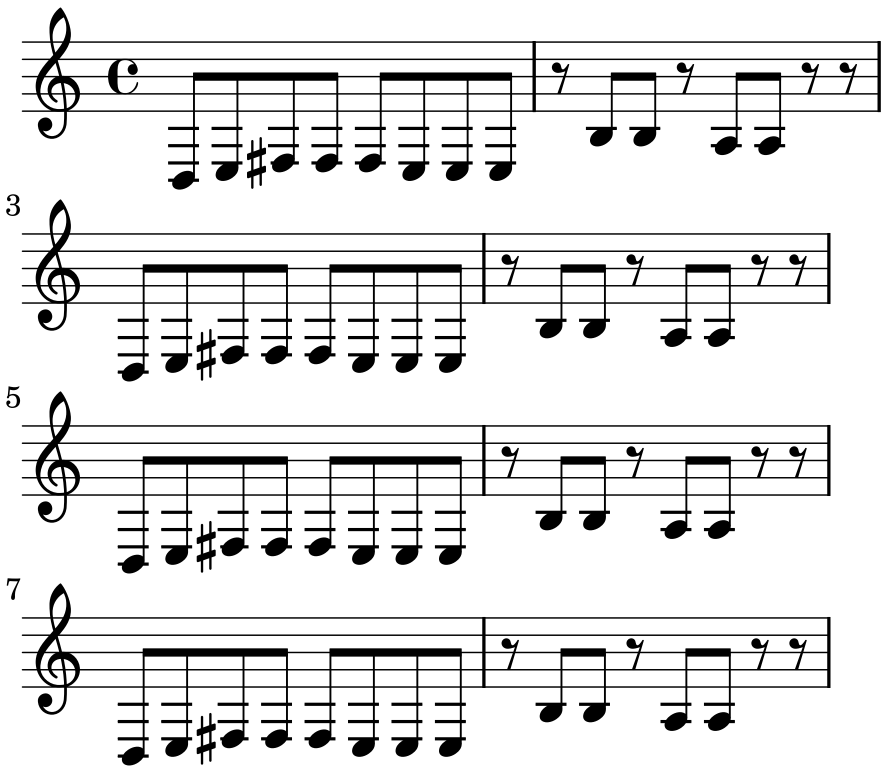

Figure A.44: Sheet music corresponding to Figure 6.5a

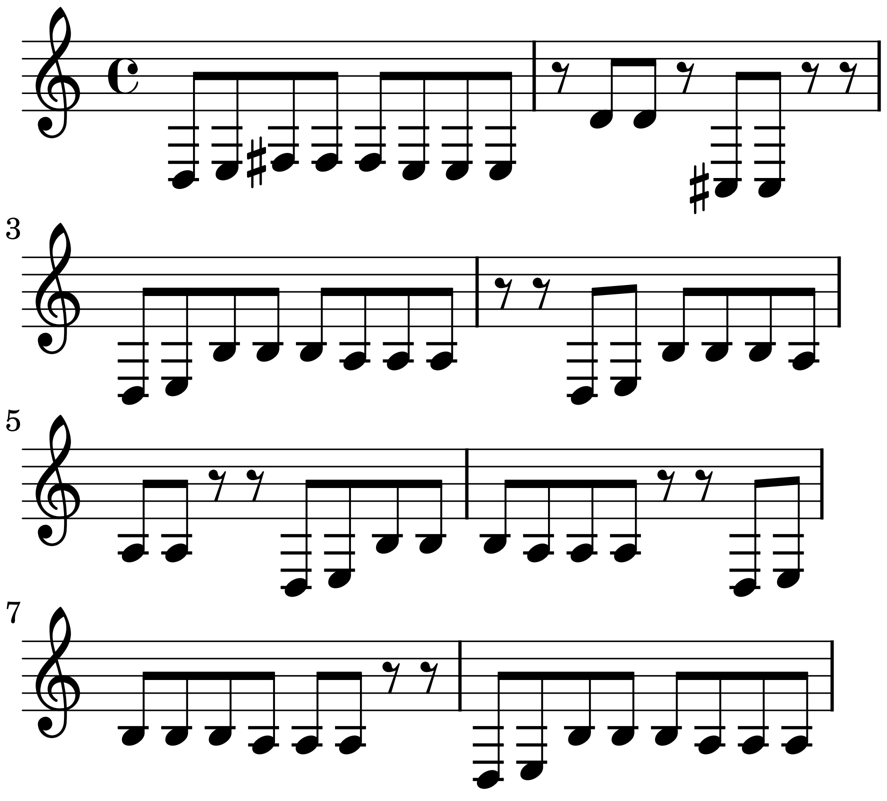

Figure A.45: Sheet music corresponding to Figure 6.5b

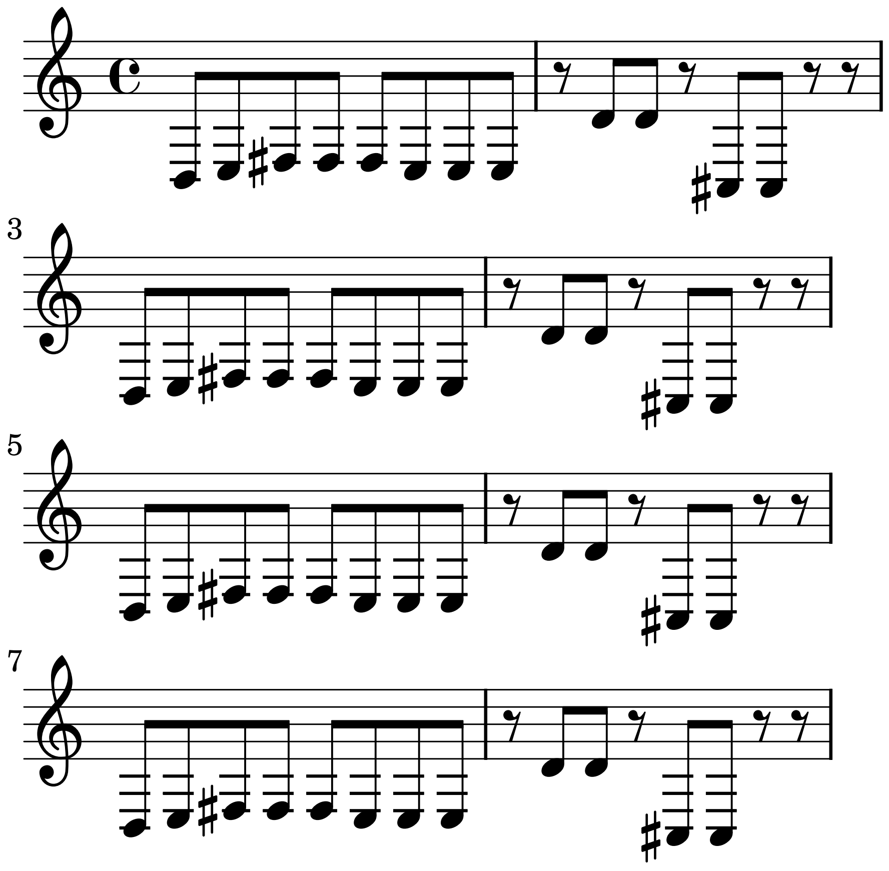

Figure A.46: Sheet music corresponding to Figure 6.5c

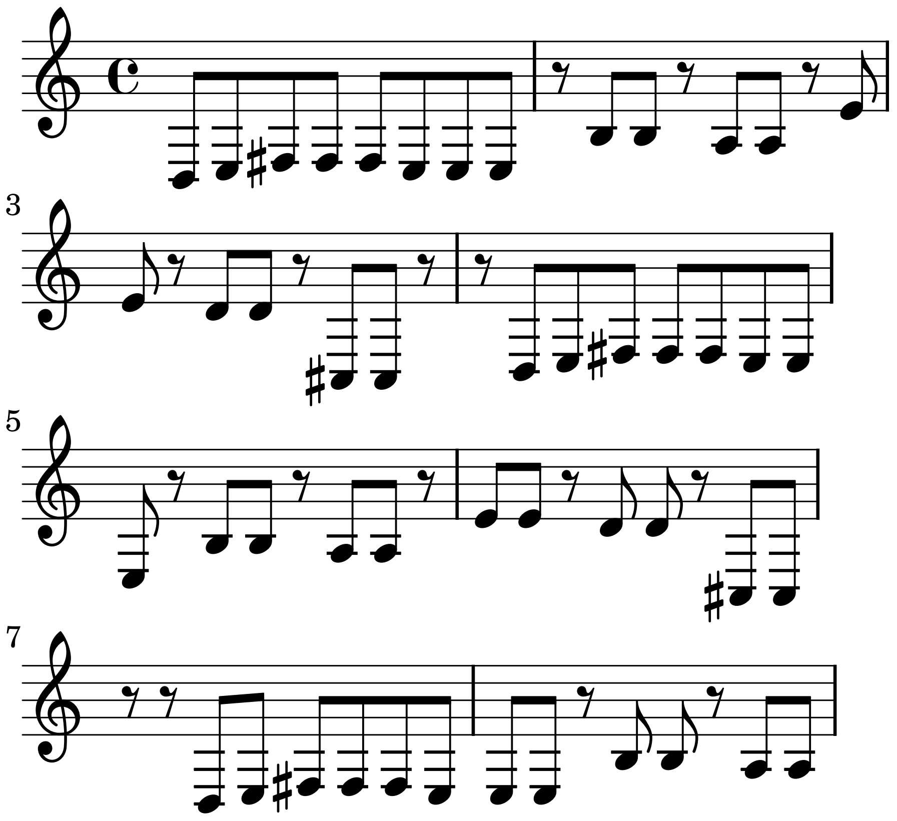

Figure A.47: Sheet music corresponding to Figure 6.5d

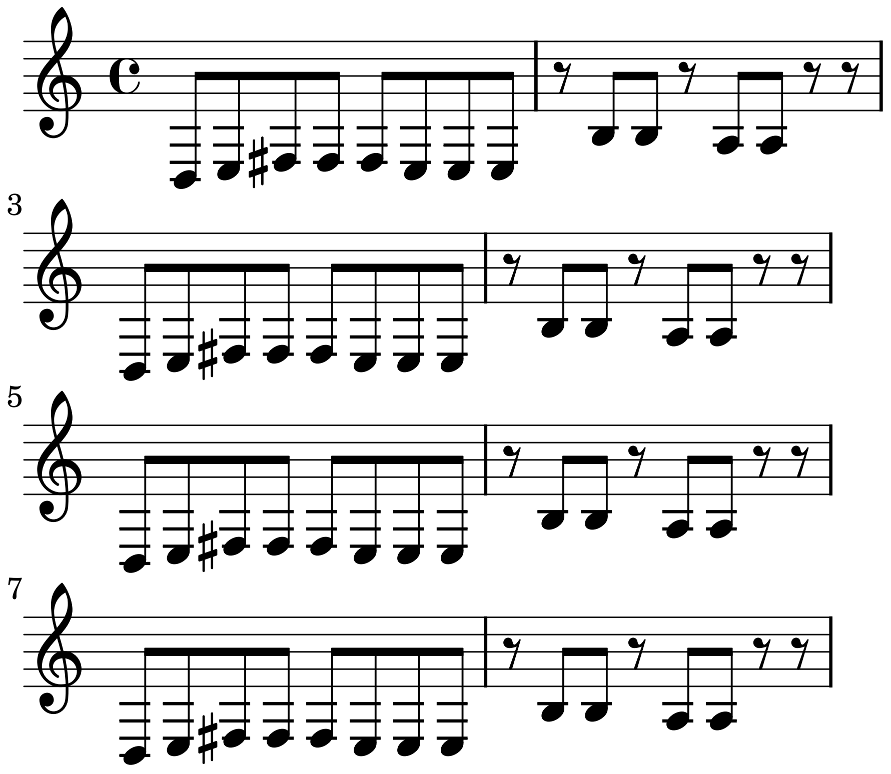

Figure A.48: Sheet music corresponding to Figure 6.6a

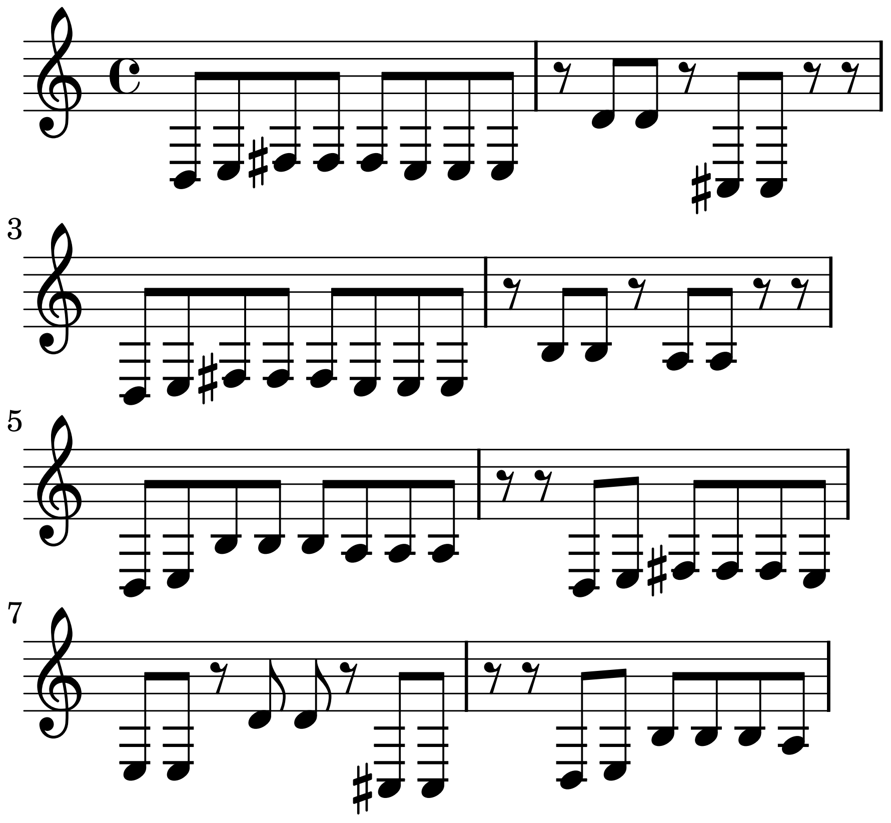

Figure A.49: Sheet music corresponding to Figure 6.6b

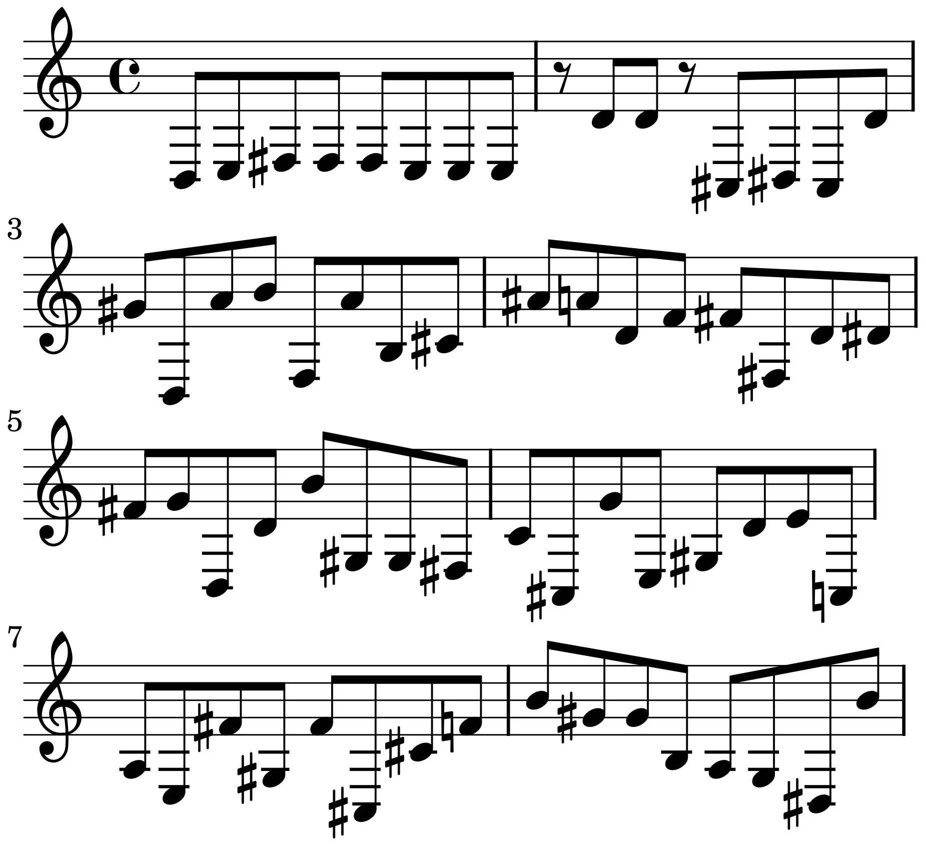

Figure A.50: Sheet music corresponding to Figure 6.6c

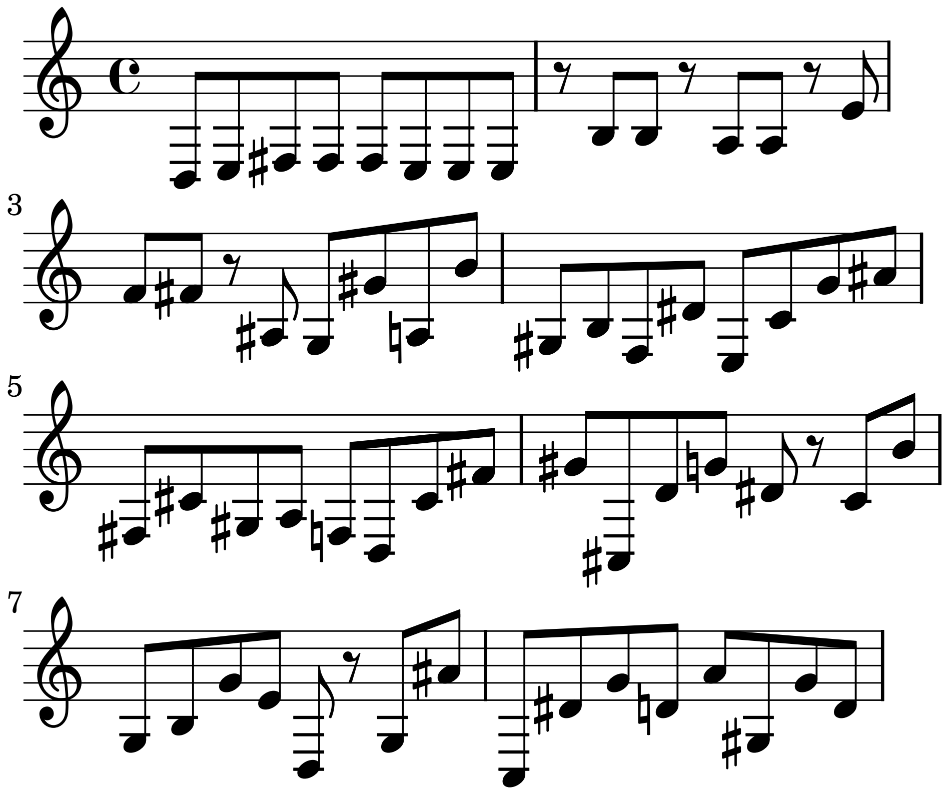

Figure A.51: Sheet music corresponding to Figure 6.6d

A.4.2 Experiment 2

This section presents the sheet music corresponding to the results of MA, experiment 2 (presented in 6.3.3).

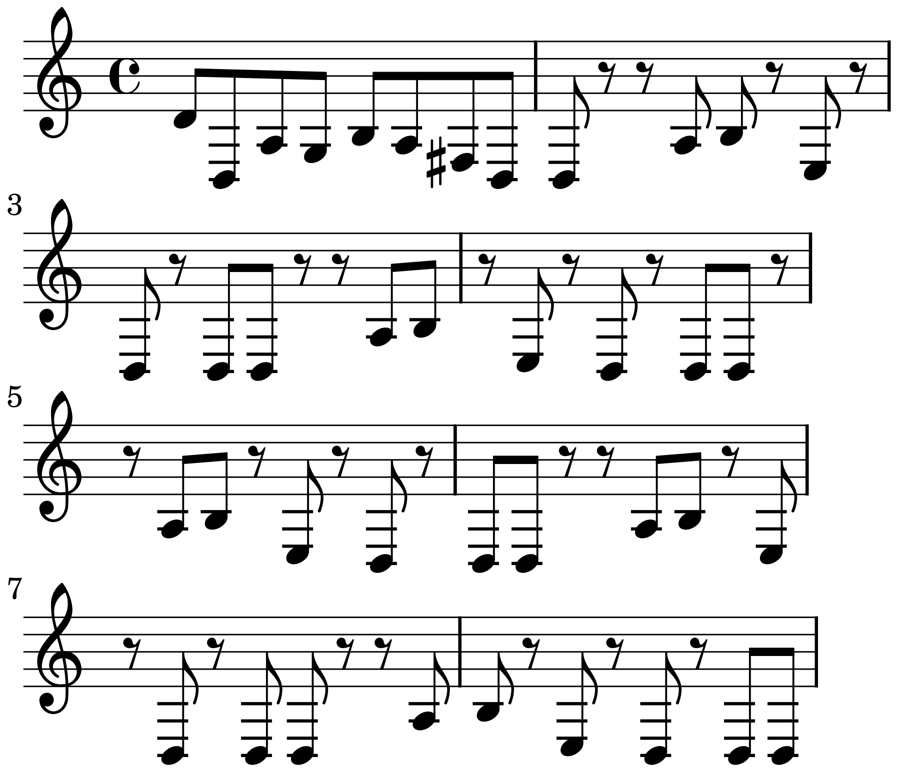

Figure A.52: Sheet music corresponding to Figure 6.7a

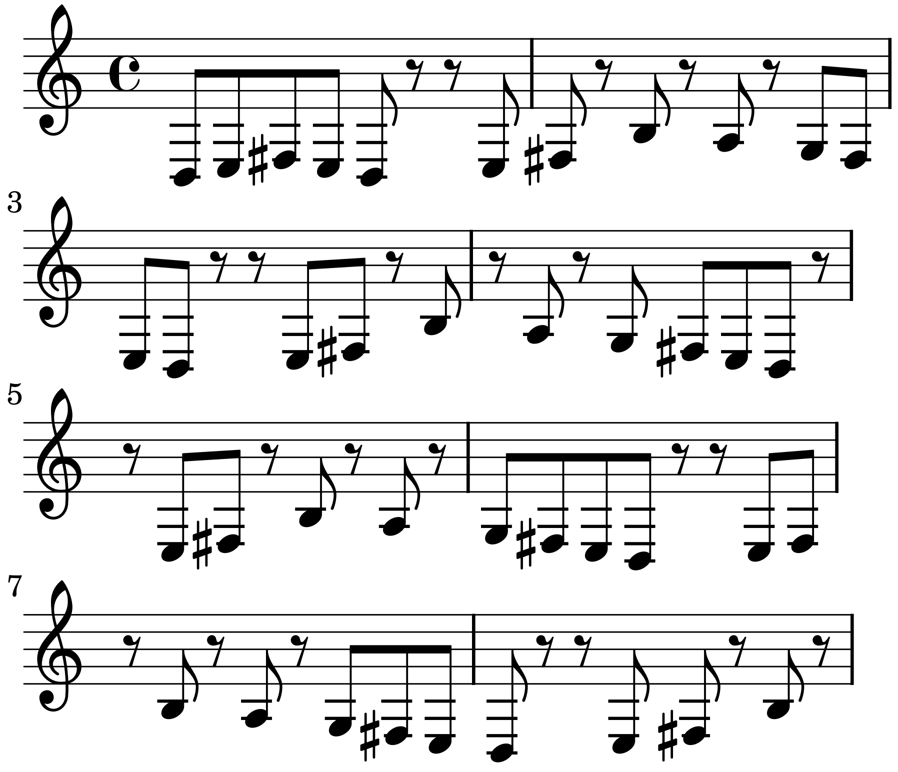

Figure A.53: Sheet music corresponding to Figure 6.7b

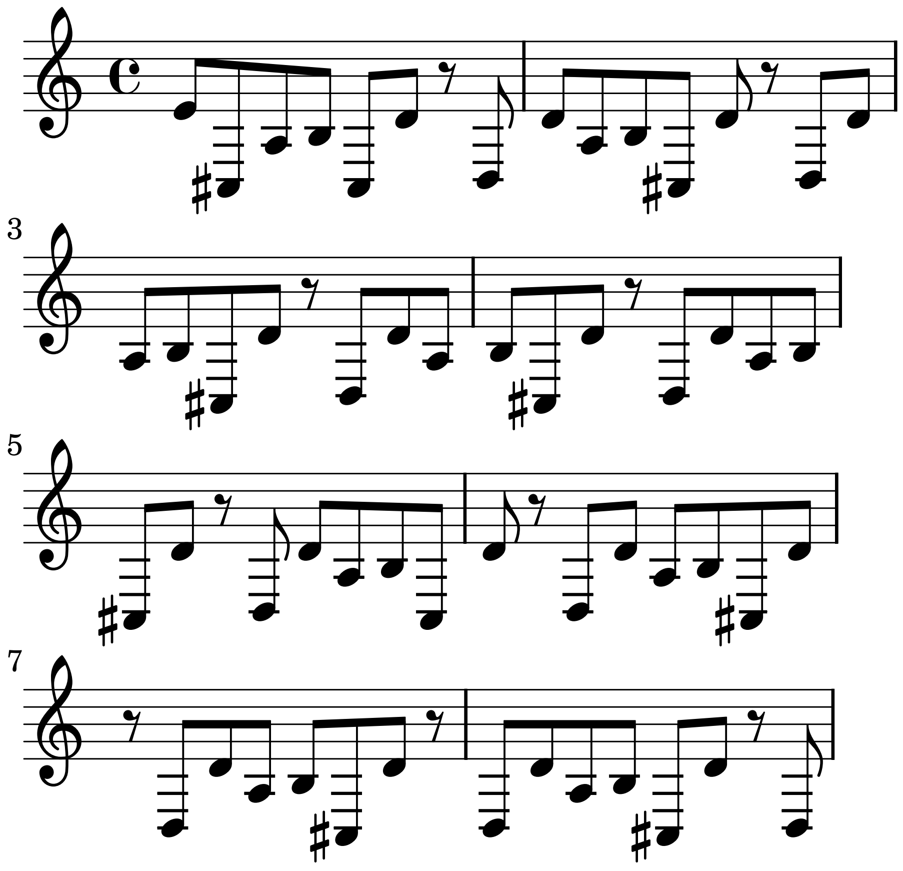

Figure A.54: Sheet music corresponding to Figure 6.7c

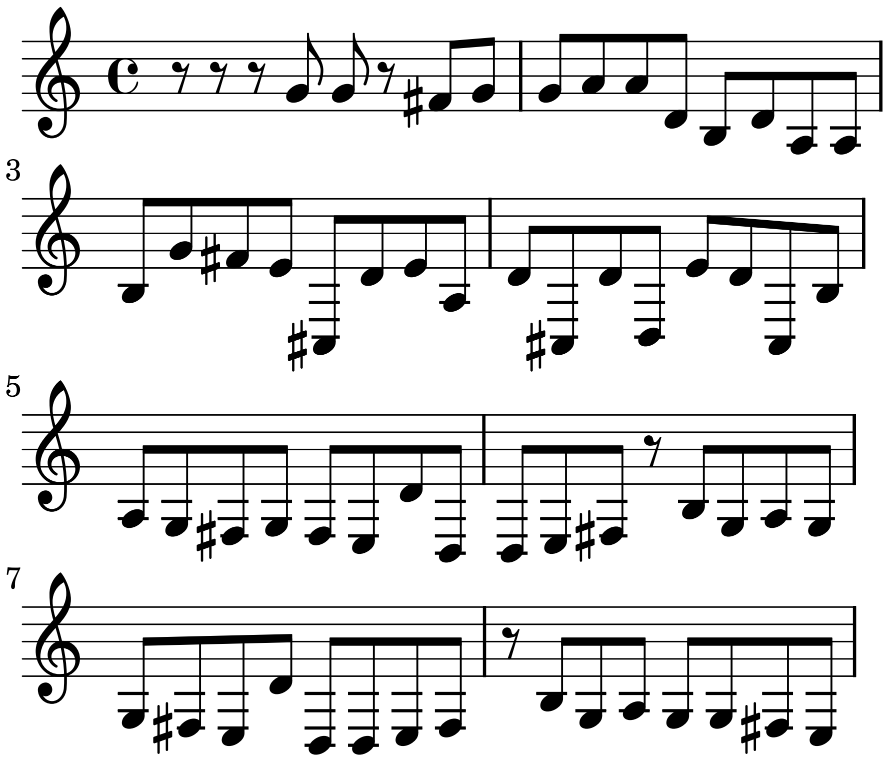

Figure A.55: Sheet music corresponding to Figure 6.7d

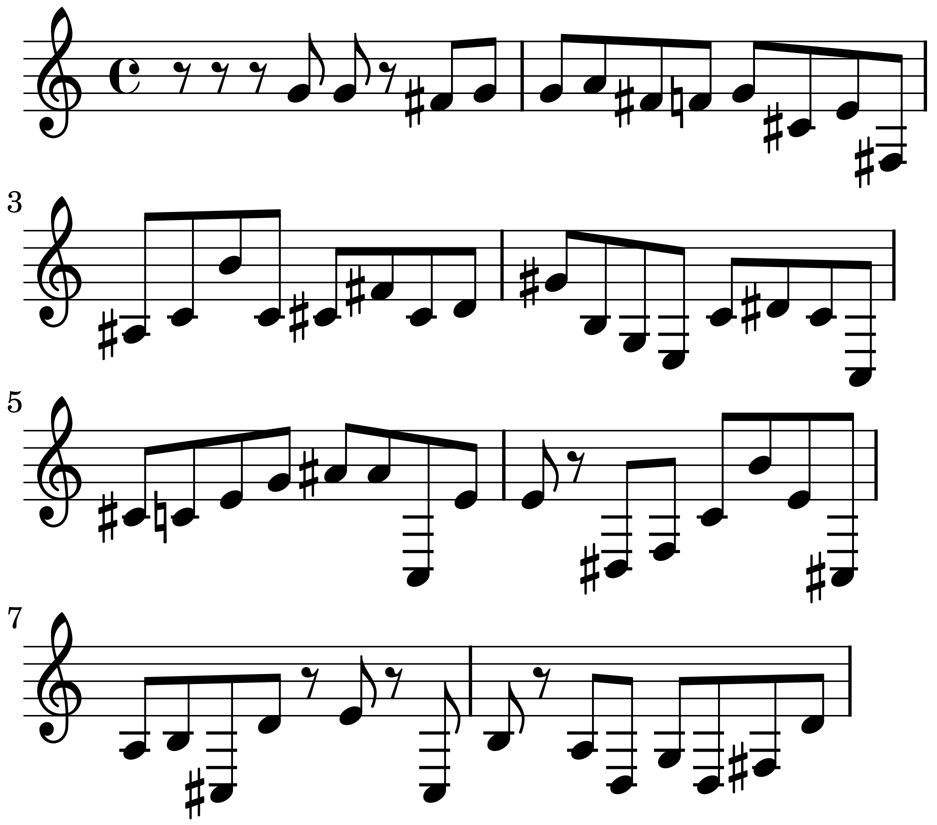

Figure A.56: Sheet music corresponding to Figure 6.8a

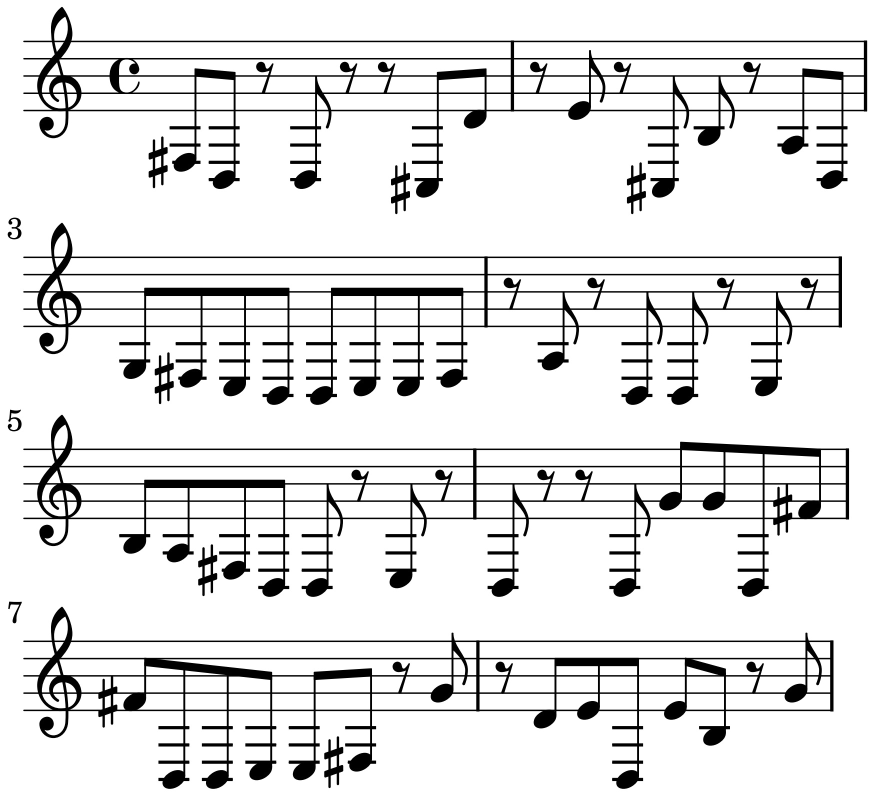

Figure A.57: Sheet music corresponding to Figure 6.8b

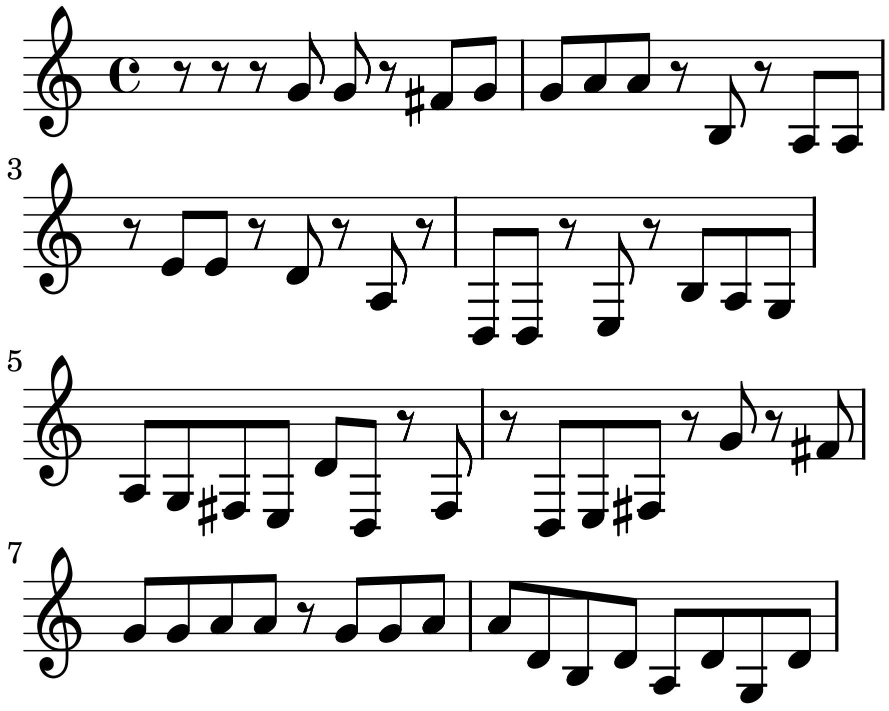

Figure A.58: Sheet music corresponding to Figure 6.8c

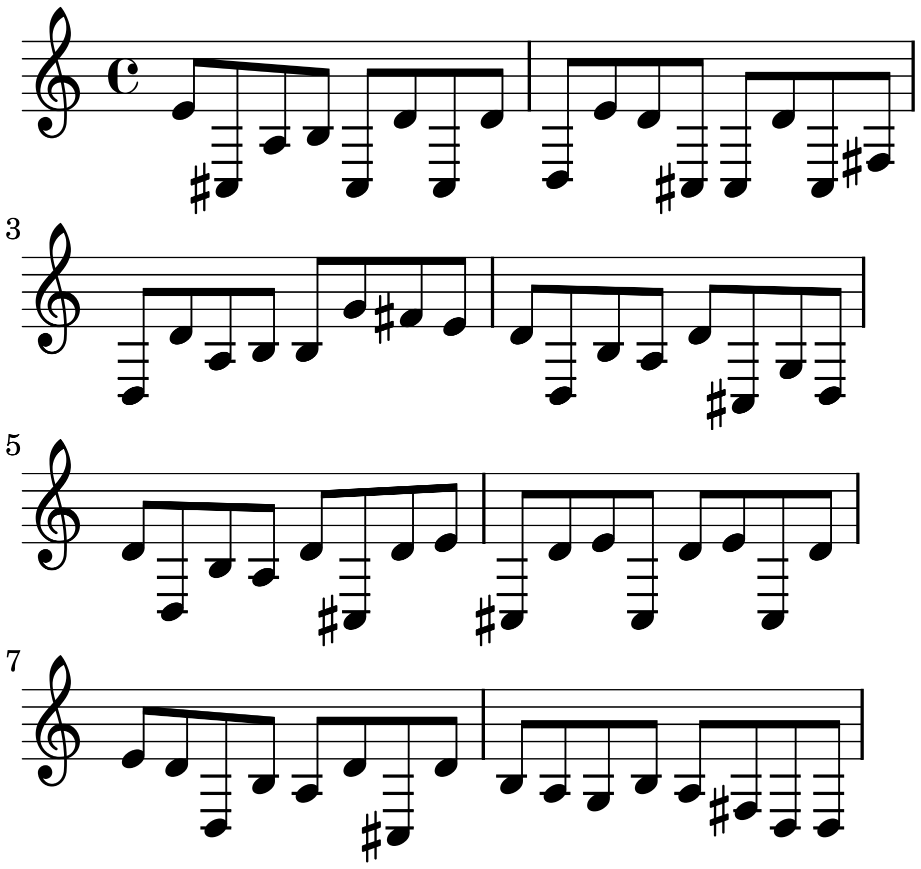

Figure A.59: Sheet music corresponding to Figure 6.8d

A.4.3 Experiment 3

This section presents the sheet music corresponding to the results of MA, experiment 3 (presented in 6.3.4).

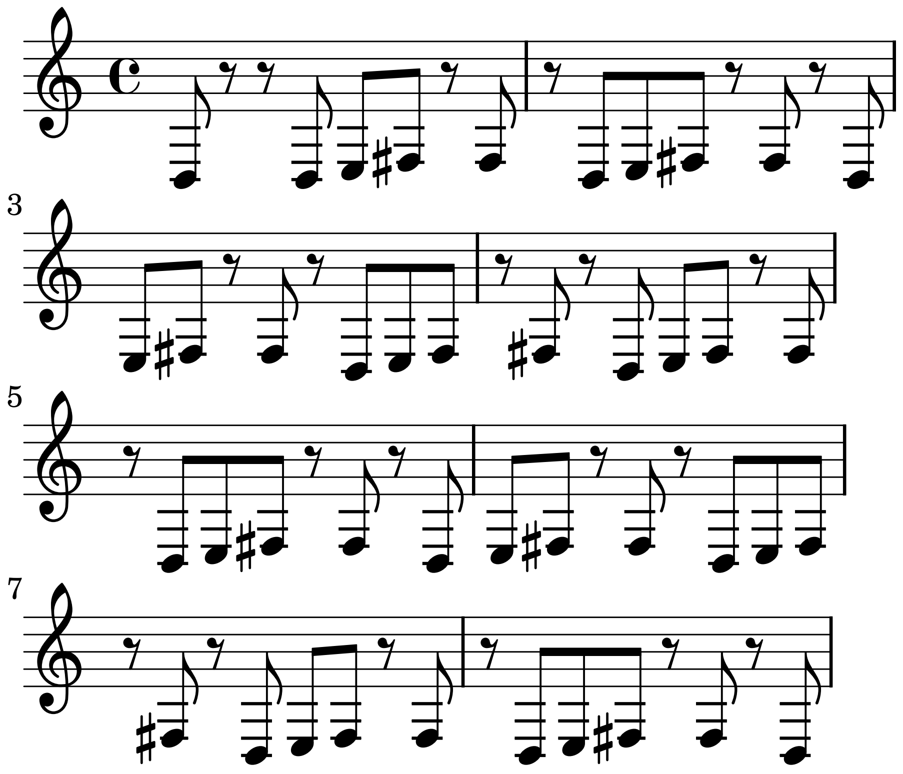

Figure A.60: Sheet music corresponding to Figure 6.9a

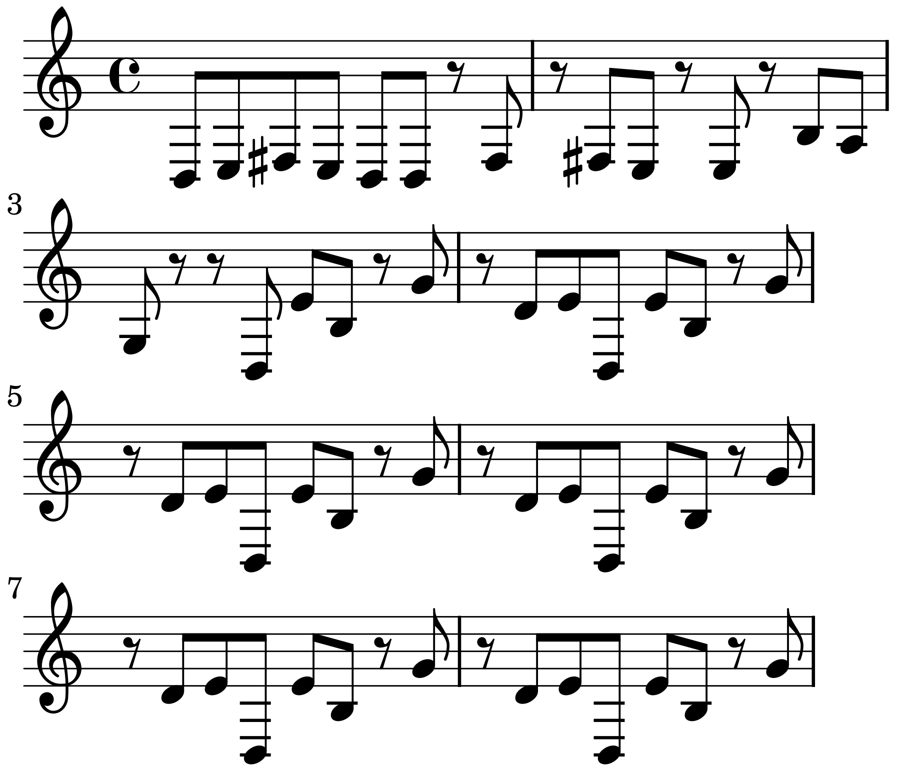

Figure A.61: Sheet music corresponding to Figure 6.9b

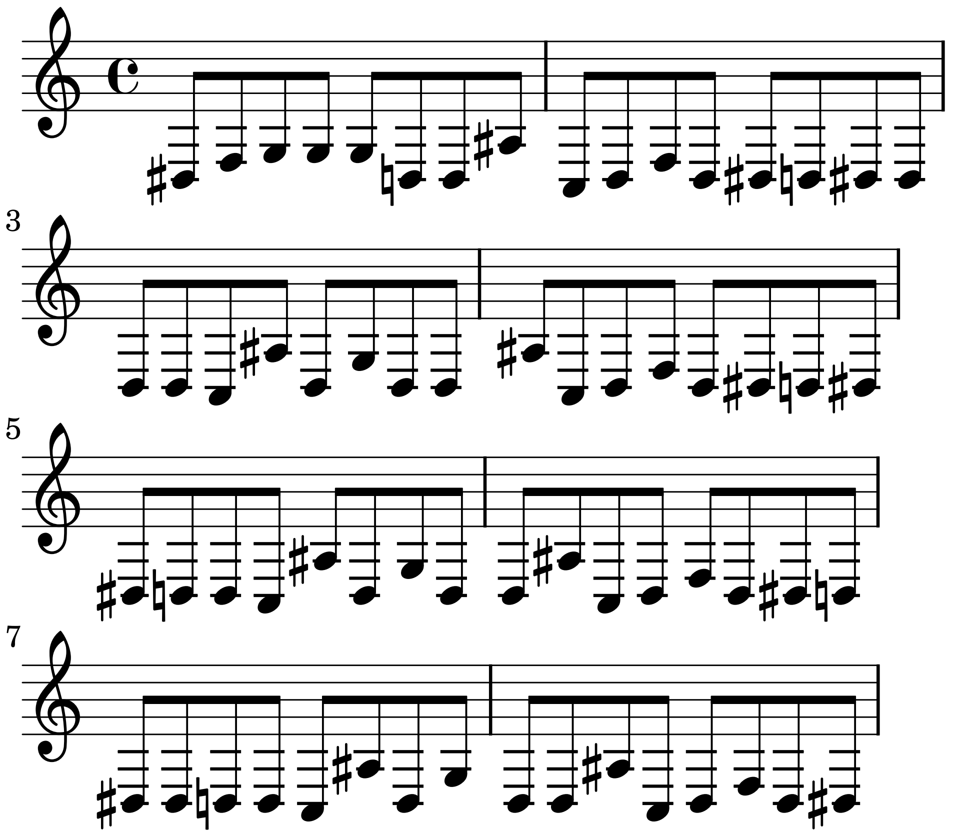

Figure A.62: Sheet music corresponding to Figure 6.9c

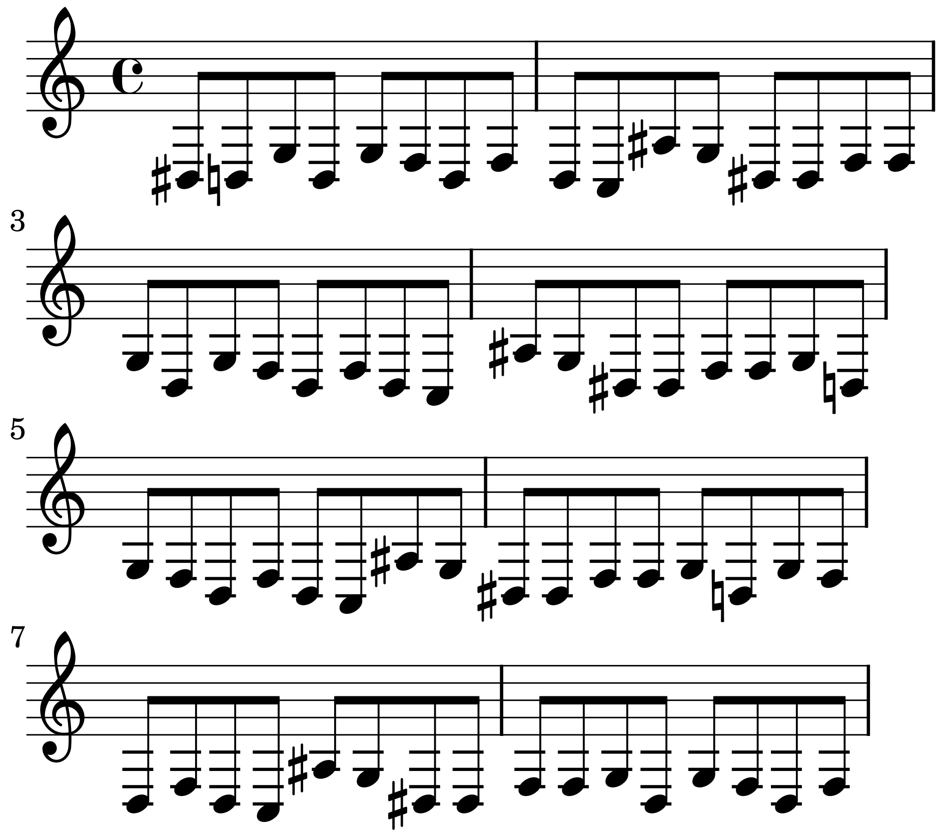

Figure A.63: Sheet music corresponding to Figure 6.9d

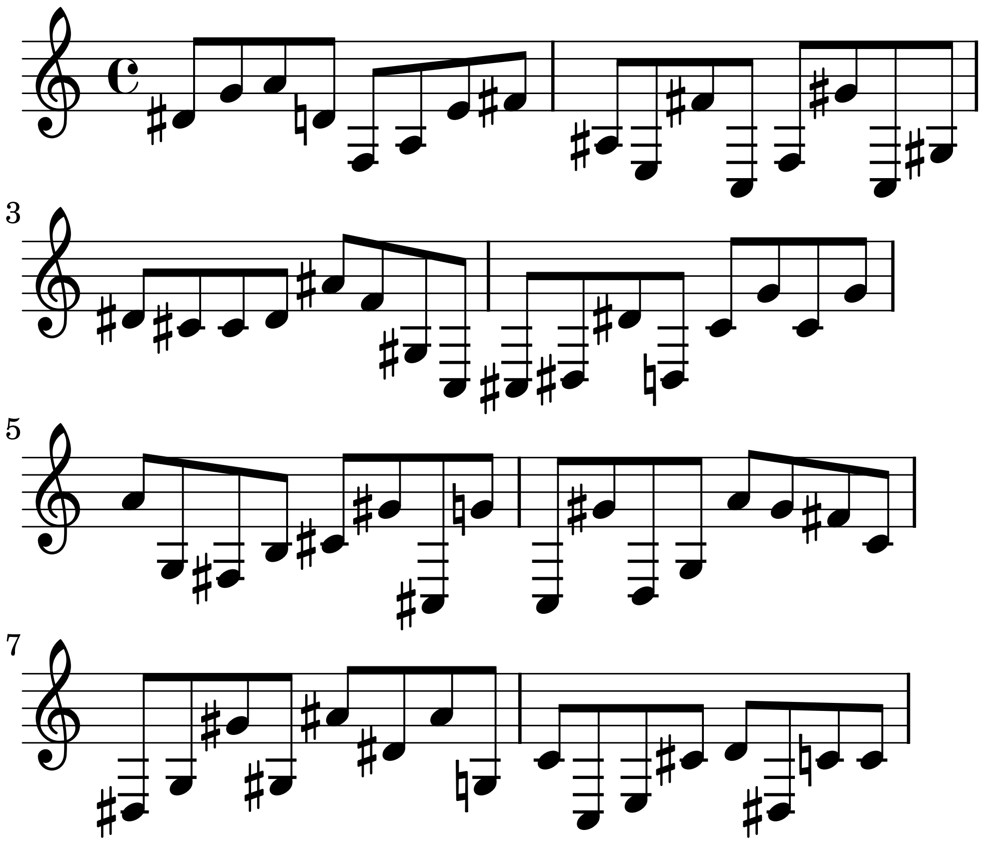

Figure A.64: Sheet music corresponding to Figure 6.10a

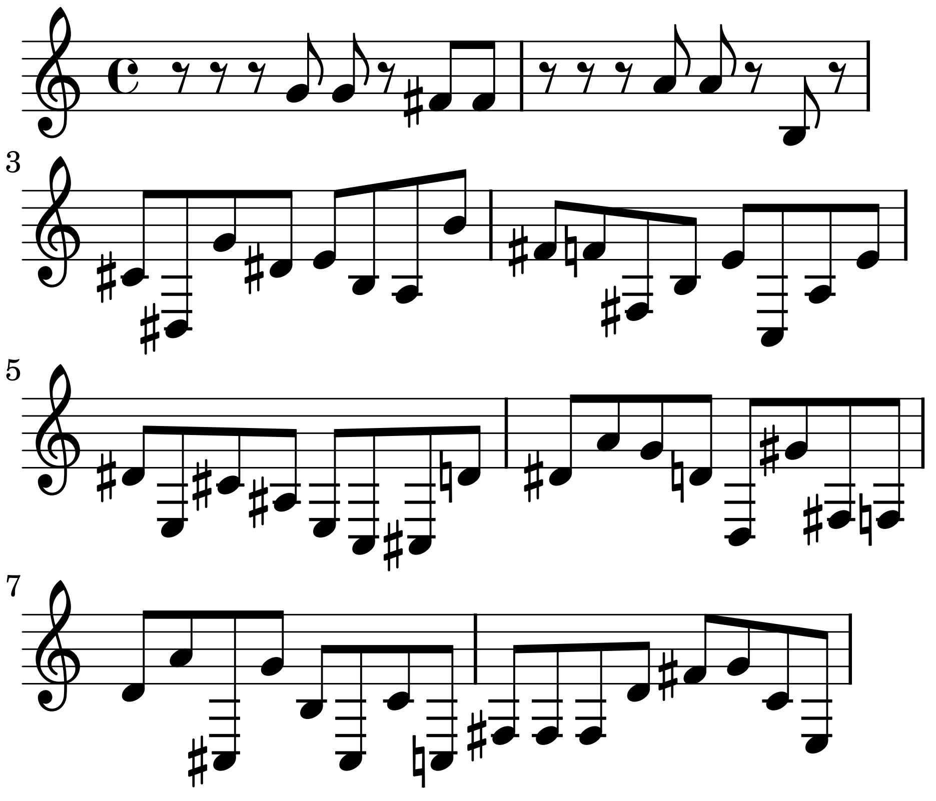

Figure A.65: Sheet music corresponding to Figure 6.10b

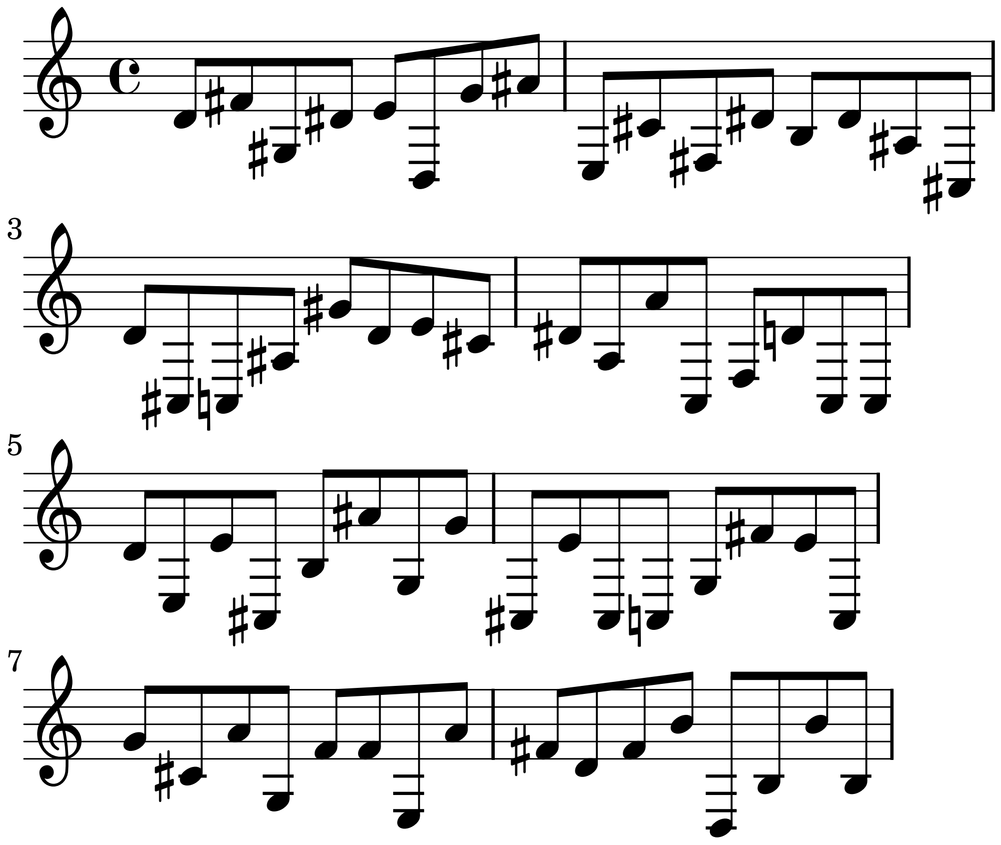

Figure A.66: Sheet music corresponding to Figure 6.10c

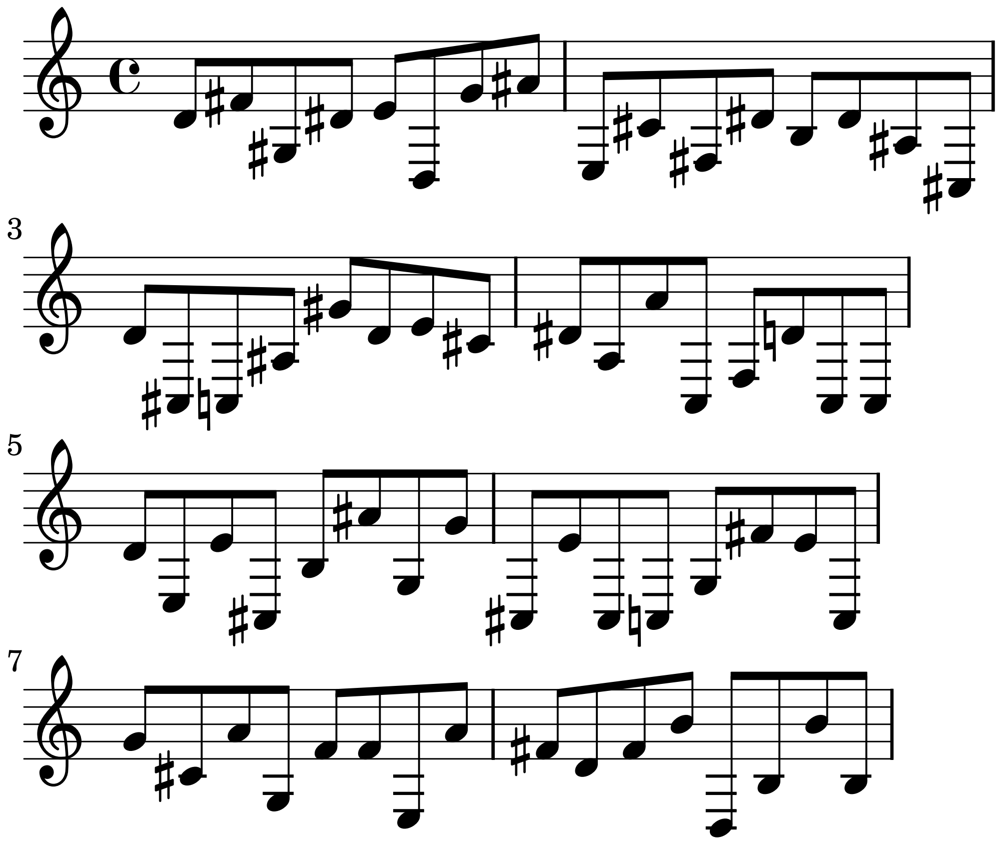

Figure A.67: Sheet music corresponding to Figure 6.10d

References

[Abbeel and Ng 2004] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[Acevedo *et al.* 2007] J Acevedo, S Gliske, and M Jayap. "musical style replication using apprenticeship learning", 2007.

[Ames 1989] C Ames. The markov process as a compositional model: A survey and tutorial. *Leonardo*, 1989.

[Ammer 1995] C Ammer. *The Harpercollins Dictionary of Music (Third Edition).* HarperPerennial, 1995.

[Assayag *et al.* 2006] Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont, and Shlomo Dubnov. Omax brothers: a dynamic topology of agents for improvization learning. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, AMCMM '06, pages 125–132, New York, NY, USA, 2006. ACM.

[Baroni and Jacoboni 1978] M. Baroni and C. Jacoboni. *Proposal for a Grammar of Melody: The Bach Chorales*. Monographies de sémiologie et d'analyses musicales. Independent Publishing Group, 1978.

[Boyden 1971] D Boyden. *An Introduction to Music*. Faber and Faber, 1971.

[Collins 2008] N. Collins. Reinforcement learning for live musical agents. In *Proceedings of ICMC2008, International Computer Music Conference, Belfast*. Citeseer, 2008.

[Conklin 2003] D Conklin. Music generation from statistical models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, 2003.

[Cont *et al.* 2007] A Cont, Dubnov S, and G Assayag. Anticipatory model of musical style imitation using collaborative and competitive reinforcement learning. In Butz M.V., Sigaud O., Pezzulo G., and Baldassarre G., editors, *Anticipatory Behavior in Adaptive Learning Systems*, volume 4520 of *Lecture Notes in Computer Science / Artificial Intelligence (LNAI)*, page 285306. Springer Verlag, Berlin, 2007.

[Diaz-Jerez 2000] G Diaz-Jerez. *Using Mathematical Models in Music Composition*. PhD thesis, The Manhattan School of Music, August 2000.

[Franklin and Manfredi 2002] Judy A. Franklin and Victoria U. Manfredi. Computational intelligence and applications. chapter Nonlinear Credit Assignment for Musical Sequences, pages 245–250. Dynamic Publishers, Inc., Atlanta, GA, USA, 2002.

[Horner and Goldberg 1991] Andrew Horner and David E. Goldberg. Genetic algorithms and computer-assisted music composition. In Richard K. Belew and Lashon B. Booker, editors, *ICGA*, pages 437–441. Morgan Kaufmann, 1991.

[hyd] Hydrogen - advanced drum machine for gnu/linux. http://www.hydrogen-music.org/hcms/. Accessed: 2014-02-27.

[Jones 1981] K Jones. Compositional applications of stochastic processes. *Computer Music Journal*, 1981.

[Little 1993] David Little. Composing with chaos; applications of a new science for music. *Interface*, 22(1):23–51, 1993.

[Loy 1985] G Loy. Musicians make a standard: The midi phenomenon. *Computer Music Journal*, 1985.

[Miranda 1993] E.R. Miranda. *Cellular Automata Music: An Interdisciplinary Project*. Swets & Zeitlinger, 1993.

[Nelson 1996] G.L. Nelson. Real time transformation of musical material with fractal algorithms. *Computers & Mathematics with Applications*, 32(1):109 – 116, 1996.

[Ng and Russell 2000] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Papadopoulos and Wiggins 1999] G Papadopoulos and G Wiggins. Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *IN AISB SYMPOSIUM ON MUSICAL CREATIVITY*, pages 110–117, 1999.

[Sutton and Barto 1998] R Sutton and A Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.