электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.9

Системы генерации музыки или как автоматизировать искусство?

Фазылова Э.Ф., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю. Е., доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gapyu@bmstu.ru

Введение

«Одна машина может сделать работу пяти обычных людей, но ни одна машина не сделает работу одного незаурядного человека.». Однако, незаурядный человек может создать машину, выполняющую незаурядные функции. «Как научить машину сочинять уникальные мелодии?» - этим вопросом давно задаются многие программисты, музыканты и просто экспериментаторы.

1. История математизированной композиции.

С тех пор как музыку стали записывать на бумаге в виде нотных знаков, стали появляться оригинальные «способы» ее сочинения. Стоит упомянуть о нескольких интересных методах. Изучением первого занимался Моцарт; созданная юным Моцартом, «Музыкальная игра в кости». Эта игра позволяет любому составить композицию в ритме вальса, бросая кости. Моцарт выписал 11*16=176 коротких тактов, и сделал доску, разметив ее делениями, указав, какой такт должен играться в соответствии с результатом бросания костей. Первый такт в любом метании кости был в тонической тональности, второй – в доминанте и так далее, так что любой такт в тонической тональности мог быть легко прикреплен к любому такту в доминанте. Таким образом, существует большая вероятность создать произведение незаурядное и никем неизведанное. Первое компьютерное музыкальное произведение – «Illiac Suite for String Quartet» – датировано 1956 годом. Создали его двое пионеров применения компьютеров в музыке – Лежарен Хиллер и Леонард Айзексон (Leonard Isaacson). В этой сюите использованы почти все главные концепты математизированной композиции: теория вероятностей, марковские цепи и генеративная грамматика.

2. Методы сочинения алгоритмической музыки.

Для того, чтобы создать мелодию, нужны какие-нибудь правила, алгоритмы. «Настоящая наука и настоящая музыка требуют однородного мыслительного процесса.» (Альберт Эйнштейн). Существует множество методов сочинения, рассмотрим некоторые из них.

Композитор Оливье Мессиан, помимо традиционных диатонических ладов европейской музыки типа мажора или минора, с помощью численных методов создавал искусственные лады, а также ритмические последовательности**,** несвойственные традиционной европейской музыке. Для генерации ритмических последовательностей использовались ряды чисел, например, простых. Все разнообразие методов Мессиана можно свести к двумерным таблицам чисел – матрицам, кодирующим высоту и длительность нот. Музыкальный материал получается с помощью операций над этими таблицами, например, сложения и умножения матриц, умножения матриц на число.

Еще один способ алгоритмизации мелодий. Контекстно-зависимая грамматика (КЗ-грамматика, контекстная грамматика) — частный случай формальной грамматики, у которой левые и правые части всех продукций могут быть окружены терминальными и нетерминальными символами.

Возьмем обычную строку: ABCDEFGIKFHLEFJ. И начнем строить для нее грамматику, начав, скажем, с символа F (вообще это нужно проделать для каждого символа). Нам нужно написать правило, которое бы указывало нам, какую букву следует поставить, если мы вдруг встретили символ F. Записывается это так: F -> *что-то*. Как видим, мы не можем создать такое правило, так как лишь по одной букве не можем определить, что же должно идти следом: после F может идти как G, так и H или J. Поэтому мы добавляем контекст к нашей букве F, контекст — это символы, окружающие F. Возьмем по одной букве перед F. Получим EF и KF. Контекстом для буквы F здесь служат буквы E и K. Мы с вами только что расширили контекст на один символ, поэтому данный метод построения грамматики называется методом динамически расширяющегося контекста.

Посмотрим можем ли мы сейчас создать правило. После KF идет H, и больше нет других вариантов. Мы получили первое конечное правило: KF -> H (читается как «KF *продуцирует* H). Теперь, если, например, на конце строки мы встретим KF и нужно будет продлить строку на один символ, мы смело напишем H.

Но у нас еще осталась проблема: после EF может идти как G, так и J. Поэтому мы должны расширить контекст еще на один символ: DEF и LEF. И наконец-то мы получили конечные правила:

KF->H

DEF->G

LEF->J

Это правила для буквы F, в зависимости от ее контекста мы выбираем какое-то одно правило.

Процесс генерации новой строки выглядит следующим образом: дана начальная последовательность, например ADEF. Начинаем брать буквы с конца. F — нет правила с такой левой частью, расширяем контекст — EF, опять нет, расширяем — DEF, есть такое правило, ставим G, получаем ADEFG. Начинаем все сначала: берем букву G и т.д. столько раз, сколько нам нужно. Будет удобно представлять грамматику в виде дерева. Для буквы F дерево будет иметь следующий вид, представленный на рис.1

В узлах находятся левые части правил, рядом с узлами написаны правые части правил — возможные продукции. Числа слева от дерева указывают, на каком контекстном уровне находятся узлы.

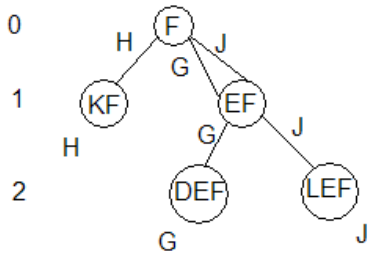

Рис. 1. Грамматика в виде дерева

Теперь перед нами встает следующая проблема: если мы будем генерировать новую последовательность строго по заданным правилам, то мы получим в точности исходную строку (мелодию), а мы хотим создать новую. Поэтому в ряде случаев мы должны не доходить до конечного правила, а взять промежуточное. Это означает, что иногда, встретив скажем такую последовательность: ADEF, мы не будем расширять контекст до конечного правила DEF -> G, а остановимся на F и рандомом или каким-то другим методом выберем любую продукцию (H, G, J) у узла F в дереве. Или остановимся

на EF и выберем продукцию G или J. Т.е. мы случайно выбираем уровень контекста в дереве и продукцию на этом уровне.
Ну а теперь представим, что каждая буква — это не буква, а аккорд, а применительно к MIDI-файлу будет вернее сказать, что это совокупность событий (к которым относится включение/выключение ноты, смена канала и т.д.), и вот уже наш алгоритм готов для мелодии.

Самыми используемыми в последние несколько десятилетий подходами к решению задачи алгоритмической композиции можно назвать несколько техник – Л-системы (системы, основанные на знаниях), клеточные автоматы, генетические алгоритмы, нейронные сети (обучающиеся системы).

Теория хаоса — математический аппарат, описывающий поведение некоторых нелинейных динамических систем, подверженных при определённых условиях явлению, известному, как хаос. К частным случаям теории хаоса можно отнести знаменитый Эффект бабочки, а к повлиявшим на развитие алгоритмического сочинения музыки - теорию Фракталов и *Л-системы.*

Работа Л-системы начинается с определения трех наборов параметров – алфавита (список символов, которыми будет оперировать система, как на входе, так и на выходе, в нашем случае, например, ноты), набора правил (например, классических законов гармонии) и аксиомы (символ или строка символов для начального ввода). Л-системы относятся к экспертным системам, основанным на знаниях и заданных наборов правил. К таким системам также можно отнести уже упоминавшиеся Марковские цепи, в которых вероятность перехода от одной ноты к другой вычисляется на основании матрицы вероятностных переходов, исходя из заранее определенных статистических правил.

Другой современный подход – использование *Клеточных автоматов,* изобретенных американским математиком Джоном фон Нейманом в 1966 году. Клеточный автомат может мыслиться как стилизованный мир. Пространство представлено равномерной сеткой, каждая ячейка которой, или клетка, содержит несколько битов данных; время идет вперед дискретными шагами, а законы мира выражаются единственным набором правил, скажем, небольшой справочной таблицей, по которой любая клетка на каждом шаге вычисляет свое новое состояние по состояниям ее близких соседей. Таким образом, законы системы являются локальными и повсюду одинаковыми.

Генети́ческий алгори́тм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию.

Первопроходцем в применении генетических алгоритмов в алгоритмический музыке считается Джон Бильс – профессор Рочестерского института технологий (Нью-Йорк, США), применивший его для генерации джазовой соло-импровизации в своей программе. Работа генетического алгоритма начинается с применения эквивалента биологического образования новых генов на пространство случайно распределенных решений для нахождения в итоге оптимального набора. Решения представлены хромосомами, а строки аллель – строками чисел, и задача рекомбинации генов сводится к созданию новых аллелей из аллель, взятых от родительских хромосом путем применения генетических операторов, в большинстве случаев, мутации и скрещивания. Перебирание хромосом продолжается до достижения определенного условия прерывания. Генетические алгоритмы в задаче автоматического музицирования разделяются по виду использованной фитнесс-функции – степень приспособленности может быть оценена исходя из заранее заданных четких условий, либо может быть определена непосредственно человеком при прослушивании и субъективной оценке.

Под *обучающимися системами* понимают системы, в которых не задано априорных правил, а система сама обучается чертам на примерах. Чаще всего, под такими системами подразумеваются искусственные нейронные сети, имитирующие работу нейронов мозга человека и способные обучаться на основе предоставленных примеров. Обычно такие системы используются в другой отрасли музыкальной математики: при распознавании музыкальных отрывков, например, для решения задач реставрации потерянных музыкальных данных или для составления автоматических партитур. Поэтому данный метод не сильно развит в алгоритмической музыке.

Существует несколько систем, использующих нейронные сети, например, HARMONET или CONCERT. HARMONET используется для оркестровки мелодий на основе обучении правилам гармонии И. Баха. Первым человеком, применившим нейронные сети к решению задач алгоритмической композиции, был Питер Тодд, профессор психологии и информатики из Калифорнии.

Также, одно из самых последних направлений в сфере алгоритмической композиции – моделирование эмоциональной нагрузки произведений для устранения проблемы сухого математического расчета и привнесения в машинную музыку элементов настроений и смысла, вкладываемых человеком в сочиняемые композиции.

Каждый водитель наверняка замечал за собой, что его манера вождения подстраивается под музыку, которая звучит в автомобиле: лирические мелодии настраивают на плавное и аккуратное вождение, а ритмичные и заводные склоняют к более агрессивному и рискованному. Немецкий автопроизводитель Volkswagen объединил усилия с музыкальным проектом Underworld и создал приложение для смартфонов Play the Road, которое работает с точностью до наоборот — генерирует музыку, исходя из стиля вождения конкретного автовладельца.

Приложение синхронизирует смартфон с бортовым компьютером автомобиля и считывает информацию о скорости, количестве оборотов двигателя, ускорении, вращении руля, плавном/резком торможении, времени суток и местоположении. Исходя из полученных данных, Play the Road создает уникальный саундтрек в режиме реального времени и тут же проигрывает его. Таким образом водитель становится композитором. В настоящее время приложение Play the Road находится в тестовом режиме.

Будет ли иметь успех подобное приложение среди музыкантов, сказать сложно. Но для людей, не владеющих музыкальными инструментами такие приложения позволят проявить творчество.

3. Интерактивное форматирование MIDI-строки.

Существует множество систем генерации, или «автокомпозиторов» которые способны генерировать звуки и тем самым сочинять музыку. Благодаря использованию *музыкальных библиотек* становится возможным выполнение сложных задач: от гармонизации мелодий, поиска закономерностей в наборе музыкальных произведений и формализации существующих методик композиции до генерации музыки с применением возможностей искусственного интеллекта.

Naudio – библиотека для работы со звуком на платформе .NET. Она предназначена для обеспечения набора полезных служебных классов. Рассмотрим функции библиотеки Petzold.dll в одном из существующих проектов. Реализуем немного теории на практике.

WPF-приложение будет выполнять функции:

- Генерация строки для проигрывания мелодий из xaml в midi;

- Воспроизведение аккорда, арпеджио.

В библиотеке Петзольда ноты записываются заглавными буквами от A до G, за которыми следует любое количество знаков «+» или «#» (повышение на полтона) либо знаков «–» или букв «b» (понижение на полтона), а за ними ставится необязательный номер октавы (октава, которая начинается со средней до, является четвертой). (Это стандартный способ нумерации октав.)

```
const string noteLookupString = "C D EF G A B";
```

Таким образом, C# на октаву ниже средней до записывается так:

C#3

Буква «R» сама по себе является паузой (rest). За нотой или паузой можно (но не обязательно) указать длительность, которая указывает период времени до следующей ноты.

```
double defaultDuration = 0.25
```

Например, вот это четвертная нота, каковой по умолчанию являются все ноты, если не указывается длительность:

1/4

Если длительность указывается не за нотой, будет использоваться последняя длительность. Если длительность начинается со знака дроби, числительное предполагается равным 1.

Эта длительность задает время до следующей ноты. Она также обозначает длительность ноты, т. е. время ее звучания. Для более отрывистого звучания (стаккато) вам понадобится, чтобы нота звучала короче, чем задано ее длительностью. А иногда нужно, чтобы ноты последовательности в какой-то мере перекрывались. Период звучания ноты указывается так же, как и длительность, но со знаком «минус»:

–3/16

Длительности и периоды всегда ставятся после ноты, к которой они применяются, но их порядок не имеет значения. Если период не указан, он приравнивается длительности.

Нотам также могут предшествовать лексемы (tokens). Чтобы задать голос инструмента, ставится буква «I» с номером патча, который отсчитывается от 0. Например, ниже задается скрипка с последовательными нотами:

I40

Патч по умолчанию — пианино.

Чтобы задать новую громкость (т. е. силу удара по клавишам) для последующих нот, используйте V:

V64

```
int volume = 127;
```

Как для I, так и для V, последующие числа должны укладываться в диапазон 0–127. По умолчанию темп равен 60 четвертным нотам в минуту.

```
int tempo = 60;
```

Чтобы указать новый темп для следующих нот, используйте T с числом четвертных нот в минуту, например:

T120

Если вам нужно сыграть группу нот с одинаковыми параметрами, заключите их в скобки. Вот аккорд до-мажор:

(C4 E4 G4 C5)

В скобках могут появляться только ноты. Вертикальная линия «|» разделяет каналы. Каналы проигрываются одновременно, и они полностью независимы, в том числе могут иметь свой темп.

Если в любом месте канала содержится заглавная буква «P», этот канал становится перкуссионным. Такой канал может содержать ноты или паузы в обычной нотации; кроме того, допускается задание перкуссионных голосов с помощью чисел. Например, это колокольчик:

P56

В формате MIDI-строки мелодию “Charge” можно выразить так:

"T100 I56 G4 /12 C5 E5 G5 3/16 -3/32 E5 /16 G5 /2"

MidiStringPlayer — единственный открытый класс в проекте библиотеки Petzold.Midi, включенном в комплект исходного кода, который можно скачать для этой статьи. Он наследует от FrameworkElement, поэтому вы можете встраивать его в визуальное дерево в XAML-файле, но он не является видимым элементом. В свойство MidiString запишем строку в формате, показанном в предыдущем примере, и вызовем Play (а если хотите, то и Stop, чтобы остановить воспроизведение последовательности до ее окончания).

В MidiStringPlayer также есть свойство PlayOnLoad для воспроизведения последовательности при загрузке элемента, и свойство IsPlaying только для чтения. Этот

элемент генерирует событие Ended по окончании воспроизведения последовательности и событие Failed, если в синтаксисе MIDI-строки обнаруживается ошибка. Это событие включает смещение в текстовой строке, указывая на проблематичную лексему, и текстовое описание ошибки.

Воспроизводим каждую нотку следующим образом:

```
outputDevice.SendNoteOn(Channel.Channel1, Pitch.C4, 80);
Thread.Sleep(500);
outputDevice.SendNoteOn(Channel.Channel1, Pitch.E4, 80);
Thread.Sleep(500);
outputDevice.SendNoteOn(Channel.Channel1, Pitch.G4, 80);
```

Арпеджио:

```
outputDevice.SendControlChange(Channel.Channel1, Control.SustainPedal, 127);
PlayChordRun(outputDevice, new Chord("C"), 100);
outputDevice.SendControlChange(Channel.Channel1, Control.SustainPedal, 0);
```

4. Заключение

В данной работе рассмотрены различные методы алгоритмизации и перспектива их развития. Также рассмотрены некоторые музыкальные библиотеки и WPF-приложение, позволяющее воспроизводить заданные ноты, генерировать текстовую строку.

Список литературы

1. Генерация музыки на основе заданного стиля. Режим доступа: http://habrahabr.ru/post/69985/ (дата обращения 15.05.2014).
2. Петцольд Ч. MIDI-музыка в WPF-приложениях. Режим доступа: http://msdn.microsoft.com/ru-ru/magazine/ee336028.aspx (дата обращения 15.05.2014).
3. Черненко В. Музыкальное программирование. Режим доступа: http://zillion.net/ru/blog/333/matiemuzyka-3-muzykal-noie-proghrammirovaniie (дата обращения 15.05.2014).