AuToTen

Introduction

This is a repository for AuToTen (as in **Au**tomatic **To**nal **Ten**sion), a Python-based system which automatically calculates the contributions to tonal tension of a piece of music according to Lerdahl's model of tonal tension.

Software specifications

AuToTen's repository has been implemented using:

- Ubuntu 16.04.6 LTS
- Python 3.7.0 (also tested with IPython 6.5.0)
- music21 5.7.2
- pandas 0.23.4

How to use AuToTen?

1. Download and decompress `AuToTen-repository.zip` at your desired location.
2. Access that location and run the file `run.py` using a Python compiler of your choice (e.g. Jupyter notebooks, IPython, etc).
3. A dialog will pop up and will ask you to select a piece of music in MusicXML format. For example, try selecting the piece *example.xml* which could be found in the folder */example/*.
4. Another two dialogs will ask you to select the piece's *prolongational reduction* and *metrical reduction*, again in MusicXML format. These files should incorporate the piece's prolongational and metrical componennts according to the Generative Theory of Tonal Music (GTTM). For example, try selecting the files *example-pr.xml* and *example-mr.xml*, both in the folder */example/*. When using your own piece of music, you must calculate these two files a priori. For this purpose, you can use the The Interactive GTTM Analyser (IGA) (see gttm.jp). Please note IGA was not developed by AuToTen's authors, nor is hosted and maintained by them. If using IGA, we recommend storing the piece of music in such a way its melody is included in a single voice. This is because the current version of IGA produces more accurate results in this way.
5. A final dialog will ask you to pick the location where you want to store AuToTen's outputs. These outputs consist of two CSV files which include quantitative values of the piece's *global tension* and *attraction* according to Lerdahl's model of tonal tension. For example, see *example-auto-tension.csv* and *example-auto-attraction.csv* in */example/*.

What else can be done with AuToTen?

Alternatively, you can also call AuToTen's util functions independently. We recommend first running `setup.py` (again, access AuToTen's directory and use your Python compiler). In this way, you will be able to directly call the following functions:

1. `distance`, which calculates the distance between two chords in Lerdahl's Tonal Pitch Space (TPS).
    - This function takes three inputs: a key, a reference chord and a chord to which transition.
    - Input the key in alphabet format, using upper case for major keys, lower case for minor keys and the symbols # and *b* for sharp and flat, respectively (e.g. *C*, *a*, *Db*, *f#*).
    - Input the chords in Roman numeral format (i.e. *degree/key*), using upper case for major degrees and major keys, lower case for minor degrees and minor keys, the symbols #, *b* and *n* for sharp, flat and natural respectively, the symbol 7 for diatonic sevenths and the symbols + and - for augmented and diminished degress, respectively (e.g. I/I, V7/i, vii-/IV, III+/II).
    - For example, try callying `distance('C', 'I/I', 'I/bII')`. It corresponds to the TPS distance between the tonic chord of C major and the tonic chord of Db major. As output, you will get the tuple `(21, 5, 5, 11)`, which icludes the total distance between the two chords and the values of TPS' parameters, *i*, *j*, *k*, respectively.
2. `offset`, which calculates the values of the offsets of the musical events (i.e. notes and chords) included in the piece's *metrical reduction*.
    - This function takes one input: a piece's *metrical reduction* in MusicXML format.
    - For example, try callying `offset('/.../example/example-mr.xml')`. Note that you will have to include the specific path to the directory where you store AuToTen in. As output, you will get a list of floats, `[0.0,0.75, ..., 22.0, 22.05]`, which represent the time of the inception of the offsets in *example-mr.xml*.
    - Nothe that, when using `run.py`, this list of offsets is stored in a CSV file in your working directory. See, for example, *example-auto-offsets.csv* in */example/*.
3. `generator`, which calculates a representation, in the form of a matrix, of the piece's *prolongational reduction*.
    - This function takes one input: a piece's *prolongational reduction* in MusicXML format.
    - For example, try callying `generator('/.../example/example-pr.xml')`. Note that you will have to include the specific path to the directory

where you store AuToTen in. As output, you will get a list of lists representing a matrix. This matrix represents the degrees of embedding in the GTTM-based hierarchical classification (i.e. a tree) provided by `example-pr.xml`.
- Nothe that, when using `run.py`, this matrix is stored in a CSV file in your working directory. See, for example, *example-auto-matrix.csv* in */example/*.

4. `parameters_finder`, which calculates the parameters needed for the calculation of *global tension* and *attraction* according to Lerdahl's model of tonal tension.
    - This function takes two inputs: a piece in MusicXML format and a CSV file containing its offsets (recall `offset`).
    - For example, try callying `parameters_finder(piece, offsets)`, being `piece` the file `'/.../example/example.xml'` and `offset` the output given by `offset('/.../example/example-mr.xml')`. Note that you will have to include the specific path to the directory where you store AuToTen in. As output, you will get a table with the data needed to use Lerdahl's model of tension. For each offset, this table includes an estimation of its *key*, *chord label* and *chordal notes*, as well as the values of the paramerters in Lerdahl's model of tonal tension (i.e. *inversion*, *non-harmonic* and *scale degree*).
    - Nothe that, when using `run.py`, this table is stored in a CSV file in your working directory. See, for example, *example-auto-piece-data.csv* in */example/*.

5. `t_calculator`, which calculates the values of *global tension* of a given piece of music according to Lerdahl's model of tonal tension. See, for example, *example-auto-tension.csv* in */example*.
    - This function takes two inputs: a piece's parameters with regards to Lerdahl's model of tonal tension (recall `parameters_finder`) and a piece's matrix representation of its *prolongational reduction* (recall `generator`).
    - For example, try callying `t_calculator(parameters, matrix)`, being `parameters` the file `'/.../example/example-auto-piece-data.xml'` and `matrix` the file `'/.../example/example-auto-matrix.xml'`. Note that you will have to include the specific path to the directory where you store AuToTen in. As output, you will get a list of integers, `[6,13, ..., 15,1]`, which represent the values of *global tension* of the piece's *offsets* according to Lerdahl's model of tonal tension.
    - Nothe that, when using `run.py`, this list is stored in a CSV file in your working directory. See, for example, *example-auto-tension.csv* in */example/*.

6. `a_calculator`, which calculates the values of *harmonic attraction* of a given piece of music according to Lerdahl's model of tonal tension.

- This function takes one input: a list of attraction parameters with regards to Lerdahl's model of tonal tension (recall `parameters_finder`).
- For example, try callying `a_calculator(parameters[1])`, being `parameters` the output of the previously described `parameters_finder(piece, offsets)`. As output, you will get a list of floats, `[1.319,4.0, ..., 3.438,0.662]`, which represent the values of *attraction* of the piece's *offsets* according to Lerdahl's model of tonal tension.
- Nothe that, when using `run.py`, this list is stored in a CSV file in your working directory. See, for example, *example-auto-attraction.csv* in */example/*.

How does AuToTen work?

The workflow of AuToTen is shown in the figure below. As input, AuToTen is fed with a piece of music, in MusicXML format, and it outputs its values of *global tension* and *attraction*. The unshaded boxes in the workflow represent the algorithms embedded in AuToTen, whereas the shaded boxes represent the corresponding inputs and outputs to these algorithms. There is also a box which has been highlighted with a dotted line. It represents the GTTM-based analysis that has to be performed using IGA.

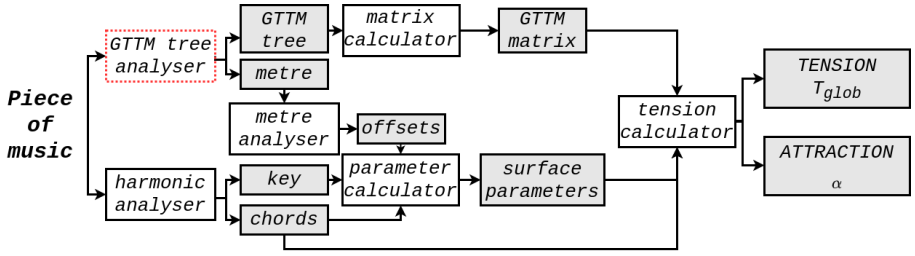

Figure 1: AuToTen's workflow

AuToTen's implementation

To calculate the values of *tension* and *attraction* of a given piece of music, according to Lerdahl's model of tonal tension, AuToTen takes the following steps:

Step 1

Identify the most probable patterns of strong and weak beats, and how might these be grouped, as well as the piece's most probable hierarchical relations according to GTTM rules.

Recall this step could be performed by IGA's *GTTM tree analyser*. As input, it needs a piece of music, in MusicXML format. As output, it produces the piece's hierarchical structure (*GTTM tree* in the above figure) and its metrical structure (*metre* in the above figure), both in MusicXML format.

Step 2

Define the musical events that will be assigned with a tension value, according to the analysis performed in the previous step.

Step 2 is performed by the *metre analyser*. As input, it is fed with *metre*, calculated in Step 1. As output, it produces a list of the piece's *offsets*. These are interpreted by AuToTen as the list of the piece's beats which will be assigned with a value of tension and attraction.

In AuToTen, the *metre analyser* corresponds to `offset` from `functions.metre`. The input file, *metre*, provided by IGA, is stored as a *MusicXML ElementTree*. In this tree, the offsets are labelled as *metric dots*. `offset` simply finds these *metric dots* and stores their offset values in a flat list.

Step 3

Represent the hierarchical relations in such a way that will facilitate the calculation of tension. Note this representation, in the case of manual calculations, is given in the form of a tree.

Step 3 is performed by the *matrix calculator*. As input, it is fed with the *GTTM tree*, calculated in Step 1. As output, it produces a *GTTM matrix*. This matrix is AuToTen's representation of the hierarchical relations embedded in a *GTTM tree*.

In AuToTen, the *matrix calculator* corresponds to `generator` from `functions.prolongation`. The input file, *GTTM tree*, provided by IGA, is stored as a *MusicXML ElementTree*. Be a *GTTM tree* with n events, `generator` will calculate its representation as a nxn *GTTM matrix*. A given event in the piece, x, will be represented in the *GTTM matrix* by the *xth* row, which will only contain a non-empty value, that coinciding with column y, where y corresponds to the branch to which x is attached in the *GTTM tree*. An example is shown in the figure below. Note that the highest dominating event is number 5, as there are no branches above it. This will be denoted by a 0 at the (5,5) element in the *GTTM matrix*. The events that connect to 5 in the tree, these are events 1 and 4, are represented by a value of 1 in the *GTTM matrix* (i.e. there is only one branch above them) (see elements (1,5) and (4,5)). And so on. Notice that the rest of the elements in the *GTTM matrix* are kept empty.

Step 4

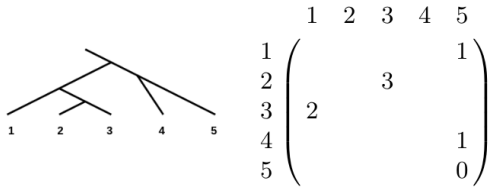

Figure 2: GTTM matrix calculation example

Perform a harmonic analysis of the defined musical events and label them accordingly (Roman Numeral analysis is the notation used all along Lerdahl's work).

Step 4 is performed by the *harmonic analyser*. As input, it is fed with the piece of music, in MusicXML format. As output, it calculates the most suitable *key* and *chord labels* of the piece's musical events.

In AuToTen, the *harmonic analyser* corresponds to `chords_parameters` from `functions.variables.chords_components`, which uses the toolkit *music21*. Given a piece of music, `chords_parameters` estimates the piece's key using *music21*'s `analyze('key')`. Likewise, it estimates the most suitable key for each measure using `analyze` in every measure (i.e. `getElementsByClass('Measure')`). In this way, different versions of the chords' labels can be calculated. On one hand, *music21*'s `chordify` is used to estimate the chords' labels using the whole piece's estimated key. On the other hand, `chordify` is used to estimate different versions of the chords' labels for every measure's key. These two different methods will allow to better identify non-diatonic chords, such as secondary dominants.

Sometimes, *music21* identifies a chord which is missing its third. In this cases, *music21*'s `quality` will not identify the chord as being major, minor, augmented or diminished but as *other*. To deal with this issue, `chords_parameters` will modify the corresponding chord labels so that they represent their respective diatonic version.

Step 5

Calculate the surface parameters for all musical events.

Step 5 is performed by the *parameter calculator*. As input, it is fed with the list of

offsets, calculated in Step 2, and the piece's *key* and its *chords*'labels, calculated in Step 4. As output, it produces the piece's values of *surface parameters*, needed to apply the rules in Lerdahl's model of tonal tension. These parameters concern the scale degree of the chords' highest note, the chords' inversions and the role the chords' notes play within the Tonal Pitch Space.

In AuToTen, the *parameter calculator* corresponds to `parameters_finder` from `functions.surface` the previous `chords_parameters`. The former calculates the *surface parameters* for every different version of the chords labels estimated in Step 4. To do so, it uses *muisc21*'s `getScaleDegreeFromPitch`, `inversion` and `pitches`, as well as the *surface parameters*' theoretical weightings defined in Lerdahl's model of tonal tension. The latter estimates the most appropriate chord labels and their corresponding *surface parameters*. To do so, `parameters_finder` compares the *surface parameter* that concerns the existence of non-harmonic notes in each chord and estimates the final chord labels that will suit the least non-harmonic musical discourse. Likewise, `parameters_finder` will correct any offset-related mismatching. Note that two different systems have been used to analyse the musical events in a given piece of music: IGA and *music21*. It might be the case where the number of events found by these two systems is not the same. AuToTen will be using the events calculated by IGA, as the prolongational relations in the *GTTM tree* are defined according to these events. So that, `parameters_finder` will compare IGA's offsets with those of *music21* and will assign the appropriate chord label and a set of *surface parameters* to each of IGA's offsets.

Step 6

Calculate the values of global tension and attraction of all musical events.

Finally, Step 6 is performed by the *tension calculator*. As input, it is fed with the *surface parameters*, the *chords*' labels and the *GTTM matrix*, calculated in Steps 5, 4 and 3, respectively. As output, it produces the values of *global tension* and *attraction*, according to Lerdahl's model of tonal tension, of all musical events in the input piece of music.

In AuToTen, the *tension calculator* corresponds to `t_calculator` from `functions.tension` and `a_calculator` from `functions.attraction`. Both functions need from the distance values between chords within TPS, which is given by `distance` from `functions.tps`. According to Lerdahl's model of tension, this distance, d, is computed as $d = i+j+k$, where i represents the distance between two chords in the chromatic circle-of-fifths, j represents the distance between two chords in the diatonic circle-of-fifths and k represents the difference between the representation of two chords within the Tonal Pitch space, known as *basic spaces*. Thus, `distance` calls the functions `i`, `j` and `k`, all from `functions.variables.tps_components`. To operate, these three functions need the *basic* and *chordal spaces* of the two input chords. These spaces are calculated through `Space` from `functions.classes.space`. The input chords,

in Roman numeral format, are translated into the alphabet key-signature format in `parser` from `functions.classes`. Their notes are then calculated by the `functions.classes.notes`. All the parameters needed for the calculations are included in `common` from `functions.parameters`.

`t_calculator` calculates the flow of *hierarchical tension* of a given piece of music. According to Lerdahl's model, the final value of *hierarchical tension* between two chords depends on the *surface parameters*, the TPS distance between chords and a collection of inherited distances that depends on the hierarchical structure defined by the *GTTM tree*. These three contributions are calculated using the functions `dissonance` and `inherited`, both from `functions.variables.tension_components`, as well as `distance`. For these functions to operate, they need the *GTTM matrix* and the *chords labels*, which are processed using `sequence.Harmony` and `matrix.Reduction` both from `functions.classes`.

`a_calculator` calculates the flow of *harmonic attraction* of a given piece of music. According to Lerdahl's model, the final value of *harmonic attraction* between two chords depends on the TPS distance between them, the intervals between their notes and their *anchoring spaces*, which are updated versions of the chords' *basic spaces*.

How was AuToTen evaluated?

One hundred test cases were manually annotated to test the validity of AuToTen's calculations. Likewise, a computational evaluation was carried out using four pieces of music: Wagner's Grail theme from Parsifal, Bach's chorale "Christus, der ist mein Leben", and harmonic reductions of Chopin's E major prelude and the first phrase in Mozart's sonata k.282. AuToTen's outputs showed strong and statistically significant correlations against all pieces, but that of Chopin's, the longest piece with many modulations, whose correlation was moderately strong.

The user can run the functions `tests.distance_tests` and `tests.tension_tests` to see how AuToTen's calculations agree with the ground-truths in all 100 cases.

The data used to produce the results of the test cases and the evaluation can be found in `/tests` and `/evaluation` respectively.

Where to find more information about the theoretical concepts upon which AuToTen is built?

- Generative Theory of Tonal Music: https://music.columbia.edu/publications/books/a-generative-theory-of-tonal-music
- Tonal Pitch Space: https://music.columbia.edu/publications/books/tonal-pitch-space

- Lerdhal's model of tonal tension: https://online.ucpress.edu/mp/article/24/4/329/95267/Modeling-Tonal-Tension
- Interactive GTTM Analyser: http://www.gttm.jp/
- music21: http://web.mit.edu/music21/