Concerto for Flute and Orchestra,
first movement

Hugo Harmens

Schulich School of Music

McGill University, Montréal

April 2015

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree Master of Music.

Abstract

The submitted movement is the first of three movements of my *Concerto for Flute and Orchestra*, commissioned by Belgian flutist Oriana Dierinck for her flautO project. I composed this movement using algorithmic and computer-assisted composition techniques programmed from scratch. The program controls certain aspects of rhythm, pitch, dynamics, articulation, orchestration and form, with the goal of expanding further to control as many details of the generated music as possible, including instrumental techniques and timbre. I used the programming language, Ruby, to create the code needed, while the text-based notation software, LilyPond, is used to generate the actual score. Some manual editing of the score and the music has been necessary, but the majority of the musical information for the movement has been generated by the program itself. The intention is to broaden the scope of my own compositional practice and its results through detailed investigation of composition technique and musical concepts as automated procedures. This allows for music that I would otherwise not have written to be created, while still adhering to my own compositional principles and aesthetics.

Résumé

Le mouvement présenté est le premier d'une suite de trois mouvements tiré de mon *Concerto pour Flute et Orchestre*, commissionné pour le projet flautO par le flûtiste belge Oriana Dierinck. J'ai composé ce mouvement utilisant l'écriture algorithmique ainsi que des techniques de composition assistée par ordinateur dès le début de la création de l'oeuvre. Le programme contrôlait certains aspects de composition tels que: le rythme, les notes utilisées, les nuances, l'orchestration ainsi que la force du mouvement, ayant pour but l'accroissement du contrôle du

plus de détails possible de la musique générée, incluant les techniques instrumentales et le timbre. C'est en utilisant le langage de programmation Ruby pour la création de codes ainsi que le logiciel libre de notation musicale LilyPond que la partition actuelle fut créée. Certains aspects d'édition manuelle de la partition furent nécessaires, mais la majorité du contexte musical du mouvement fut créé par le programme lui-même.

L'intention musicale de ce travail consiste à élargir le champ d'application de mes propres techniques de composition ainsi qu'une analyse détaillée de la technique de composition et des concepts musicaux des procédés automatisés.

Ce travail me permet donc d'avoir créé une oeuvre musicale complète, que je n'aurais peut-être pas créée par moi même, tout en utilisant mes propres principes et esthétiques de composition.

Acknowledgements

I would like to extend my gratitude to my advisor, Prof. Melissa Hui, for excellent guidance throughout the process of writing this thesis, as well as Prof. Brian Cherney who tutored me during my first semester at McGill.

Prof. Jon Wild – for giving advice on programming languages and methodology.

Prof. Chris Paul Harman – for continued support and practical advice.

Composer Arnt Håkon Ånesen – for your friendship and honest feedback.

Composer Scott Rubin – for wonderful conversations, and for lending me your viola.

Flutist Oriana Dierinck – thank you for believing in me, and for the commission.

And, finally, a big thank you to all the musicians I have consulted with numerous questions about their particular instrument: Julie Harnois, Mark Bradley, Maarten Vos, Hinse Mutter, Gabriel Wolff, Iason Tzanakos.

Table of Contents

1. Introduction

I believe the term "algorithmic composition" should be understood in the broader sense:

> *"...the study of algorithmic techniques — although not explicitly formulated as such — represents an essential part of [university educated students'] education. Subjects such as counterpoint and harmony are disciplines that are to a high degree determined by rules, and practicing them teaches students to work in formalizable systems." (Nierhaus 2010).*

Essentially, there are algorithmic elements in a lot of the work we do as composers. An algorithm is basically a step-by-step procedure intended to solve a specific problem, mathematical or otherwise. As Nierhaus points out, though, the techniques we learn and practice are usually not categorized explicitly as algorithms, perhaps because as creative human beings we tend to apply systems and procedures (and their outcomes) somewhat loosely, thus escaping the strictness associated with an algorithmic approach. An algorithm need not be strict. It may involve several random or semi-random steps, and may even be designed to produce an unpredictable result.

There has been a growing interest in algorithmic composition since Hiller and Isaacson's *ILLIAC Suite* in 1956 (commonly understood as the first completely computer-generated composition), with a myriad of computer programs specifically designed for this purpose. Programs such as *OpenMusic* (developed at IRCAM) and *MaxMSP* are well-known and frequently used by many composers. Among the lesser known programs we find Michael Edwards' *Slippery Chicken*,

which has been pivotal to my understanding of how to build an algorithmic composition program from scratch. Edwards' website contains a plethora of detailed information about how his program works, including detailed video tutorials with clear examples.

Discussing the applications of algorithms in composition, Nierhaus concludes that algorithmic procedures are mainly used in the field of style imitation, with composers such as David Cope in the forefront of these experiments *(Nierhaus 2010)*, while using algorithms for what he describes as "genuine composition" is less common.

Some composers, such as Magnus Lindberg (in his work *Engine* (1996)) and Asbjørn Schaathun, have defined rules and constraints based on their own style. Lindberg[1] uses the program *Patchwork1,* a predecessor to *OpenMusic*, developed at *IRCAM*, while Schaathun also writes his own code, which he explains serves as "…a practical tool, as sort of extension of myself, for my own composition" (Schaathun 1988/89). Schaathun's aim with his program package *"Rythmod,"* was to create a bridge between an intuitive, improvisational approach to compositon, and a more structural approach. Both Lindberg and Schaathun have an attitude towards the computer-generated music as "raw" material to which the composer himself will add the final touches and/or shape into actual music. Other composers, such as Joshua Fineberg and Gustavo Díaz-Jerez, are utilizing alogrithmic approaches and computer applications in their realizations of concepts within the realm of spectral music.

The dominant legacy in the field has been focused mostly on manipulating the parameters of pitch and rhythm. Recent developments in the field include the autonomous computer *IAMUS*, developed by a research team based in Spain. As a part of the larger domain of Artificial

[1] http://www.musicsalesclassical.com/composer/work/7693 (accessed Apr-14-2015)

Intelligence, *IAMUS* creates complete works without human intervention, including the production of score and parts, through the use of genetic algorithms.

After examining the many possible entry points into algorithmic composition, I decided to program what I needed from scratch, especially since my ultimate goal is to run the program, and instantly have a full score generated, rather than just using these tools to simply generate source material which I will later shape myself. My approach is thus closer to that of the *IAMUS* team, ultimately aiming at including all musical parameters in generating a full score of playable and viable music, or indeed many such works, going beyond an algorithmic computer-assisted approach used to generate raw material to be further shaped by the composer. The aesthetics and compositional approach, however, are based on my own accumulated knowledge as an individual artist, and my own aesthetic ideas, although I am certainly attempting to expand the boundaries of what my music is and may become. In this sense I share ideological grounds with Asbjørn Schaathun, using the computer "as a psychoanalyst" (Schaathun 1988/89).

I have some experience with programming, mostly web oriented, but also from doing a similar experiment as this one in 2012, where I used JavaScript as a programming language, running the code through my web browser every time, which was far from optimal. I searched for a simple and efficient language that I could run through an interpreter, and that would give me a high level of control and flexibility. After doing some research, I decided to learn Ruby.

2. Methodology and philosophy

2.1 Programming in Ruby

Ruby is a very straight-forward programming language with a fairly simple syntax, and often several ways of accomplishing the same task. It is object-oriented to the core, as everything in Ruby is treated as an object. Learning Ruby from scratch was still quite a task, especially since I hadn't done much programming the past year or so when I began preparing for the process of writing this program.

The Ruby installation for Windows comes with a work environment called *Interactive Ruby*, which allows for running code in real time, line by line. This is extremely useful, especially while learning the language, to test small concepts before entering the code into one's program. Various internet resources have also been very helpful in dealing with the programming language. The Ruby interpreter is also fairly specific with regards to error reporting, which increases the overall ease of use.

Essentially, my decision to program from scratch to suit my own needs gave me more control, but also a severely bigger challenge, as all necessary information needed to be entered into the program. Finding a system for handling rhythm was one of the main challenges I had to face.

2.2 Music notation using LilyPond

When programming in JavaScript in 2012, the output was ultimately presented to me as text in my web browser, describing the pitches, rhythms and other information, which I then had to

manually enter into a music notation program. The text-based music notation software LilyPond simplifies this process. The music is written in formatted text in a simple text file, which is then interpreted by LilyPond and converted into music notation (PDF), with an accompanying MIDI-file for sound output. LilyPond is certainly not perfect, and with larger scores it takes a while to convert the information into actual notation. One then has to edit the text file rather than the graphic score to make any necessary changes. The engraving, however, is quite well done, and the automatic layout works fairly well. The software does also offer a fair amount of flexibility and editing possibilities. Some of the settings require quite some effort, but once they have been properly configured, they can easily be employed on a score-wide level.

For algorithmic and computer-assisted composition, especially when programming from scratch, LilyPond is an indispensible tool, both for generating little snippets during the programming process to learn about how a certain algorithm functions, and to further improve it, and for a larger final score of automatically generated music.

LilyPond has allowed me to generate well over 1000 output samples focusing on either pitch or rhythm, or a combination of the two. It allows for easy access to the generated material, rather than interpreting text and only imagining the actual music and notation. Once I had created conversion tables converting my pitch and rhythm information (mostly consisting of numbers and a distinction between sound/rest/tied from/tied to) to the LilyPond equivalent, creating multi-part scores with the generated material was fairly easy. For a final score for full orchestra, however, a lot of editing is necessary.

2.3 The composer versus the computer

As an improviser, I am used to accepting randomness and spontaneity within a controlled framework. As a composer, the process I go through is often very similar, but allows for more reflection and shaping of the material with a bird's eye view, refining the music until a satisfactory result has been achieved. The process of programming has its own unique set of challenges, advantages and perhaps even disadvantages, but moreover it is quite similar to other ways of creating new music. This is particularly true when one programs everything from scratch rather than using existing software or templates. In essence, the computer and the programming environment is a space to experiment with and formalize structural concepts and ideas, and be able to test their limits and implications. It is a tool for learning, for developing one's compositional techniques, to search for solutions, to go into the depth of one's own musical and creative thought.

Feeding a vast amount of the knowledge and experience I have so far accumulated into a computer program is both eye-opening and inspring. Both music itself and the process of composing it functions on so many levels and layers, that it is sometimes hard to wrap one's head around the complexity of it. Handling one task at a time, therefore, is essential. When programming, one has to work this way anyway, since the program has to be constructed block by block. From something as simple as calculating the *bpm* value of a particular duration in a particular tempo to the placement of ties within a measure, every task must be meticulously written down as computer code, and thus clearly envisioned in the mind of the composer/programmer. The computer programming environment, then, becomes more of an extension of the composer's conceptual space than an opponent or a "threat" of some sort. This

is particularly true in the case of *one* composer writing a program from scratch, as opposed to either a team working together to develop composition software from scratch, or one composer using already existing tools that may have certain aesthetic and technical qualities not necessarily stemming from the composer him/herself, but from the software developers. There are of course varying degrees to which the tools out there allow for a personalized process of composition, but most of the already existing software will have certain methods and ways of working that may have clearly recognizable stylistic features.

2.4 Aesthetic considerations and artistic vision

A tendency many artists have is to walk in one's own footsteps. This is a double-edged sword. On the one hand, a clear and recognizable personal "style" is often pointed to as a sign of great artistry, great master of one's craft, and a particular clarity of artistic vision, philosophy and personality. On the other hand, a composer who keeps walking in his/her own footsteps and is always stylistically coherent within the framework of his already established style of writing, may very well have stagnated. An important philosophical question for every artist is whether reinventing one self and one's artistic and technical approach is seen as an important part of one's artistic quest, or if crystallizing a particular method, technique, style or artistic "persona" is the main objective of one's work. My opinion is that there should be a balance, and that these two aspects of the artistic mission are not mutually exclusive. By balance I simply mean that one should always seek to expand, to learn more, to develop, and to include a vast array of stylistic and aesthetic concepts in one's musical palette, while at the same time making sure to go deeper

into each of the avenues chosen, to shape one's artistic diamonds. This is particularly true when something has been found which clearly inspires a further exploration and utilization.

There were several ideas and concepts I wanted to explore in the first movement of the flute concerto, as well as within the realm of algorithmic composition and programming in general.

1. *Unpredictability*. The element of surprise, of not always taking a tendency to its natural conclusion, at least not in a linear and direct way, is something I always appreciated, and something which for me is one of the trademarks of well-composed music. For obvious reasons, predictability and unpredictability function more on the larger scale than on the micro level, but even there one can lay out the sonic canvas in such a way as to yield a sense of unpredictability. This can be done by establishing a clear character and tendency, just to move away from it as soon as it has been established *enough*. Of course, if one does this repeatedly, this process loses its power and becomes predictable in itself.

2. *Irregularity*. Just as unpredictability has been a focal point in my work, so has irregularity. A music that becomes too regular and static, both in terms of larger form, and how the music is shaped from moment to moment, easily becomes tedious and uninteresting. A sense of irregularity may be achieved in many ways, some more subtle than others. Changing the overall average rhythmic density just a little can be just as effective as a drastic tempo change. Creating a sense of irregularity also requires establishing a sense of regularity, otherwise the irregularity itself becomes regular.

3. *Instability*. As with the first two points, the concept of instability plays on the same notion of departing from the monochrome and monotonous in more than one parametric realm. Thus, pitch, rhythm, timbre, density, texture, articulation, dynamics, register and other parameters, may all be molded through the same filters. Instability requires a sense of stability, just as unpredictability needs to relate to an established sense of predictability, and irregularity relies on the notion of regularity. It is always a play between the established and the contrasting characters. How much doesn't a sudden staccatissimo on three notes stand out in a landscape of mostly soft legato and portato playing? If a mashup of the two different characters was already established, nobody would notice. Such a mashup, however, is also a character in itself, from which one may depart. The specifics of a certain character, then, becomes a lense through which one views the possible departure points and their implications. A clear sense of instability will often require more extreme means than simply creating moments of unpredictability or a general irregularity. Instability is close to being a character of its own, chaotic in nature.

4. *Density and textural development.* Texture in music has been one of my focal points for quite some time. Density, both horizontal and vertical, is an inherent part of the textural makeup of any piece of music. Textural development and diversity is one of the greatest tools a composer can use to keep the music interesting, and the listener interested. As any tool, though, it needs to be utilized cautiously. Too much textural diversity may easily lead to lack of musical identity and character, just as mixing all the colours on one's canvas results in a grey and fairly uninteresting mass. As a part of the overall textural approach, I have had an interest in exploring multiple simultaneous layers who all have

their unique characteristics. How can I as a composer lay the grounds for a perception of stratification and the simultaneous presence of multiple unique characters? How different does each layer need to be, and where do several layers begin to interact and rely on each other for their musical functions? At which point do they start merging, and why? These are all questions I continually ask myself as a composer. In addition, the listener's role as a "co-creator" of the artwork through perception plays a part. I can only do so much to convincingly "guide" a listener in his or her listening process.

5. *Rhythmic flexibility*. Again, this concept fits the overall theme of creating a music which is dynamic, and which fluctuates. On the other hand, as I will explain more in detail later when discussing the specifics of rhythmic generation in my program, a relatively flexible rhythm may still have a crystal clear character. It depends on the type of flexibility employed. For me, some of the most interesting discoveries during the process of programming was made in the realm of rhythm. These are discoveries I could not as easily have made without using these particular methods, allowing for quite complex concepts to be carried out instantaneously and effortlessly, once the basic code had been constructed. Once this task is completed, it enables a systematic exploration of the concepts in question.

   Rhythmic flexibility as a core concept arose out of pure necessity, as I have often struggled with this aspect in my previous works. I have attempted to achieve a more fluid and flexible way of expressing rhythm, but have had difficulties in creating rhythmic characters that have both a clear character and is flexible and irregular, without becoming

too blurred or unfocused. Exploring the impact of various parameters of rhythm and combinations of these, have helped me greatly in exploring what to me used to seem like an overwhelming mystery locked in a secret box in a foreign land.

6. *A more abstract pitch world*. Ever since I started composing, pitch has been a challenge for me to work with. I have often struggled to find pitch systems and pitch worlds that represent my musical and aesthetic visions of both the abstract and fluctuating on the one hand, and on the other hand a steered developmental trajectory guiding the listener through a more comprehensible and clear pitch language.

7. *A flexible form of multi-layered textures*. In terms of form, I am aiming for a section-based, but still fluid, shaping of my music. For this movement, and the program in general, I have focused on creating separate layers that have both an internal development in pitch, rhythm and register, and an influence upon each other. Instrumental groups are static within each section, but flexible for the entirety of the form. Layers have varying start and end points, and may overlap with other layers, creating an ever-evolving form moving from texture to texture, with certain passages of a more static character. Combining different instruments for every section, this evolution will also happen on a timbral level. Eventually, the program is intended to include a vast array of playing techniques and timbral possibilities. For the first movement of the flute concerto, however, the focus is more on pitch and rhythm, and less on specific playing techniques.

3. The program and the precompositional process

3.1 Overview of program structure

The program consists of four files dealing with different aspects of the compositional process. The *basicdef* file controls the overall process, including the number of parts, formal sections, and so on. The *rhythm* file contains the basic rhythm generation algorithm, along with code for testing rhythmic output and the code that formats all the LilyPond code to be sent to each part and later written to file. The *pitch* file contains the pitch algorithm, and code for testing pitch output with a static rhythmic pattern or rhythms generated by the code in the rhythm file. Finally, there is the *instruments* file, which contains information about each instrument, including its range, its type, its general timbre, etc. While working on different aspects of the program, these files have either been run on their own, or in combination with one or more of the other files. The *basicdef* file is the file one must run in order to create a full score in LilyPond format.

3.2 Concepts of rhythm

3.2.1 Overview

Rhythm is, in my experience, a particularly challenging musical parameter to program, but it is also a perfect fit for algorithmic composition techniques because it uses numbers, and in many ways the translation between the representation as numbers and what we experience as a listener is fairly direct. This is evident with, for example, tuplets, which are based on odd subdivisions of the whole note, such as 3, 5 or 7. This basic subdivision of the whole note (1/3, 1/5, 1/7), can then be divided by 2, 4 and so on, and grouped, thus paving the way for values such as 1/12

(triplet 8th), 3/28 (dotted 8th within a septuplet 16th note group), etc. The different basic subdivisions of the whole note are both logical in the mathematical sense, visible in the notated music, and, at least to some extent, audible to the listener. For example, a narrow or wide range of sounding durations will also have a direct and audible musical impact.

In order to work efficiently and create a system for rhythmic generation that functions well, I chose to only use the time signature of 4/4 for the first movement of the flute concerto, and up to this point in the development of the program. The system I have developed can be expanded to include other time signatures as well, but 4/4 was chosen because it provides a good basis for testing and developing rhythmic generation, also because its length corresponds to that of the whole note. Thus, the *length* of a particular rhythmic line can easily be represented by a whole number.

Another restriction chosen up until this point in the development of this program has been the exclusion of nested tuplets, and the limitation of tuplets filling a whole quarter note beat, or 8th note in the case of a 16th note triplet figure. Tuplets extending over a 1/4 note beat are only achieved through the use of ties to a tuplet figure of the same basic beat subdivision (3, 5, 6, or 7) on the following beat. Thus, a rhythmic configuration not allowing for the use of ties will not include over the beat tuplet figures at all.

I have chosen to use a system of representation corresponding to the actual durational values as fractions of the whole note, which is exactly the way our notation system and logical thinking about rhythm works; a 16th note is called such because it is 1/16 of a whole note, which is 1, or 1/1, or 4/4, etc. Thus, a triplet 8th note is 1/12, a dotted 8th within a septuplet 16th group is 3/28. Values that cannot be *notated* without using a tie or double dot are not included in the list of *basic values* (see Figure 3-2).

The basic rhythm generation algorithm ensures that sounding durations are within a set range, for example between 1/16 and 7/8. Furthermore, parameters are set that control the inclusion/exclusion of ties (for now 0 or 1), tuplets, the maximum number of basic values allowed, values used for rests, maximum amount of rests, and distribution of rests (complete overview given in Table 3-1). The algorithm outputs a string of rhythmic values that fit notational conventions (without using nested tuplets or tuplets extending over beats) within a 4/4 time signature, of a total duration of x measures/whole notes.

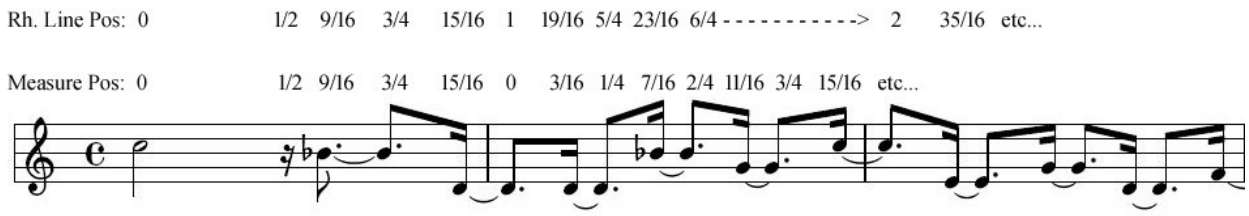


Figure 3-1: Example of positions within a longer rhythmic line and per 4/4 measure

In order to respect both notational conventions and the desired duration range, a system of *positions* is used. Each basic value has a set of positions where they may be placed within a 4/4 measure. Position values are also used to represent exact points within one generated rhythmic line, and for analysis and combination of multiple rhythmic lines. Thus two parallel position charts are always in use: one within a single measure, the other for the entire length of the generated line (see Figure 3-1). The starting position value is *zero*, and as a value is added, the new position is the sum total of added values up to that point. Thus, in a generated rhythmic line using a 4/4 time signature, *position 2* is at the very beginning of the third measure. In other words, *two whole notes* is the time that has passed. Similarly, every measure starts at internal position *0*, and all subsequent positions would be the sum of the values added up until that point.

Position 1/4 is thus after the first quarter note. The following is an overview of all basic values from the 32nd note up to the whole note, and their respective positions.

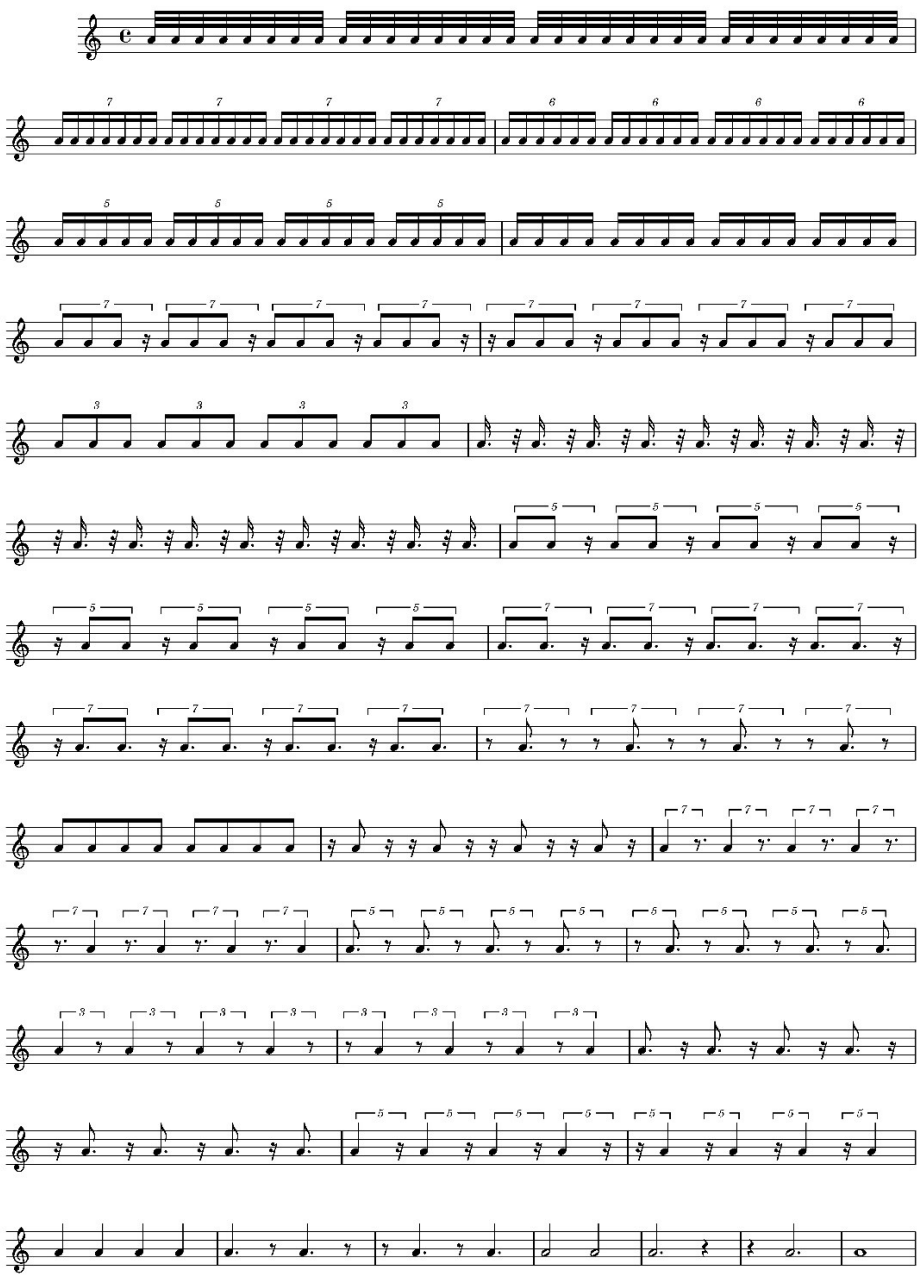


Figure 3-2: Positions within a 4/4 measure, for values 1/32 up to 1/1 in ascending order

This list of positions does not exhaust all possible positions for all values. This is particularly true for the triplet-based values, where positions 1/24, 1/8, 4/24 (for the first beat) and so on are excluded for the 1/12 value, 1/24 after the beat are exluded for the 1/6 value, the 1/8 value represented as a *dotted* 8th is not used within triplet figures, thus excluding positions 1/24 and 1/12 after the beat. For the 16th note values positions 1/32 and 5/32 after each beat are excluded. These positions may be included in the future, along with position charts for other time signatures.

Ties are generally only allowed across beats and measures, with a few additional "special cases" where ties are used within a ¼ beat. This list is, just as the list of positions, not an exhaustive list. Of course, in theory any value may be tied to any other value, but this will often result in a notation which does not comply with established conventions, and may be hard to read and require additional editing for clarity (an example would be two 8th notes tied together starting on the beat). It is important to note that at its current state the program only allows for the use of *one tie at a time* (i.e. two values tied together), but support for multiple tied values will be added in the future. The Ruby code below shows the "special cases" for values that may be tied together, even within a ¼ beat. The order is always respected, meaning that combinations including 3/8 values will always have this value to the left tied to another value (for example 1/16) to the right. This list may of course be expanded in the future.

```
tiecombos = []                              # Values that can be tied; special cases
tiecombos.push([Rational(1,24), Rational(1,16)])
tiecombos.push([Rational(1,16), Rational(1,24)])
tiecombos.push([Rational(4,28), Rational(2,28)])
tiecombos.push([Rational(2,28), Rational(4,28)])
tiecombos.push([Rational(4,28), Rational(1,28)])
tiecombos.push([Rational(1,28), Rational(4,28)])
tiecombos.push([Rational(3,8), Rational(1,16)])
tiecombos.push([Rational(3,8), Rational(1,24)])
tiecombos.push([Rational(3,8), Rational(3,32)])
tiecombos.push([Rational(3,8), Rational(1,32)])
tiecombos.push([Rational(3,8), Rational(1,12)])
tiecombos.push([Rational(3,16), Rational(1,32)])
tiecombos.push([Rational(1,32), Rational(3,16)])
tiecombos.push([Rational(1,6), Rational(1,24)])
tiecombos.push([Rational(1,24), Rational(1,6)])
```

Figure 3-3: Ruby list of allowed within-the-beat ties

3.2.2 Specifics of basic rhythm generation

Values for the generation of rhythms are initially picked randomly from a selection of the entire set of values, based on the aforementioned parameters. When maximum number of ties is set to 0, all values *shorter* than the desired minimum duration are excluded from the possible choices. This is true also for all tuplet-based values. All values that are *longer* than the desired maximum duration are also excluded. From the remaining list, x values are picked at random. If the number of values desired is larger than the number of values in the list, the entire list is selected for further analysis and configuration.

The algorithm then examines the positions, values and resulting (tied) durations to make sure all positions used can:

1) *be reached* through the use of at least one value in the list (may also be the value itself),

2) result, with the value in question, in a position *where at least one value can be placed*, and

3) *be tied to* another value if too short, resulting in a value which is neither too long nor too short, either backwards or forwards. In certain cases, a value at a specific position can only be tied to *or* from, and not both. An example of this would be quarter note values at position 0 within a measure when the shortest permitted duration is longer than a whole note; quarter notes at these position can then only be tied *to*, and not from (using only one tie at a time), as this would result in durations that are too short.

Values and positions are examined *one at a time*, and each position that does not comply with the requirements will be removed from the position chart for that specific value. If a value reaches a state where it cannot be placed anywhere within the measure, it is replaced. If the number of

desired basic values is equal to the number of possible values when this state is reached, the number of values to be examined is reduced by 1. This also happens at certain intervals when/if no combinations are found. This is to ensure that the program doesn't get stuck in an endless loop due to a too high number of desired values compared to what is possible. These types of limitations also happen the other way around; certain durational ranges require a minimum number of basic values. This is particularly true when the minimum desired duration is longer than a whole note (1). An example would be a range of 5/4 (whole note plus quarter note) up to 7/4 (whole note plus dotted half note). This requires a minimum of 1 tie, as well as a minimum of 4 basic values (if more than 1 tie is allowed, the number of possible values may increase); the quarter note, the half note, the dotted half note, and the whole note. Whole note tied to quarter note requires the dotted half be used to fill the rest of the measure, which can then be tied to the half, dotted half or whole note, etc. The two middle positions for the quarter note are not used, while the 0 position (beginning of the measure) can only be tied *to*, and the ¾ position (completing the measure) can only be tied *from*. The same is true for the half note and dotted half; ties to or from these values can only happen across the barline, to create durations longer than 5/4.

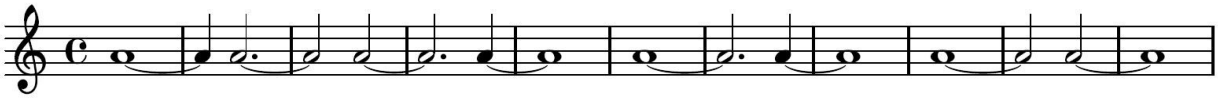


Figure 3-4: Durations of 5/4 through 7/4 using a maximum of one tie per duration and four basic values

As soon as a set of basic values and corresponding positions are chosen, a random value is picked for every resulting measure position, ensuring that values that need to be tied (if ties are allowed) are placed in such a way that the resulting durations are within the desired range. This

means looking backwards at the two previous values as well as examining the possible current value with or without ties to the nearest previous value, and ensuring that if the current value is to be tied to the previous, values exist for the next resulting position that don't need to be tied to another value to be within the desired range. This part of the algorithm also creates a list of which values are included in each tuplet, for easy conversion to LilyPond code later on. However, this poses a particular challenge in editing rhythmic material after the initial generation, as is the case with the solo flute part in this movement.

Once the entire desired length in measures/whole notes is filled with rhythmic values, these are all set to *"s"* for sound by default. Some of these will be converted into *"t"* and *"tt"* (tie and tied to), or *"r"* (rest), ensuring correct durations, and respecting the set maximum amount of rests (% of the entire length of the rhythmic line). A random conversion into rests where allowed, up to and including the limit, and respecting values that need to be tied as well as allowing for unlimited consecutive rests, will easily result in a very uneven rest distribution, especially when the total duration of a line is quite long. Typically this will provide a fairly low density of sounding values at the beginning, increasing quite abruptly when the amount of rests allowed have all been distributed. Because of this, an extra rest distribution algorithm is necessary, which only allows for a small part of the rests to be distributed at first. The algorithm then goes backwards from the end of the line, using a system of quotas per whole note/measure. This ensures a more even distribution, but still showing a tendency of lower density early on and somewhat higher density later in the line. Ties are *not* broken by this algorithm, which means that only stand-alone rhythmic values ("s") may be converted into rests. If the parameter *pickrestvals* is set to *true*, the algorithm will only pick a subset of basic values for possible

conversion into rests, unless the value is too short and not tied to another value. The following is an overview of the initial parameters currently used for the basic rhythm generation algorithm:

len	Length in measures/whole notes. Integer; 1+
minrv	Shortest duration (also incl. ties). Rational; 1/32 up to 2/1
maxrv	Longest duration (also incl. tied). Rational; 1/32 tup to 2/1. Must be larger than or equal to minrv !
incltup	Include tuplets. Boolean; true/false
maxties	Maximum number of ties per duration. Integer; 0 or 1 (2+ not in use)
restmax	Maximum amount of rests. Float; 0.0 up to 1.0
pickrestvals	Pick a subset of basic values for rests. Boolean; true/false
maxstvals	Maximum basic vals for initial matching. Integer; 1 up to 21
distrest	Make use of rest distribution algorithm. Boolean; true/false

Table 3-1: Parameters initializing the basic rhythm generation algorithm

The part of the algorithm that distributes rests and ties goes through all the values from beginning to end (all initially set to "s"), always basing its rules on what type the previous value was. The simplest of these set of rules is the rule for when the previous value was set to "t"; this simply gives the current value the label "tt" (for tied to). This requires that the previous round examined the current value as well to make sure the resulting duration would not be too long, and checking whether a tie (or rest) is necessary due to one of the two being too short, or if a tie is optional. In that sense, when choosing whether to keep a value as "s" or converting it into a "t" and thus a "tt" for the next one, it examines both the preceding and the next value. Initially consequtive rests were not allowed, but I decided to change this for a more dynamic rhythmic flow. A challenge with this approach, however, is that rest clusters are sometimes generated, where rest values that should be represented with just a few symbols end up being represented as

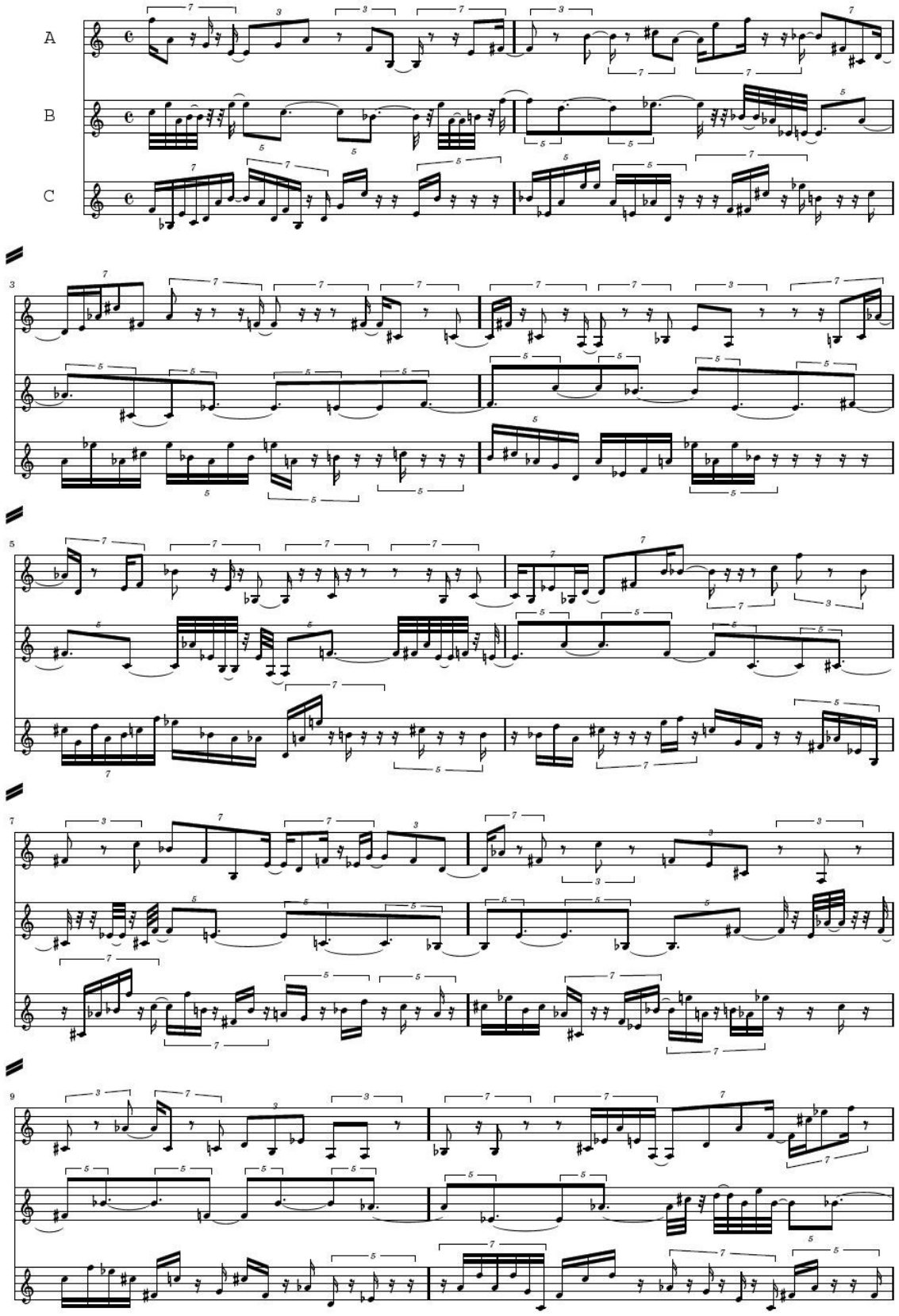
a longer row of short rests throughout the entire process. The final LilyPond output requires some additional editing because of this, but it also provides an interesting insight into how the algorithm works, as the conversion process is so clearly evident in the end result.

3.2.3 Case study: rh_out412

The actual Ruby *object* generated by the basic rhythm generation algorithm contains, in addition to the values and their types (r, s, t, tt), and LilyPond notation symbols, an extensive analysis of the generated rhythm and its general properties in terms of durations, density, amount of rests, relationship to the 1/4 beat, usage of basic values, etc. A similar analysis may be carried out for several rhythmic parts. To get a general idea of this analysis both on the level of the individual lines and the combined analysis, we can look closer at one of the sample output files, *rh_out412.* Three files are generated for these test outputs: The LilyPond file (rh_out412.ly), the file containing information about each individual line (rh_out412.txt), and the file containing the information about the combined rhythmic lines (rh_out412_combinfo.txt). LilyPond generates a PDF file, a MIDI file and a log file containing information about the conversion including any errors encountered. In total six files are generated.

Rh_out412 consists of three parts. Each part is 20 measures/whole notes long. The initial parameters are the same for all three lines, except for the *minrv* and the *maxrv* values. The common parameters are *len: 20, incltup: true, maxties: 1, restmax: 0.3, pickrestvals: false, maxstvals: 3, distrest: true*, while the minumum and maximum durations for each part are:
A) minrv: 1/32, maxrv: 1/8

B) minrv: 1/32, maxrv: 1/2 *C) minrv: 1/32, maxrv: 1/12*


A
B
C

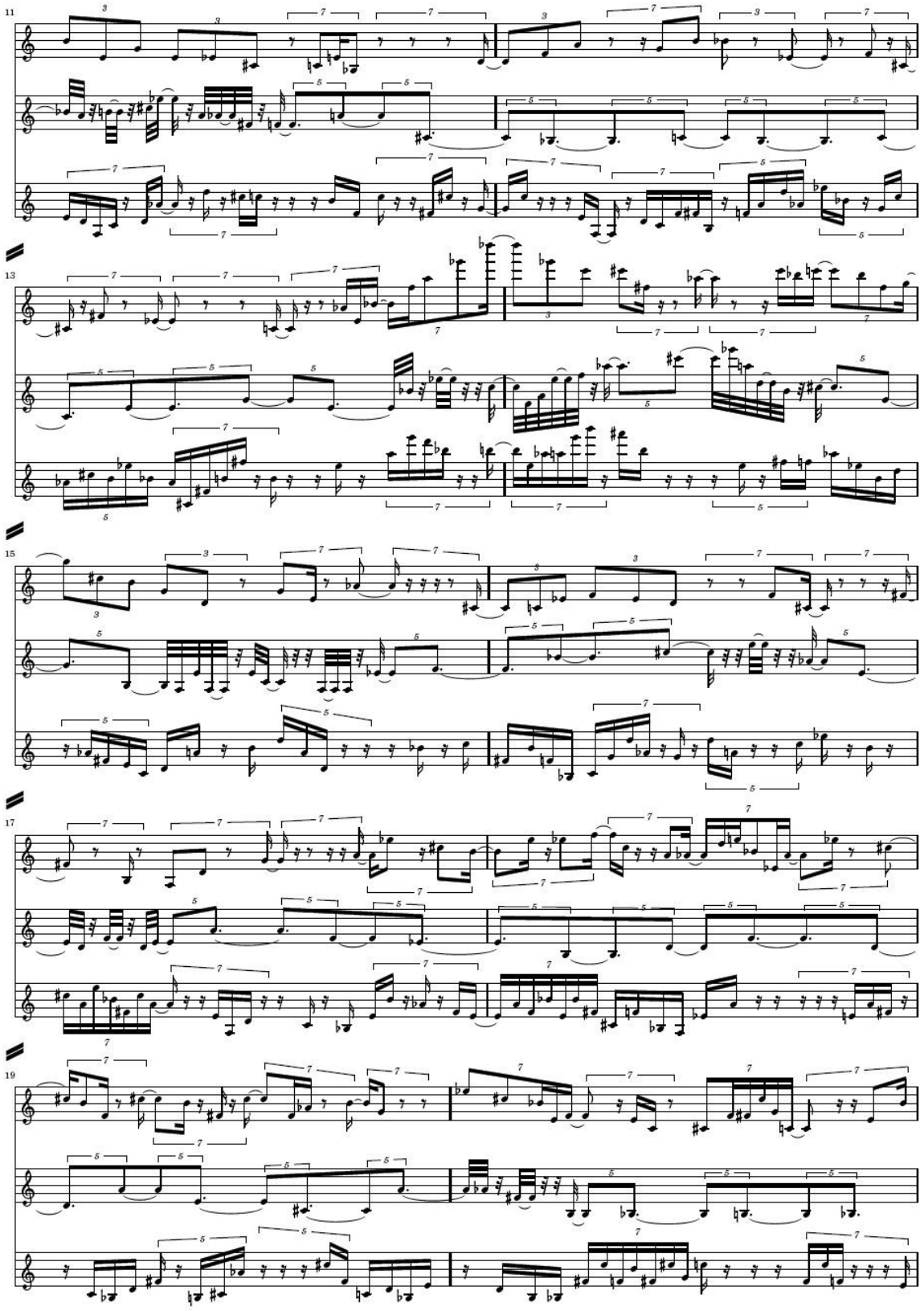


Figure 3-5: Example of three part rhythm output, rh_out412

All three have the same *minrv* value, but different *maxrv* values, and all the other initial parameters are identical. They are all generated with a maximum starting value count of 3. The first line uses values *1/12, 1/14, and 1/28*. The second uses *1/32, 1/10, and 3/20*, while the third line ends up using *1/28, 1/20, and 1/16*. All basic values for all parts are within the durational ranges given, so no positions have been deleted from the original position lists. This provides us with quite a wide palette even with just three basic values per part. Part B ends up with more tied durations than the others due to the longer *maxrv* value. Notice that because of this, part B also contains a lesser amount of rests than the two other parts. A contains approximately 5.87 whole notes of rests, B contains approximately 1.15 whole notes of rests, and C contains close to 6 whole notes of rests in total (6 whole notes is the very maximum allowed, as 20 * 0.3 = 6). A brief look at the analysis file gives us an overview of all three parts individually:

	Part A	Part B		Part C
Durations possible (incl. ties)	1/28 1/14 1/12 3/28 5/42	1/32 1/16 1/10 21/160 3/20	29/160 1/5 1/4 3/10	1/28 1/20 1/16 1/14
Count basic values (sounding)	1/28: 114 1/14: 79 1/12 :53	1/32: 107 1/10: 62 3/20: 62		1/28: 145 1/20: 78 1/16: 79
Real min dur.	1/28	1/32		1/28
Real max dur.	5/42	3/10		1/14
Count actual durations (sounding)	1/28: 51 1/14: 73 1/12: 41 3/28: 21 5/42: 12	1/32: 36 1/16: 21 21/160: 15 3/20: 1 29/160: 14	1/5: 12 1/4: 23 3/10: 12	1/28: 127 1/20: 78 1/16: 79 1/14: 9
Density	9.9 average	6.7 average		14.65 average
Presence	Ca. 0.706	Ca. 0.942		Ca. 0.7
Beat relation types (notable observations)	Predominantly 7-tuplet offbeats ending before beat or extending over to next beat	Predominantly 5-tuplet Offbeats extending over to next beat and 32nd based values ending before beat. Only *one* on-beat attack!		Predominantly 5-tuplet and 7-tuplet values ending before beat, but also 50(!) on-beat attacks and 41 attacks off-beat by 16th based values. Very few durations extending over to next beat!
Tuplet types	3, 7	5		5, 7

Table 3-2: Comparison of the three lines in rh_out412

Density here is measured as average number of attacks per whole note. The presence value subtracts the rests from the total and gives a 0.0 to 1.0 value for the amount of sounding notes. It is interesting to note that only part B actually contains the shortest possible duration (minrv) of 1/32, and none of the parts reach their maximum possible duration (maxrv) of 1/8, 1/2, and 1/12 respectively. This is partly due to the limited number of basic values (3 per part), and their resulting durations when tied. If more basic values were available it is very likely that 20 measures would contain at least one instance of both the shortest and longest possible duration

for each part, at least if the minrv and maxrv parameters are set to values that can actually be obtained using basic values with maximum one tie. The reason why part B has more possible resulting durations than the other two parts, is that the durational range is much wider (up to 1/2). So even with only three basic values, nine different durations are possible with the use of up to one tie per duration. Part C is the one with the definitively highest density. Interestingly, it is also the part with the lowest presence/fill (i.e, the most rests in total). This combination happens because of a high number of short stand-alone 1/28 and 1/20 durations, which in turn is a result of an overall set of basic values that are shorter than for the other parts combined with a narrower durational range (1/32 up to 1/12). In fact, the only viable tied value for part C would be two times 1/28, resulting in a 1/14 duration. As we can see, there is only nine of those in 20 measures. Even combining the two smallest values of 1/28 and 1/20 results in a value longer than the set maxrv of 1/12. The inclusion of 1/24 may have resulted in more tied values for part C.

The *beat relation* types reflect the relationship of each individual attack to the ¼ beat, including both its starting and end points. There are seven types of starting points and four types of ending points. The starting points are on the beat (o), half beat (h), 16th note-based (4), 32nd note-based (8), triplet-based (3), quintuplet-based (5), and septuplet-based (7). These reflect the placement of the attack within the beat. The four types of endings are completing the current beat (E), ending within the beat without completing it (W), extending over to next beat or further without completing the last beat (G), and extending over to the next beat or further, completing the last beat (C). The beat relation analysis of the first measure of part A looks like this:
oW, 7W, 7W, 7G, 3W, 3E, 3W, 3G, 7W, 7G.

At its current state, the program does not make use of this analysis for generative purposes, but this is a future possibility. In addition it is a useful analytical tool and source of statistical

information about a specific rhythmic line, or even an entire work. In the current example of rh_out412, it becomes clear when analyzing this information that even though there are many similarities between the three parts, and they function well together (also because of pitch relationships, which I will discuss later), there are also clear differences. One of these are the on-beat attacks, which C has many of (50), A has some of (22, on average a little over 1 per measure), and B only has one of (the very first attack in the first measure).

The analysis file for the three lines combined provides additional information. The space filled with rests in all three parts simultaneously is a mere 1/12, with two short points in measure 8 and one in measure 15. The most prevalent beat relation type is 7W, and the 7-group as a whole covers 230 of the total attacks (at simultaneous attack points, the longest value decides the ending type), followed by the 5-group at 111. Looking at the o-group (on-beat attacks), the number is 59 (the third most prevalent group). This number is lower than the combined number is on-beat attacks for each part (73). The reason for this is that several of the on-beat attacks happen simultaneously in two or more parts, and these are only counted as one attack in this collective analysis. The number of simultaneous attacks in at least 2 parts is 64, and the collective density is an average of 28 attacks per whole note. This high number (approaching 32, which for a single line is the highest possible density using the 32^{nd} note as the smallest basic value) is due to the differing rhythms in all three parts, as well as many short durations in general. The duration of 1/28 is by far the one occuring most frequently with 178 attacks among the three parts. The average space between attacks is also 1/28, while there is a total number of 55 unique lengths of spaces from one attack to the next, the shortest being 1/224, and the longest being 29/140 (slightly longer than 1/5). It is important to note that space between attacks represents something other than the durations of sounding events, as it takes into account any

rests that occur between attack points. This means that while the duration of the second note of part A is 1/14, the space between that attack and the next is 3/28 (1/14 sounding and 1/28 rest). This is also used to calculate, on a note-to-note level, the density per attack, as the finest density curve for each rhythmic line, or a combination of 2 or more. A 1/28 duration is short, but if followed by 6/28 of rests before the next attack, the density of the 1/28 attack is not 28, but 4 (measured as attacks per whole note).

The overall presence vertically (average of all three parts) is about 78%, while the 1/12 collective rest duration gives a horizontal presence of approximately 99.6%. The distinction between these two numbers is important, as they signify sound versus silence in each part and sound versus silence in all parts respectively. The combined analysis is useful as an additional tool when combining 2 or more rhythmic lines into one musical layer, or when attempting to create layers with contrasting rhythmic characters.

3.3 Concepts of pitch

Up until this point, I have chosen to focus my pitch algorithm around sets of intervals applied to two or more parts. I have many ideas for further development of algorithms for pitch, but I chose to focus on interval sets because I have previously explored this concept for individual horizontal lines, but not in multi-part textures.

Chromatic pitches are numbered 0-96, the lowest being C0 (four octaves below middle C), and the highest being C8 (four octaves above middle C), giving a total possible range of 8 octaves. Quarter tones may be used, and have numbers 0.5-95.5. These have not been implemented in the

pitch algorithm thus far, but the intention is to expand to include quarter tones in the future. LilyPond also handles these well. Enharmonics are not used, and any respelling of notes must be done manually at post-generation score editing.

Below is the first measure of *rh_out873* (the rh_out files were at first only used for rhythm with random pitch, but later on rhythm and pitch were both algorithmically generated and used for the rh_out files), showing the use of the interval set *[4,12,13,14]* (Major 3rd, Octave, minor 9th, Major 9th). A few measures further in, changes occur to the basic interval set of rh_out873, and the permitted pitch range is altered. We will take a closer look at this, as well as the procedures behind these changes.

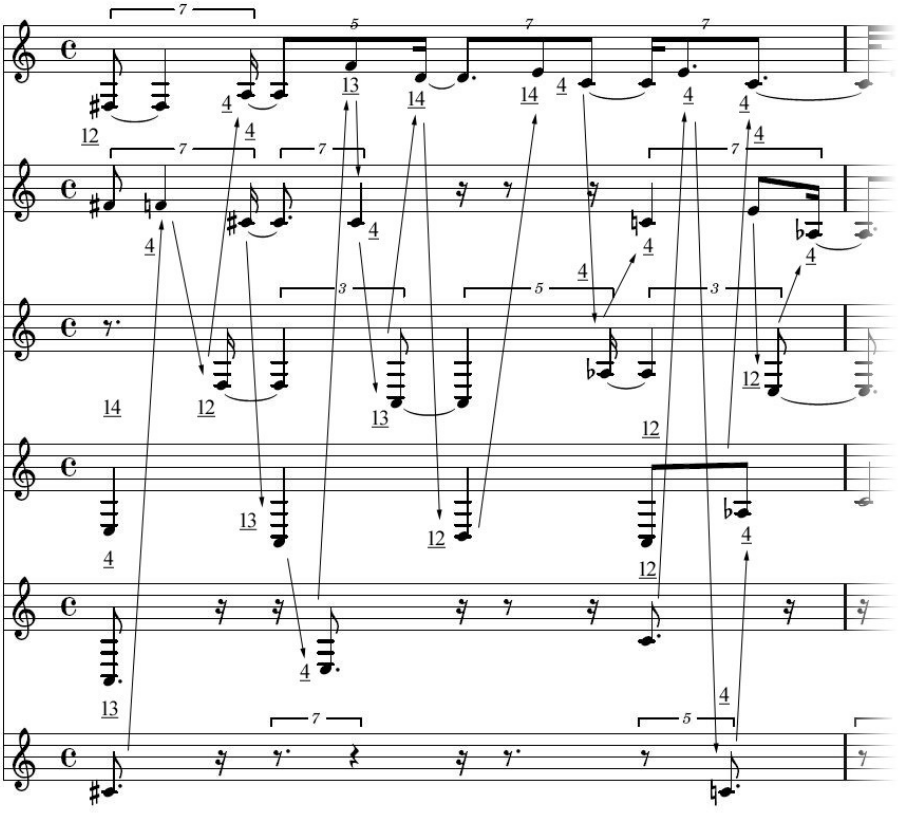


Figure 3-6: First measure of rh_out873 showing the intervals used from one attack to the next (simultaneous attacks treated the same as horizontally consecutive attacks, starting with the uppermost part)

Here we see the basic procedure of the interval set algorithm. Basically, multiple parts are treated as *one* horizontal line with the pitch of each new attack relating to the previous by use of one of the intervals in the set. When simultaneous attacks between two or more parts occur, the same principle is employed from top to bottom. The very first attack of this measure is shared between 5 of the 6 parts, the intervals between them being 12, 14, 4, and 13 respectively. Whether an interval is used in ascending or descending motion is not controlled directly, but is partly a result of other parameters such as the overall range, the range of the individual parts, and the maximum interval from one pitch to the next within a part. In addition, there is a weighted probability with regards to which pitch is chosen, based on the amount of possible pitches a particular choice opens up for one step ahead. The Ruby method that generates the pitch rows for each part along with an analysis, takes the following parameters:

setlen	Number of intervals in the initial set
basicrange	The desired starting range
onsetlist	Chronological list of attacks containing part numbers (from 0 up)
numparts	Number of parts
maxinternalint	Largest interval permitted from one note to the next within a part
smint	Lower limit for size of intervals in initial set
lint	Upper limit for size of intervals in initial set
invinset	Allow inversions in initial set (true/false)
extrange	The lower and upper limits of range alterations (or false if not used)
setchange	Allow alterations to initial interval set (true/false)
aqt	Allow quarter tone-based intervals/pitches (currently not in use)
forceset	Force an initial interval set (or false if not used)
lineranges	Ranges for individual parts/instruments

Table 3-2: Initial parameters for the multi-part interval set algorithm

The *setlen* parameter can in theory be set to any number between 1 and 16, but both of the extremes are problematic; only one interval may make it hard to find solutions, depending on other parameters, and too many intervals defies the purpose of the algorithm, which is to ensure a certain character within the pitch realm. If setlen is higher than the *lint* minus *smint*, it is automatically reduced. The basic range may be of any size initially, but will be adjusted to fit the individual ranges of each part, as well as a set minimum range interval, which in the case of larger interval sizes will depend on the intervals, and otherwise on a parameter which may be altered. In cases with many parts with differing ranges, however, this parameter should not be set too low, as this may make it difficult to find solutions. In some cases, only one interval is possible within a narrow range, and this creates a point of stasis jumping between two pitches. The *maxinternalint* parameter controls the largest interval permitted between two consecutive pitches within an individual part. If this interval is set too low, it limits the possible choices drastically, and the algorithm may need to run several times to find a solution. In general this parameter should be set with the agility of the instruments in questions in mind. At this point, it is one single value, but an option in the future would be to allow it to be a list, so that each part may have a different interval limit. If a flute and a double bass are in the same group, they are now both limited by the agility settings of the double bass, which is set to "low," while the flute's agility is set to "high." The details of the different agility categories will be covered in the chapter about orchestration. If the *forceset* parameter is an interval set, this is used as the initial set, no matter what other parameters are given for generation of an interval set. If *setchange* is set to true, changes may be made to the initial set even when a particular set is forced.

The possible changes made to interval sets and ranges (if *extrange* is given) are as follows:

1) Expanding the range upwards
2) Expanding the range downwards
3) Contracting the range from the top
4) Contracting the range from the bottom
5) Altering the set one interval at a time by:
    a) Inverting, for example replacing 3 (m3) with 9 (M6)
    b) Increasing by one (from 3 to 4)
    c) Decreasing by one (from 3 to 2)
    d) Removing an interval from the set
    e) Picking an interval with a higher dissonance level (see below)
    f) Picking an interval with a lower dissonance level (see below)
    g) Adding an octave (3 becomes 15)
    h) Subracting an octave (15 becomes 3)

The dissonance level mentioned for 5d and 5e is a value between 0 and 3 given to each interval from minor second up to Major tenth. The value of 0 is given to the perfect fourth, perfect fifth and octave, 1 to the thirds and sixth, 2 to the Major second, minor seventh, Major ninth, minor and Major tenth, and 3 to minor second, tritone and Major seventh.

If no changes are allowed, the music will have a more or less static character, depending on the initial range and the initial interval set, whether forced or randomly generated based on the initial parameters provided. Changes to the active range occur gradually, one semitone at a time. Intervals are altered one at a time. If changes to the range, interval set or both are allowed, the first change point is randomly placed somewhere between 10% and 20% of the length of the list

of attacks (which includes any simultaneous attacks). Thus, if the list contains 100 attacks, the first point of change will be between the 10th and the 20th attack. Any subsequent points of change add another 5-20% of the entire length of attacks. Note that the attack point list does not directly correspond to any rhythmic of temporal unit. It is, however, easy to convert the points of change into points along the temporal axis. The approximate spacing of the points of change (or initiation of change in the case of range, which changes gradually), may be altered or more strictly controlled if necessary. In the example of Figure 3-6 (rh_out873), with its initial interval set of [4, 12, 13, 14] and range of B2 to F#4, the changes made are as follows:

1) Range floor rising from 35 (B2) up to 41 (F3) between attack number 125 and 135, which on the temporal axis corresponds to 17/4 and 19/4, respectively, meaning the change takes place during the first three beats of measure 5.
2) Set change from the original [4, 12, 13, 14] to [1, 4, 12, 14]. Here an octave has been subtracted from 13, replacing it with a minor second, which immediately changes the character of the music. This change takes place at 145/16, which is shortly after nine measures have passed.
3) Shortly before the end of the 12th measure, another change to the set occurs. We now go from the established [1, 4, 12, 14] to [2, 4, 12, 14]. Here, the minor second has been replaced by a Major second, abruptly changing the character from somewhat dissonant to more consonant. Note that 2 and 14 are an octave apart, and that both 2 and 4 are part of the whole tone scale, which gives a very particular character. No other changes to the interval set occur in rh_out873.
4) Range ceiling rising, from 54 (F#4) to 80 (Ab6) within about a measure and a half, from 289/20 up to 255/16 (ca. 14.45 to ca. 15.9). The floor is still at 41, so even though the

change in range is definitely audible, it is more widened than pushed upwards, until the fifth and final change.

5) Between ca. 17.6 (a little over halfway through the 17th measure) and ca. 18.4, the range floor rises from 41 (F3) to 56 (Ab4). This means that for the final measure and a half of the 20 measure long example, the boundaries of the range have changed drastically since measure 1, and does now cover two octaves from Ab4 to Ab6.

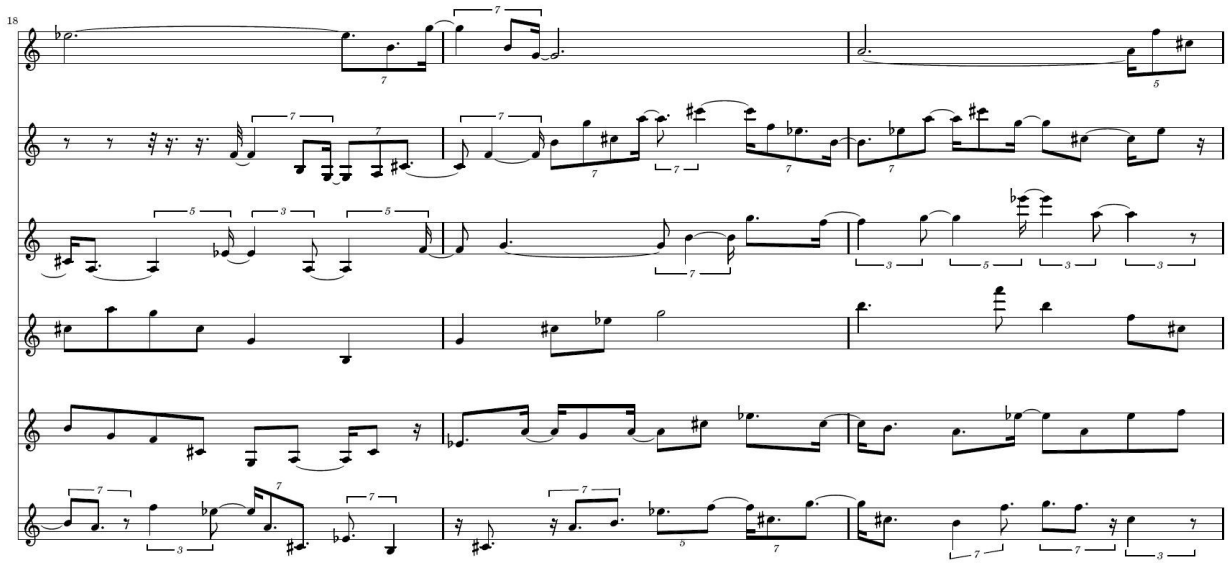


Figure 3-7: Final three measures of rh_out873, the range being pushed up to Ab4-Ab6, and the interval set having been converted from the initial [4, 12, 13, 14] to [2, 4, 12, 14]

Additional information obtained from the analysis file shows that all chromatic pitch classes are used, but Ab/G# (always notated Ab) only occurs nine times, while C#/Db (always notated C#) occurs 83 times. The focus range is from 41 to 54, while the lowest to highest actual sounding range is from 35 to 77, meaning that the three highest pitches in the overall range are never played. The average pitch is 49. The most frequently used interval is the Major 3rd (4 semitones), which occurs 357 times in the attack point-based row of pitches.

As the inclusion of different instruments and their respective ranges became an inherent part of the algorithm, the expansion of basic intervals by octaves became necessary. As a last resort, inversions of each interval in the set are added if no other options are viable, including expansion by octaves to cover a wider range and any range disparities between instruments. When examining the options for any pitch, the algorithm looks *backwards* one step globally to create a list of options using the interval set, narrows this down using both the overall range at that point as well as the range of the part in questions. Furthermore, the options a narrowed down based on the maximum interval within one part. The final factor is the possible options at the subsequent step, which means looking one step *ahead* in the attack point list, and the previous pitch for that particular part, as well as its individual range. Once this is done, a list is created for the current part, where each pitch giving at least one possible pitch for the next step, is represented x number of times, where x is the number of possible pitches for the next part using that specific pitch. The list is then shuffled, and a random choice is made. Pitches that will result in more possible choices for the next step are thus more likely to be chosen.

3.4 Orchestration and timbre

According to the formal principles used for this movement, outlined below, orchestration is largely a matter of grouping instruments together to form separate layers. Instruments are combined based on a matching algorithm where points are given for each of the following criteria: timbre (keywords), range, loudness, agility and type. The first instrument is chosen randomly from a list of available instruments. This list excludes instruments already used within the formal section in question. A range overlap of an octave or more adds 5 points, matching

timbre keywords (*dark, bright, noisy, etc.)* awards up to 3 points, an exact match of relative loudness gives 2 points, etc. A strong mismatch may also result in negative points for one or more parameters. This procedure ensures that instruments that are grouped together will have commonalities in one or more areas, while at the same time giving room for more flexible groupings than simply dividing the orchestra by family alone.

Timbre is described generally per instrument, and additionally for many playing techniques (not fully implemented for this movement), mallets/beaters, mutes, etc. Keywords such as *dark*, *reedy*, *very bright*, *eerie*, *hollow*, and *noisy* are used as descriptions of timbral qualities. These keywords are to a certain degree subjective, and a list of keywords like this would never fully capture the subtleties of instrumental timbral qualities. It does, however, give the composer/programmer a way of describing the aural experience of a certain instrument or playing technique. Keywords are also entered manually into two lists ordered by brightness and noise level, depending on what the keyword describes. "Bright" does not necessarily say anything about the noise level, but keywords such as "mellow" and "distorted" would be placed on such a list. This type of ordering is perhaps more relevant when combining different playing techniques on different instruments than for grouping instruments together based on their general timbre. In addition to timbre keywords, each playing technique is also connected to others through similarity. Thus, tremolo and flutter tongue are listed as similar, as are glissando and pitch bend. Pizzicato and Bartók pizzicato are both linked to slap tongue because of the percussive quality of attack they all embody. Playing techniques are not fully implemented within the program at this stage, but the foundation is laid to include this in the future. For the first movement of the flute concerto, a short list of playing techniques is allowed. A layer may

have a small selection of playing techniques, but these must be distributed by the composer at post-generation score editing at this point.

Agility is an important factor for grouping instruments together. It also limits the *minrv* (shortest duration) parameter for rhythm generation based on the tempi used in specific sections, and the maximum interval per part for pitch generation. There are four agility categories, from low to high. *Low* has a maximum note speed of 240 bpm (16^{th} note at quarter note = 60 bpm), while *high* has a maximum note speed of 720 bpm (16^{th} sextuplet note at quarter note = 120 bpm). For the *low* category the maximum jump at a relatively slow speed (half of maximum speed) is set to 10 (minor seventh), and above this it is set to 8 (minor sixth). For the *high* category these limits are set to 16 (Major tenth) and 12 (octave) respectively. Setting the interval limits for the *low* category too low may cause problems when distributing pitches. Higher values may, on the other hand, produce parts that are more difficult to play, and may require more manual post-generation editing by the composer.

3.5 Articulation and dynamics

Dynamics range from *ppp* to *fff*, and are represented by numbers *0* to *7*. For each layer in a section a lowest value between 0 and 4 is chosen randomly. Secondly, a maximum value between the lowest and the lowest plus 3 is chosen, and a base value picked within that range. This ensures that a layer doesn't fluctuate too much dynamically. In addition to this, a final step controls the distribution of dynamic markings within each part to avoid changing dynamics for every note. A certain amount of post-generation score editing is to be expected at the program's

current state although the instrument grouping algorithm does minimize the need for this to some extent. Crescendi and diminuendi are set to true or false per layer, and distributed per part.

Articulation types are stored in a list containing information about the usage of each type. Staccato, for example, is listed as a "shortener" (shortening the actual duration); thus, it is not suitable for tied values or for longer values beyond the quarter note, etc. There are 11 articulation types, ranging from legato to martellato-staccato (short martellato attack). For each form section, the full list of articulations is avaiable, and articulation types are distributed per layer, contributing to the uniqueness of its character. Automatic distribution of slurs and bowings is only done for layers whose articulation types include legato. Care must be taken, therefore, to take a second look at the generated score with regards to the technicalities of each instrument, as well as the general phrasing and articulation of each part.

3.6 Form, structure, and tempi

The form of the first movement of the flute concerto is based on sections of varying length. How many sections a movement or work is divided into, is chosen randomly within a range set by the composer. These sections are either *short* sections where a selection of the available instruments have *homorhythmic* material for a few measures, or *longer* sections where the orchestra is divided into different groups that are given different material. The program first calculates tempi based on a desired duration of *n seconds*. The desired duration in the score generated for this thesis is ten minutes, or *600 seconds,* chosen by the composer. Each tempo is either chosen randomly from a pre-defined list comprised of five different bpm values (40, 60, 80, 90, 104), or

calculated as a metric modulation from one rhythmic value in the previous tempo to another value in the following tempo. Only metric modulations resulting in whole number bpm values are allowed. A measure length of minimum 10 and maximum 40 measures is given for each tempo. This continues until the total duration is between 95% and 105% (set manually by the composer) of the desired duration.

Once the tempi are calculated, the total number of measures is divided into 5-8 long (layer-based) sections, and 4-9 shorter (homorhythmic) sections. Each short section is set to be between 2 and 6 measures long although this range may be altered. The total measure count for these sections are then subtracted from the total. The remaining measures are divided by the number of longer sections, and a random number of measures between 0 and a third of the measures allocated to each section is then either added to or subtracted from the equal division. This provides us with sections of *unequal lengths* in measures, with changes in tempo often occurring within a single section rather than coinciding with the beginnings and endings of sections.

Longer sections are divided into layers, ensuring that there is some overlap, and that at least one layer begins at the first measure of the section, and at least one layer lasts until the last section of the measure. A very high likelihood of layers overlapping by several measures is built into the layer algorithm, as the possible starting points for a particular layer depend on the already calculated starting point of another layer.

A layer can have one of three different *roles*. It either serves to *influence* another layer in terms of specific parameters relating to rhythm, pitch, dynamics or articulation (timbre and playing techniques will be included in the future), is *influenced by* at least one other layer, or is independent with an *internal development* of pitch (interval sets), register, and possibly rhythm (in cases where a tempo change occurs in the part of a section where such a layer is present). For

layers that are influenced by at least one other layer, the rhythm and pitch generation is executed twice even if there is no change in tempo. In cases where pitch is influenced, the interval set or the layer exerting influence is copied and expanded to include one extra interval, for use in the second section of the influenced layer. Range influence involves copying the range of the influencer, and customizing it to fit the influenced layer. Rhythmic influence includes a change in the overall density, *minrv/maxrv* values (shortest and longest actual duration), or inclusion of basic values present in the influencer and not in the influenced layer. All *minrv* values are adjusted to the maximum bpm value for the instrument(s) with the lowest agility setting within a layer, depending on the tempo in question. Every rhythmic line is stored as a part of the global *$rhythmobjects* Ruby object. For each *layer*, rhythmic lines are generated within a limited range of values for the initial parameters used for basic rhythm construction (see Table 3-1 for overview of these parameters). This ensures a relatively defined, but flexible character. The resulting rhythmic lines are grouped together as one "generation" of rhythms. Each rhythm object with all its analytical information is also written to a text file for reference. Form information, including details on each section, is saved in a separate file. To get a more accurate and complete picture of the results of the form algorithm, we will take a closer look at the actual generated score, along with its sections and layers, in chapter 4.

3.7 The role of the solo flute

The solo flute in this movement is active in all sections except the shorter homorhythmic ones. At the beginning and end of each section it has between one and four measures of rest. The material for the flute part is generated through a process of copying and alteration of material

created for the other parts. Rhythm is borrowed globally, which means that for every section generated, the rhythmic possibilities for the flute part are expanded. The pitch material is borrowed from the generated horizontal lines within a section, whether it be for single parts or an entire layer. Rhythms are copied per whole measures, and altered through conversion of rest to sound (and vice versa), and the splitting of ties. The values themselves and their placements within the measure are not altered. Rhythms are chosen that have relatively short *minrv* values to keep the solo part active and audible. Pitches are also generally transposed one or multiple octaves up, where necessary, to fit the instrument and to highlight its more prominent register.

The pitch material is copied in chunks of up to ten pitches at a time. This basic cell is either kept as is and superimposed onto the rhythms already generated for the flute part, or altered through one of the following methods:

1) Sequenced 3-5 times using intervals m2 up to P4
2) Filtered by deleting every other pitch
3) Repetition of 2-4 notes between 2-4 times
4) Extension through the use of the intervals occuring between 3-4 pitches
5) Expanding and contracting intervals

Pitches are then transposed up one octave if all resulting pitches are within the range of the flute. If not, they remain the same, and any pitches that fall outside the flute's range are deleted.

4. The generated score and its formal sections

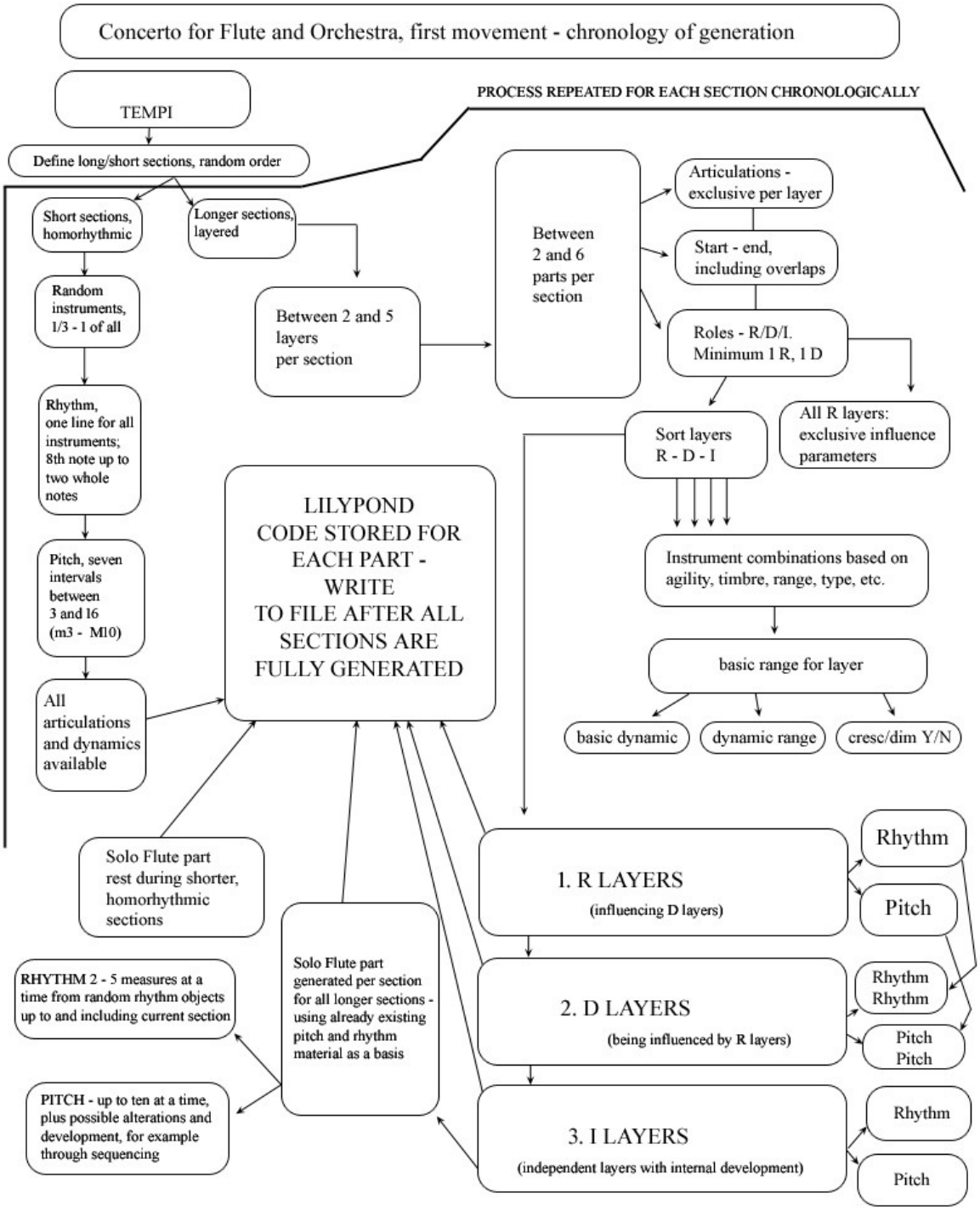


Figure 4-1: Chronology of generation

Figure 4-1 illustrates the chronology of music generation, explained in further detail below. Figure 4-2 shows the generated formal sections and tempi of the score, along with an overview of layer activity and roles, which was generated for this thesis.

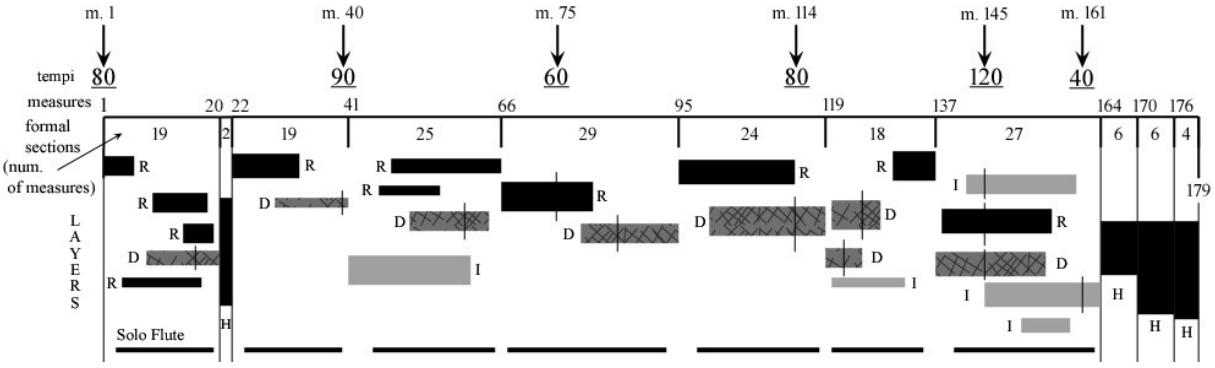


Figure 4-2: Overview of formal sections, layers, and tempi

Layers marked *I* are independent layers with internal development, *R* denotes layers influencing other layers, and the layers marked *D* are those influenced. Homorhythmic sections are shorter than the others and marked with an *H*. There are seven longer sections (of 19-29 measures in length), and four shorter sections (of 2-6 measures), making eleven in total. Three of the shorter (homorhythmic) sections are at the very end, which creates a longer homorhythmic part of 16 measures. The order of sections is randomized, so the placement of the three final sections is purely coincidental. The thickness of each layer represents the number of instruments playing relative to the other layers in a section, and in the movement as a whole. The solo flute part is active in all the longer sections, except for the very beginning and end of each section (between one and four measures of rest on each end of any section, following the same procedure concerning whole mesure rests as each layer in a section; x measures of rests before an entry point, y measures at the end of a section). The built-in rests are based in both practical and aesthetic considerations; to allow the soloist a bit of rest, and to avoid an omnipresent solo part.

The quarter note tempi used are noted in bpm above the sections, with their respective measure numbers. A vertical line through a layer is used to show a) when influence begins for *D* layers, or b) when one rhythmic line ends and the next begins where a tempo change occurs within a section. In the third section (measures 22-40), a tempo change occurs at the very last measure (40), splitting the *D* layer into two parts of 11 and 1 measures, respectively. Thus, any influence from the only *R* layer in the same section lasts only one measure. To reach a better understanding of how the influence process works, we will take a closer look at the fourth section (measures 41 through 65), and how the *D* layer in that section is influenced by the two *R* layers.

The music for each *layer* is not generated in chronological order. All *R* layers are generated first, then the *D* layers, and finally the *I* layers. For this particular section, the *R* layer starting in measure 48 was the first to be generated (*fl2, cl, bscl*). This layer's influence concerns the *presence* (percentage filled, average all parts in a layer) and *maxrv* (maximum actual duration) parameters. The other *R* layer (*bsn1, bsn2*), starting in measure 46, will influence the *interval set*, *register*, and *minrv* of the *D* layer. Because this layer was generated after the one starting in measure 48, however, and they both have influence over the *minrv/maxrv* range, the *maxrv* influence of the flute and clarinet layer is transferred to the bassoon layer. This is to avoid mismatching values in cases where the *maxrv* of one layer is smaller than the *minrv* of the other. The *D* layer in the section enters in measure 51. The influence from the two other layers occurs from measure 60. The *D* layer thus continues for 4 measures with its new properties; longer shortest/longest durations, fewer rests, higher range, and a new interval set. The following table provides an overview of the influence parameters and their values, both at the influencing end and the influenced *D* layer before and after the change takes place.

***R* layer value**	minrv: 3/16 (shortest duration)	maxrv: 7/8 (longest duration)	presence: ca. 0.86 (sound vs. rest)	weighted pitch range: 27-48	int.set: 4, 9, 13, 16
***D* original value**	minrv: 3/32	maxrv: 1/2	presence: ca. 0.65	weighted pitch range: 23-29	int. set: 2, 3, 6, 10
***D* new value**	minrv: 1/6	maxrv: 6/5	presence: ca. 0.91	weighted pitch range: 30-44	int set: 4, 5, 9, 13, 16
Alteration	longer	longer	increased	higher	assimilation

Table 4-1: Layer influences with resulting values in section 4, with shortest duration, longest duration, average rest amount per part, range, and interval set altered.

The *presence* values here are averages of the presence values of all parts in each layer (actual amount of sound, compared to the amount of rests). Note that other properties of the *D* layer, while not directly influenced, may still be affected by the change. First of all, all *D* layers go through at least two rounds of rhythm generation, which means that the standard deviation for *minrv* and *maxrv* values may be deviated from even if these parameters are not directly influenced by other layers. In addition to this, as is evident in section 4, articulations may occur at a higher or lower rate than before simply because the durations are affected. In section 4, the typical short note articulations (staccatissimo, accented staccato) of the first part of the *D* layer become much less frequent as longer durations become more frequent from the point of change (see comparison below).

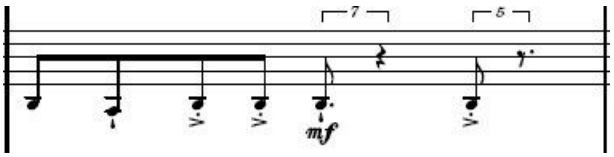


Figure 4-3: Cello 2, measure 58; staccatissimo and accented staccato frequently used

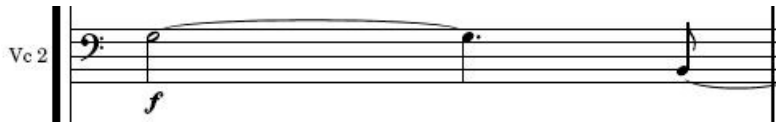


Figure 4-4: Cello 2, measure 60; longer values making staccatissimo and accented staccato appear less frequently, even though articulation types remain unaltered for this layer

The fact that the influences included in the program thus far are measurable and traceable through analysis, doesn't necessarily mean that they are immediately audible and perceptible in the music itself. This is also true for the differentiation between overlapping layers within a section. Measurably differing behaviours is not the same as perceptibly noticable differentiated layers of sound. This becomes clear in the last long section of this movement, where up to five layers coexist, and to some extent merge into a larger mass for some time.

5. Conclusion and future development

Concerto for Flute and Orchestra, first movement, is the first part of a three-movement work composed using algorithmic and computer-assisted composition techniques, programmed by the composer, using the programming language Ruby along with the notation software LilyPond. The work was commissioned by Belgian flutist Oriana Dierinck.

The algorithmic, conceptual, aesthetic and technical considerations of rhythm and pitch employed in this movement, and in the program as a whole, have been explained in detail. Other musical parameters such as articulation, dynamics and timbre have also been discussed, along with an outline of formal procedures and the role of the solo flute.

I am considering two options for the creation of the *second and third movements* of the flute concerto. The first option is to use the program in its current state to generate both of the other movements, displaying three different outputs of the exact same procedure. The other option is to generate only the third movement this way, and for the second movement make a selection from the vast library of output material the program has generated so far (see Figure 3-5), and shape it myself, as a way of re-integrating my own role as a composer in a more direct fashion.

A large project such as this program may never be completely finished. My main objective with this work is to formalize and automate the entire compositional process, within the confines of my own artistic and aesthetic visions. As much as I will continue developing my program, even beyond the second and third movements of the flute concerto, the experience and knowledge gained through this process will seep into my compositional practice also outside of the programming environment.

As much as the goal is to automate every parameter of music generation, some post-generation editing has been necessary in the score generated for this thesis, for example with regards to dynamics, slurs and bowings, etc. Instrumental techniques are not yet fully implemented, but a lot of the necessary information has already been organized in the *instruments* file (see chapter 3.1 for an overview of the different program files and their functions). Other challenges along the way have included coping with the vast number of instruments at hand, and their individual ranges and specific limitations, especially when combining them in groups forming separate

layers. Learning both a new programming language and the symbolic text formatting of LilyPond has also been quite time consuming and challenging, but ultimately this has given me a great deal of control and autonomy in the creative process of shaping my algorithmic procedures. I initially spent a lot of time searching for good solutions with regards to the representation and systematic programming of rhythm, respecting both the mathematical essence and the musical and notation-oriented reality of this particular parameter. Music is a time-based artform, and thus every composer is forced to relate to the flow and division of time. I believe the system of rhythm generation that I developed suits my needs, and is well-adjusted to the reality of notational conventions in music.

One of the main challenges for the future development of the program will be improving upon instrument-specific writing, and in the realm of timbre and instrumental techniques. Further developing this aspect will provide a wider palette of colours and possibilities. There is also a lot of work to be done with regards to dynamics and articulation, creating a more solid basis for the usage of these parameters, also with respect to notation and performance practicalities. I intend to include a wider array of pitch systems and procedures not necessarily based on interval sets, and further develop the rhythmic procedures to include other time signatures. The possibilities are practically endless, but I believe that one should take advantage of the incredible richness an algorithmic approach to composition offers, and spend time exploring the output of a particular procedure thoroughly, rather than swiftly moving on to the next one, just because one can.

The program at its current state only represents the beginning. In spite of its relatively simple principles for generating music, however, it is quite capable of producing output which in my view is both creative, original, interesting and well-shaped. It is music I would not have composed without the program, yet strangely familiar and very much my own.

Bibliography

- Agon, C. / Assayag, G. / Bresson, J. 2006. *The OM composer's book*. IRCAM
- Alsop, R. 1999. "Exploring the Self through algorithmic composition." Leonardo music journal vol. 9, pp. 89-94
- Anders, T. / Miranda, E.R. 2011. "Constraint programming systems for modeling music theories and composition." ACM Computing Surveys 43:4
- Berg, P. 2009. "Composing sound structures with rules." Contemporary music review 28:1
- Burton, A.R. 1999. "Generation of musical sequences with genetic algorithms." Computer music journal 23:4
- Collins, Nick. 2009. "Musical form and algorithmic composition." Contemporary music review 28:1
- Cope, D. 2005. *Computer models of musical creativity.* MIT Press
- Cope, D. 2000. *The algorithmic composer.* A-R Editions
- Cope, D. 1999. "Facing the Music: Perspectives on Machine-composed music." Leonardo Music Journal vol. 9, pp. 79-87
- Diaz-Jerez, G. 2011. "Composing with Melomics: Delving into the computational world for musical inspiration." Leonardo music journal vol. 21
- Edwards, M. 2011. "Algorithmic composition: computational thinking in music." Communications of the ACM 54:7
- Harley, J. 1995. "Generative processes in algorithmic composition: chaos and music." Leonardo Music Journal 28:3
- Hedelin, Fredrik. 2008. "Formalizing form: An alternative approach to algorithmic composition." Organised Sound 13(3), pp. 249-257, Cambridge Univ. Press
- Hoffman, P. 2002. "Towards an "automated art": Algorithmic processes in Xenakis' compositions." Contemporary Music Review, 21:2-3
- Kollias, P-A. 2011. "The self-organising work of music." Organised sound 16:2
- Kuuskankare, M. 2009. "ENP: A system for contemporary music notation." Contemporary music review 28:2
- Lindberg, Magnus. 1996. "Engine"
- Luque, S. 2009. "The stochastic synthesis of Iannis Xenakis." Leonardo Music Journal, vol. 9, pp. 77-84
- Manzolli, J. / Moroni, A. / Von Zuben, F. / Gudwin, R. 1999. "An evolutionary approach to algorithmic composition." Cambridge Univ. Press
- Miranda, E.R. 2011. *A-life for music: music and computer models of living systems.* A-R Editions
- Nakajima, T. 2004. "Synchronic and diachronic hierarchies of living systems." International Journal of General Systems 33:5
- Nieminen, Risto. “Programme note: Magnus Lindberg Engine (1996)”. Music Sales Classical. Accessed Apr-14-2015.

http://www.musicsalesclassical.com/composer/work/7693
- Nierhaus, G. 2010. *Algorithmic composition - Paradigms of automated music generation.* Wien and New York: Springer
- Papadopoulos, G. 1999. "AI Methods for Algorithmic composition." AISB Syposium on Musical creativity
- Rowe, R. 2001. *Machine musicianship.* MIT press
- Sandred, Ö. 2009. "Approaches to using rules as a composition methdos." Contemporary music review 28:2
- Schaathun, A. and Lemouton, S. 2005/6. "Knitting and Weaving: Using OpenMusic to generate canonic musical material." IRCAM
- Schaathun, A. 1988/89. "The computer as psychoanalyst - Thoughts on computer-aided composition."
- Schilingi, J.B. 2009. "Local and global control in computer-aided composition." Contemporary Music Review 28:2
- Shan, M-K. / Chiu, S-C. 2009/10. "Algorithmic compositions based on discovered musical patterns." Multimedia tools and applications 46:1
- Simoni, M.H. / Dennenberg, R.B. 2013. *Algorithmic composition: a guide to composing music with Nyquist.* Ann Arbor
- Tanzi, Dante. 1999. "The cultural role and communicative properties of scientifically derived compositional theories." Leonardo music journal vol 9, pp. 103-106
- Todd, P.M. / Loy, D.G. 1991/2003. *Music and connectionism.* MIT press
- Xenakis, I. 1992. *Formalized music - Thought and mathematics in music.* Pendragon Revised edition
- Zambelli, S. 2002/2004." Production of ideas by means of ideas: A Turing Machine metaphor." Metroeconomica 55:2-3

Concerto for Flute and Orchestra,

first movement

HUGO HARMENS

Composed with financial support from Komponistenes Vederlagsfond (Norway)

ORCHESTRA:

2 FL

2 OB

CL in Bb

BASS CL in Bb

2 BSN

4 HRN

2 TRP in C

2 TRB

1 TBA

PERC 1: VIBRAPHONE

PERC 2: MARIMBA, GLOCKENSPIEL

VLN 1 DIVISI (2)

VLN 2 DIVISI (2)

VLA DIVISI (2)

VC DIVISI (2)

D.BASS

SOLO FLUTE

Score in C

Duration: ca. 10 minutes

I

Hugo Harmens

♩ = 80

Flute 1
Flute 2
Oboe 1
Oboe 2
Clarinet in Bb
Bass Clarinet in Bb
Bassoon 1
Bassoon 2
Horn in F 1
Horn in F 3
Horn in F 2
Horn in F 4
Trumpet in C 1
Trumpet in C 2
Trombone 1
Trombone 2
Tuba
Percussion 1
Percussion 2
MARIMBA
Solo Flute
Violin 1a
Violin 1b
Violin 2a
sul pont.
Violin 2b
sul pont.
Viola 1
sul pont.
Viola 2
Cello 1
Cello 2
Double Bass

4

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
pp
mf
fff
ff
mp
f

7

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

ppp *pp* *f* *fff* *ff* 8va

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

ppp
mp
fff
f
pp
8va

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va

pp

mp

pp

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf
ppp
ppp
fff
ff
ppp
ppp

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

ff *mf* *8va* ord.

22

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

25

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp *mp* *f* *p* *mp* *pp* *mp*

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp *mf* *mp* *mf* *p* 8va 8va

29

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf *mp* *ff* *f* *ff* *fff* *mp* *p* *pp* *pp* *mf* *pp* *p* *p*

8va

32

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

fff mf fff ff mp pp 8va 8va

35

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

3 7 7 3 6 7 5

fff *mf* *mf* *mf*

38

♩ = 90

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

ff

fff

p

8va

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp
mp
mp
mp
mp
GLOCKENSPIEL
mp
f
mf
mf

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf *mp* *8va* *mf* *f* *ff* *mf* *f*

50

Fl 1
Fl 2
8va
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

pizz.
pizz.
8vb

Fl 1
Fl 2
8va
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf f mf mf f mf mf mf 8vb arco mf mp mf

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp
mf
8va
8vb
arco
fff

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

58

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
8vb
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

60

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

62

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

64

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

VIBRAPHONE

sul pont.

sul pont.

sul pont.

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp
f *mf* *f* *mf* *mp* *p*
sul pont.

70

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

p
ff
mf
f
mp

73

♩ = 60

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp *mf* *f* *fff* *p*

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mp *fff* *mf* *fff* *mp*

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf *ff* *mp* *fff* *f* *fff* *mp* *p* *mf* *mf* *mf* *mf*

8va

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
mf
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
mp
mf
ff
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
8va
Vc 1
Vc 2
DB

85

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

88

Fl 1

Fl 2

Ob 1

Ob 2

Cl

Bs Cl

Bsn 1

Bsn 2

Hrn 1

Hrn 3

Hrn 2

7

pp

5

Hrn 4

Trp 1

Trp 2

Trb 1

Trb 2

Tba

Perc. 1

Perc 2

S Fl

7 7 7 3

3 5 7

mf

7 7 7

mf

Vln 1a

Vln 1b

Vln 2a

Vln 2b

Vla 1

8va

Vla 2

Vc 1

Vc 2

3

mp

DB

91

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf
ff
fff
8va
mf

95

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

f *mf* *8va* ord.

98

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va

f

fff

mp

mf

100

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
mf
pp
mp
f
mf

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
mf
pp
mp
fff
f

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
mf
mp
f
pp
fff

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

mf
pp
fff
ff
8va

♩ = 80

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

pp *f* *ppp* *fff* *mf* *mp*

flz.

ord.

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

120

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

123

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
mp
fff
mf
ff
f
p

126

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
8va
8va
p
mf
ff
mp
fff
f
mf
p

129

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va
ff
mp
f

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
mp
mp
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
GLOCKENSPIEL
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

♩ = 120

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

♩ = 40

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

8va

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB

Fl 1
Fl 2
Ob 1
Ob 2
Cl
Bs Cl
Bsn 1
Bsn 2
Hrn 1
Hrn 3
Hrn 2
Hrn 4
Trp 1
Trp 2
Trb 1
Trb 2
Tba
Perc. 1
Perc 2
S Fl
Vln 1a
Vln 1b
Vln 2a
Vln 2b
Vla 1
Vla 2
Vc 1
Vc 2
DB