Computer Music

Assignment 5

Delimited Set Multiplication

1 Objective

In this project, you will enrich your set multiplication algorithm. In the set-multiplication video, you may have noticed some sneaky chicanery: we freely chose our chords for set multiplication somewhat arbitrarily led by aesthetic taste, and yet they all somehow magically formed an octatonic scale. How can this be?

In this project, we get to the bottom of that question by solving the problem systematically through exhaustive search. This is meant to serve as an example of one of the many ways computation can augment a creative compositional process, allowing us to solve musical problems and achieve aesthetic goals that couldn't be realized without these tools—or at least not easily.

1.1 Philosophical Context

For deeper insight into the philosophical underpinnings of this project, consult Michael Gordon's reflections on irreducible algorithms in composition:

`https://michaelgogins.tumblr.com/ComputerMusic.html`

If the content no longer discusses computational irreducibility, access it via the Wayback Machine.

In short, some of the richest areas of exploration in Computer Music are those that don't replicate existing musical practices, but yield new aesthetic possibilities only made practical—or even possible—with technological augmentation. This project provides one example of this kind of research and serves as a fitting culmination to all previous exercises.

2 Setup and Installation

2.1 Install Previous Projects

First, verify that all previous Java projects are installed in your local Maven environment, as this project depends on each of them.

Navigate to each directory one at a time:

1. `Set-Theory-Calculator`
2. `Stepwise-Voice-Leading`
3. `Set-Multiplication`

In each folder, navigate to the Java option (the directory level containing `pom.xml`). In this directory (the root of the Java project), run:

```
mvn clean install
```

2.2 Clone the Starter Repository

Once you've installed the dependencies, clone the starter repository:

```
git clone https://github.com/twmcdunn/Delimited-Set-Multiplication.git
```

All dependencies should resolve automatically. Many of them are modules you've implemented yourself, which can be located since they are installed to your Maven environment.

2.3 Copy Your Implementation

Copy your existing implementation class, `SetMultiplication.java`, from the previous project into the new project in the `delightofcomposition` folder. We will enrich it while maintaining backward compatibility.

3 Algorithm Overview

In broad strokes, your enriched class will:

1. Take as input a chord cardinality, a target pitch collection, and a depth of set multiplication
2. Find all unique chords (set classes) of a given cardinality (using set-theory dependencies)
3. For every unique pair of these chords, give each chord a turn being the multiplicand
4. For each member of the multiplicand chord, build one of the two chords off of this note (the "root," if you will)
5. Consider every possible sequence of multiplier chords (within the given pair) to build off of each chord member of the given multiplicand
6. If any of these configurations yields the target pitch collection (when all notes of the resulting progression are collected and put into prime form), store it as a solution within an `ArrayList` of `Solution` objects
7. Perform set multiplication to compose a musical texture using solution(s) that satisfy the constraints

Consult the abstract class `Multiplication.java` in the new project for detailed specifications. Note that this class is an enriched, backward-compatible version of the corresponding class in the old project.

4 Implementation Details

4.1 Add Imports

Add your imports to `SetMultiplication.java`:

```
package org.delightofcomposition;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;
import java.util.stream.Collectors;

import org.delightofcomposition.util.Types;

import war.PrimeFormCalculator;
import war.CombinationsCalculator;
```

4.2 Implement Default Constructor

Implement a default constructor in `SetMultiplication.java`. We need to be able to construct this type of object without specifying multipliers (since they will be chosen procedurally from all possible options):

```
public class SetMultiplication extends Multiplication {

    public SetMultiplication() {
        rand = new Random(123);
    }
```

4.3 Building Superset with Java Streams

As a hint, the syntax to perform step 6 in the algorithm overview can be quite concise using built-in Java classes. It looks similar to JavaScript.

6.a. Convert the results of `multiply(multiplicand, multiplierIndexesArray)` to a stream. This is done by calling `.stream()` on the `ArrayList`, which returns a `Stream`.

> A Java `Stream` has many helpful built-in methods to transform the shape of the data structure it represents, filter its contents, map each of its items (1-to-1) to new objects, or collect its items into a new object. Each of these methods returns another object (usually a `Stream`) on which we can call another such method.

6.b. First, let's change the shape with `flatMap`. Specifically, call

```
.flatMapToInt(Arrays::stream)
```

on the stream. This "flattens" the data, so that a 2D stream of items becomes a 1D stream of the same items in the same order.

In other words, we're taking all the notes that are grouped into chords (a stream of `ArrayList`s `[note1, note2, note3] [note4, note5, note6]`) and putting them into a single "superset" (a stream of ints `[note1, note2, note3, note4, note5, note6]`).

`Arrays::stream` is a function to call for each chord to produce the stream of integers from an `ArrayList`. In other words, before flattening, for each item in the original stream, we're calling `Arrays.stream(chord)`, where `chord` is an `ArrayList`.

6.c. Now, let's filter out any repeated notes in the superset. Simply call `.distinct()` on the stream that resulted from step 6.b.

6.d. Now wrap the stream's primitive ints in `Integer` objects. Simply call `.boxed()` on the stream that resulted from step 6.c.

6.e. Now collect the results in a new array by calling

```
.collect(Collectors.toCollection(ArrayList::new))}
```

You can do all of this in one line, or separate lines without semicolons. It returns a superset of which you can find the prime form using your set-theory dependency.

5 Execution

5.1 Uncomment Main Code

Once you've finished implementing, uncomment these lines in `Main.java`:

```
Multiplication m = new SetMultiplication();
ArrayList<int[]> chords = m.compose(scale, chordCardinality, depth);
generateComposition(chords, 0.05, synth, new
    VoiceLeading()::uncommonDirectedVoiceLeading);
```

5.2 Run the Program

Now run the program in VS Code, another IDE, or pure Maven:

```
mvn clean compile
mvn exec:java
```

This brings up a command-line interface (CLI) which lets you make choices before producing the musical texture.

6 Using the CLI

6.1 Key Parameters

There are two choices that significantly impact the color of the resulting musical texture:

1. **The target pitch collection/scale:** Hard-coded in `Main.java`. Change `scale` to taste.

2. **The chords used in set multiplication:** Chosen within the CLI from the possibilities of the hard-coded value of `chordCardinality` (which you can also change).

6.2 Navigating the CLI

To use the CLI:

- Browse the solutions with the arrow keys
- When you see solutions that strike your fancy, type their respective solution numbers separated by commas
- To filter options by chord, type comma-separated note names instead of numbers, with capital letter names

6.2.1 Example: Filtering by Chord

If you're interested in progressions that have a minor chord:

1. Set `chordCardinality = 3` prior to running the CLI
2. Within the CLI, type `C,Eb,G`
3. Notice that no solutions show up until you finish this string, since there aren't any solutions with chord-cardinality 3 that have the chord C,Eb

Once you've selected the solutions you're interested in (either with note names or solution numbers), hit Enter to generate the texture.

7 Envelopes

Again there are envelopes to play with, as in the last project. Consult the documentation of the previous project for details—they are the same envelopes controlling:

- **Env 1:** Range
- **Env 2:** Amplitude
- **Env 3:** Density
- **Env 4:** Pan

Access the envelope gui as before:

```
mvn exec:java -Dexec.mainClass="org.delightofcomposition.envelopes.GUI"
```

8 Final Composition

Compose a brief piece for xylophone and fixed media using this tool.

> **Optional:** If it strikes your fancy, you may want to consider replacing `resources/4.wav` with a metallic sound to blend with the xylophone. If you do that, you will want to update `synth.origFreq` defined in `Main.java` to match the fundamental frequency of the new sample you use.

Your composition should demonstrate:

- Effective use of the delimited set multiplication algorithm
- Thoughtful selection of solutions that satisfy your aesthetic goals
- Thoughtful control of musical parameters through envelopes
- Musical coherence between the electronic fixed media and the xylophone part