Constraints-Based Music Generation

Semester Project Report

Maëlle Lise Colussi
{firtname.lastname}@epfl.ch

Laboratory for Automated Reasoning and Analysis (LARA)
Responsible professor: Prof. Viktor Kuncak
Supervisor: Nada Amin - Programming Methods Laboratory (LAMP)
Dep. of Computer Science and Communication Science - EPFL

Abstract

This report summarizes the results of our project on constraints-based classical harmonization of melody in Scala. The framework used is the `MusicInterface` project ([Pittet 2014]). The program that results generates a harmonic progression from a melody in two ways: one way with formal constraints, solving them with the `CafeSat` constraints solver ([Blanc 2013]), the other way linearly.

1. Introduction

The problem of harmonization of melody using constraints has been widely studied (see for examples in this survey [Pachet and Roy 2001]). Difficulties of the problem come from the fact that constraints have to be met at different intricated levels: for the series of chord (harmonic progression), voice leading, also with taking into account the dissonances one could add to make effects, ... As an example of a constraints-based program for composing music there is Strasheela : it can create whole pieces of music (rythm, harmony, ...) given composer constraints. It is not linked to any type of harmony, which means more flexibility, but also is more complicated when one wants to use it.

For our program we chose to follow classical harmony (based on [Merriman 1997]). We decided to generate harmony taking into account rules for harmonic progression (series of chords) only (and not of concrete note placement). In fact, the harmonic progression gives the main color to the melody; adding constraints on concrete placement would mostly make the piece of music sound ”softer”, more subtile, so we decided, at least for this first program version, to restraint ourselves to harmonic progression.

The program can be found on a GitHub repository[1].

2. Implementation

The composer gives a sequence of notes that he wants to be harmonized. Also, he specifies which kind of cadence he wants at the end. If he does not specify any cadence, for the linear harmonizer, the default cadence (authentic cadence) will be forced, for the constraints-based harmonizer, a random possible chord for the end will be chosen (this assymmetry between the harmonizers will hopefully be corrected in a subsequent version).

The composer also specifies which harmonizer he wants to use, and can specify his own constraints on possible chords for specific notes. When there is a composer constraints on the chords on the end of the notes to harmonize, the constraints-based harmonizer does not take the cadence indication into account, only the added constraint.

Then, a list of possible chords is generated for each note to harmonize (a specific silent chord for silences). Possible chords are defined like that : in the set of all accepted chords considered in the program, are considered ”possible” the ones that contain the note. This is to avoid dissonances. Notes with accidentals have no possible chords (for some of them, this problem could be solved with a bigger grammar, but not all). In this case, the constraint-based harmonizer cannot find a solution. What the linear harmonizer does is explained in its detailed explanation part.

[1] https://github.com/MaelleC/MMusicScala

The problem is then solved with the chosen harmonizer. If no solution is found from the constraint-based one (if it was chosen), a warning is given and the linear one is used to guarantee a solution and to give diagnostic to help the composer to change the input to get a solution from the chosen harmonizer.

The grammar that is used is a simple one: it gives possible pairs of successive chords, and also possible chords for ends of cadences. Also, for the constraints-based harmonizer, if it can solve constraints with this simple grammar, it adds constraints on chords at bigger intervals than one (two and three, see below) and try to solve the problem as well. It returns this last solutions if it could solve, otherwise the previous solution.

The list of chords returned by the harmonizer is then converted in a lists of lists of 4 notes for each chord, (no need of 3 notes as accompaniment as usual, since no constraint on concrete note placement is considered) and then in a `ParallelSegment` (see [Pittet 2014]). Then the composer can combine it with the melody (all the notes, not only with the harmonized ones) and play it.

2.1 Linear harmonizer

The linear harmonizer begins with the last non-silent note (silent chords are given for the silent notes at the end if there exist some) and gives the chords from the end to the beginning of the list of notes to harmonize. This was done this way because it made more sense for the grammar to go in the reverse way, also to enforce the end, which is more important to be right than the beginning, harmonically.

At each step, the intersection of the possible chords of a note with the set of chords accepted by the grammar (knowing the "next" note) is taken, and a chord in the resulting set is randomly selected. If the note is a silent, the silent chord is given, and the harmony continues for the next step (previous note) as if there had been no silent (the pairs considered in the grammar can have silents in-between).

If the intersection is empty, a warning is given in each case that can cause this problem. If we are at the end, a random chord from the end-compatible chords is given (even if we will have dissonances). Otherwise, if the note has not any possible chords (note with accidentals), the last given chord is given this time, too. If the note has possible chords, a chord is randomly chosen from the set of possible chords, to avoid dissonance, and harmony continues from it (to avoid too much disruption from harmony).

At the end, a series of chords is given, with one chord for each note to harmonize.

2.2 Constraints-based harmonizer

Consraints are created from the possible chords of notes and the possible pairs from grammar. We first explain the basic constraints.

For each note, we need exactly one chord in the possible chords it has. Also, for each pair of two subsequent notes, we need exactly one of the possible pairs. We need also to make the link between the pairs and the chords.

For the constraints added if a solution could be found for the basic ones, we hinder having, for some type of chords, the same chord at interval two (so with one chord in-between), or at interval three (with two chords in-between). Only the most important chords of the harmony can be repeated at these intervals.

So constraints are in this form :

"No more than one in list of as" : $not(a_i)||not(a_j)$, for all $i! = j$ and $i < j$, "and" of all

"At least one in list of as" : $a_0||a_1||...||a_{last}$

"Pair p true $<=>$ chords a and b true" : $not(p)||a$ and $not(p)||b$ and $p||not(a)||not(b)$

"Exactly one" : "No more than one" and "At least one"

Silents are not taken into account in the constraints (before creating all the constraints, but after having created the "exactly one" constraints on possible chords, they are removed).

The found solution, if any, is converted at the end into a list of chords.

2.3 Composer constraints

The composer can specify for each note the set of possible chords he wants, and they are used instead of the possible ones generated (for the notes for which they exist). It was considered to keep only the intersection between the generated ones and the composer-chosen chords, but finally it was decided to give more responsibility to composer, and allow him/her to create dissonances if wanted. Also, doind this way can be solved the problem with notes that have no possible chords.

3. Use of the program

For the cadence and melody, it is straightforward how to give the input to the program, but for the composer

constraints, there are some details to know. The composer gives a list of indexed elements; the indices correspond to the ones of the notes in the to-be-harmonized ones, the elements are lists of "chords with inversions" one would like to have at the corresponding index (then one of them will be chosen, as explained in the Implementation part above). There is an implicit conversion from chords to "chords with inversion", and the fondamental form of the chord is chosen (not inverted). There are some examples of inputs in the code (in the composer input tests and other tests, for example) if needed.

The strategy to use this program, once one has selected notes to be harmonized, is to begin with the linear harmonizer. It will give indices of what could sound a bit weird, this with help of warnings. If one gets the warning that there is a too bizarre note, one could either not harmonize this note and make longer either the previous note or the next one (not in the melody, but on the notes to harmonize), or add a constraint on the chord to give to this note. If one gets other warnings, one can either decide to not harmonize a given note, as before, or separate a note in parts, or fusion two notes together, or, as before also, add constraint. One should do that until one has no more warning.

There is an example of use of this strategy in the gen.play.Example.scala file and the recordings of the corresponding results are in the root of the git repository.

Then one can try the constraints-based harmonizer, which will be able to solve with basic constraints, and perhaps with the more constraints, too, for a better sound.

One can also add some chords in the program, modify the grammar in the code but that is tedious when one does not know the code, but still a remark is given in the code to explain how to do that.

4. Possible Extensions

A possible extension of this project could be to allow the composer to force some series of chords, even if not harmonically correct. In fact, for now, as explained in implementation part, the composer constraints have to result in a harmonically correct serie of chords, otherwise either no solution is given (in the case of the constraints-based solving) or some constraints are not satisfied (in the case of the linear harmonizer).

Another possible extension could be to add constraints on the concrete note placement in chords.

5. Conclusion

To conclude, we will recall that we have on one hand, the linear harmonizer which always give some solution, but which does not always necessarily satisfies all composer constraints and can add dissonances (that are resolved in a subsequent chord, hopefully). On the other hand, we have the constraints-based harmonizer which does not always give a solution, but when it does, it has no added dissonances and satisfies all composer constraints. The linear harmonizer, with its diagnostics (warnings), can help the composer to get a solution from the second harmonizer, which can be better if the additional constraints could be solved, too.

So, with this program, melodies can be harmonized (using classical harmony rules), and composer can influence the process with more constraints and by the choice of which notes to harmonize.

References

R. Blanc. CafeSat: A Modern SAT Solver for Scala. *Conference: Proceedings of the 4th Workshop on Scala*, 2013. URL `https://github.com/regb/scabolic`.

M. Merriman. *The Music Theory Handbook*. Thomson Schirmer, 1997.

F. Pachet and P. Roy. Musical Harmonization with Constraints: A Survey. *Constraints Journal*, 6:7–19, 2001.

V. Pittet. ScalaMusicGeneration - MusicInterface. 2014. URL `https://github.com/vtpittet/ScalaMusicGeneration`.