FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automatische Generierung von Arrangements für Solo-Gitarre auf der Basis von Lead Sheets

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Computergraphik/Digitale Bildverarbeitung

eingereicht von

Arnaud Moreau
Matrikelnummer 0325440

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Univ.-Prof. DI Dr. Gerhard Widmer

Wien, 15.10.2010 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automatic generation of solo-guitar arrangements from lead sheets

DIPLOMA THESIS

in Partial Fulfillment of the Requirements for the Degree

Master of Science

completing the studies of

Computer Graphics/Digital Image Processing

by

Arnaud Moreau
Matriculation number 0325440

at the
Faculty of Informatics of the Vienna University of Technology

Under the direction of
Supervisor: Univ.-Prof. DI Dr. Gerhard Widmer

Vienna, 10/15/2010

(Signature Author) (Signature Supervisor)

Abstract

In this work the process of automatically arranging songs for solo-guitar by a computer on the basis of melodic and harmonic information contained in lead sheets is analyzed and discussed. The problem is defined and broken down to sub problems (guitar fingering, music segmentation) and solutions are presented and discussed. The focus of this work is mainly on technical rather than musical aspects. First an overview on the state of the art is given. Furthermore a prototype implementation is presented that incorporates many of herein discussed thoughts. The primary target of the implemented algorithm is to ensure the playability of the arrangement on the guitar, which imposes by nature constraints on the result. The results are generated using a genetic algorithm that optimizes the arrangements towards better playability. Background information on problems, methods (genetic algorithms) and data representations (MusicXML) is given in the introduction part. The arrangements produced by the prototype based on a testbed of six songs in the genre of Bossa Nova are discussed in detail. This work concludes with an outlook on future work.

Kurzfassung

In dieser Arbeit wird der Prozess des automatischen Arrangierens von Liedern für Solo-Gitarre mit Hilfe eines Computers auf der Basis von melodischer und harmonischer Information, die in Lead Sheets enthalten ist, analysiert und diskutiert. Das Problem wird definiert und in Subprobleme aufgespalten (Fingersatz, Musiksegmentierung) und Lösungen werden präsentiert und diskutiert. Der Hauptfokus dieser Arbeit liegt eher auf den technischen als den musikalischen Aspekten. Zuerst wird eine Übersicht auf den aktuellen Stand der Technik geboten. Weiters wird eine Prototyp Implemenentierung vorgestellt, die viele der in dieser Arbeit geäußerten Ideen einbezieht. Das primäre Ziel des implementierten Algorithmus ist es, die Spielbarkeit des Arrangements auf der Gitarre, die von Natur aus die Ergebnisse einschränkt, zu gewährleisten. Die Resultate werden mit Hilfe eines genetischen Algorithmus generiert, der die Arrangements in Richtung besserer Spielbarkeit optimiert. Es werden Hintergrundinformationen über Probleme, Methoden (genetische Algorithmen) und Datenrepräsentationen in einem Einführungsteil bereit gestellt. Die Arrangements, die vom Prototypen erzeugt wurden, werden anhand eines Datensatzes aus sechs Bossa Nova Liedern im Detail diskutiert. Diese Arbeit endet mit einem Ausblick auf mögliche zukünftige Entwicklungen.

Acknowledgements

This work would not have been possible without the help of countless people. First there is Prof. Widmer, who thankfully accepted to supervise it. He is incredibly inspiring, supporting and contributed helpful comments all along the way (and it was a long one!). I am thankful that he is one of the very few geniuses with whom I had the honor to work in my life.

My parents Doris and Rémy Moreau who have supported me with all their dedication. My uncle Gerhard Dirmoser who brought me to the field of computer science. Helmut Schönleitner, for having me taught everything I know about music. My partner Judith Weißengruber who withstood my moods and endless moaning and constantly attempts to make me a better person (which is not an easy thing to do). I would also like to thank Daniele Radicioni, the author of very important, illuminating and thorough articles [RAL04, RL05a, RL05b, Rad06, RL07] for his input on the guitar-fingering subject.

Many thanks also to Judith Weißengruber and Laura Trunkenpolz for their editorial work and all the contributors (and uploaders) to the wonderful projects Musescore and Wikifonia.

Joe Zawinul, Richard Bona, Mother's finest, Yes, John McLaughlin, Trilok Gurtu, Antonio Forcione, Joni Mitchell, Steps ahead, Nguyên Lê, Red Hot Chili Peppers, The Beatles, Shakti, Mahavishnu Orchestra, The Police, Sting, U2, Jimmy Hendrix, Helmut Schönleitner and many more for making the best music on this planet.

Contents

Abstract i

Kurzfassung i

Acknowledgements iii

Contents v

List of Algorithms vii

List of Figures vii

List of Listings ix

List of Tables ix

I Introduction 1

1 Motivation 3

2 State of the art 5
2.1 Music arrangement 5
2.2 The guitar 7
2.3 Segmentation 9

3 Algorithms and data representation 11
3.1 Music notation 11
3.2 Genetic algorithms 15
3.3 Constraint satisfaction problems 19

II Methodology 23

4 Framework 25

4.1 Goals . . . 25
4.2 Design . . . 26

5 Arrangement algorithm **31**
5.1 Segmentation . . . 31
5.2 Genetic algorithm . . . 35

III Results **47**

6 Executing the arrangement algorithm **49**
6.1 Segmentation . . . 50
6.2 Genetic algorithm . . . 54

7 Conclusion and Outlook **57**
7.1 Future work . . . 57

A Lead sheets and arrangements **59**
A.1 One Note Samba . . . 60
A.2 The Girl from Ipanema . . . 63
A.3 Corcovado . . . 67
A.4 Desafinado . . . 70
A.5 A Felicidade . . . 75
A.6 How Insensitive . . . 80

Acronyms and abbreviations **83**

Bibliography **85**

Erklärung zur Verfassung der Arbeit **91**

List of Algorithms

1 AC-3 as defined in [Kum92]. 20
2 Backtracking as defined in [RN03] 21
3 Algorithm computing the local boundary detection model as defined in [Cam01]. 32
4 Algorithm that searches optimal segments from LBDM data. 33
5 Algorithm to segment an arrangement into MEL, CHO and MIX blocks. 39

List of Figures

2.1 A prototypical electric guitar. (GreyCat, "File:Electric guitar parts.jpg" May 22, 2010 via Wikimedia Commons, Creative Commons Attribution-Share Alike 2.5 Generic license) 7
2.2 The first 12 frets of a guitar fretboard. The pitch is indicated in MIDI number. See Figure 3.2 in [Rad06]. 8
2.3 This diagram shows the number of different position on the fretboard for playing the indicated pitch (given as MIDI number) on the x-axis. See Figure 3.3 in [Rad06]. 9
2.4 The LBDM applied to the first 8 bars of the song "How Insensitive" by Antonio Carlos Jobim. The blue line shows the resulting LBDM boundary-strength function. 10

3.1 An example tablature. (Guitar intro from "Stairway to Heaven" by Jimmy Page and Robert Plant) 12
3.2 Single point crossover. (Rgarvage, "File:SinglePointCrossover.png" May 30, 2010 via Wikipedia, Creative Commons Attribution-ShareAlike 3.0 license) 17
3.3 The map coloring problem (a): Each region $V_1 \dots V_4$ has to be painted with one color such that no two neighboring regions have the same color; (b) shows the equivalent constraint graph. See Figure 3.9 from [Rad06]. 19

4.1 UML diagram of the classes representing notes. 26

4.2 UML diagram of the classes representing the guitar. 27
4.3 A demonstration of the difference between position (a) and onset (b) representation. 28
4.4 UML diagram of the classes representing higher level data structures. 28

5.1 Visualization of the segmentation tree and the corresponding LBDM boundary-strength function from the song "The Girl from Ipanema". 34
5.2 Illustration of the weight function as defined in Equation 5.3 with $\mu = 25$ and $\delta = 9$. 35
5.3 Definition and visualization of different movement types in guitar performance according to [Rad06]. 44
5.4 $fret\_stretch$ functions for the index finger (from [Rad06]). 45
5.5 This figure shows examples of a MEL and a CHO block including fingering that is used to illustrate the functionality of the fitness function. Guitar string and fret information are contained in the lower tablature staff and fingers are indicated in the upper staff. 45

6.1 (Pseudo) Bossa Nova pattern used for potential chord note positions. 54
6.2 Demonstration of the optimization process of the first segment in the song "Corcovado". The blue line in both plots shows the fitness score of the best candidate in the population over generations, the green line indicates the mean fitness in the population. The plot below zooms in the same data of the first 700 generations, such that the increase in the mean fitness after the first generation can be seen distinctly. . . . 55

A.1 One Note Samba by Antonio Carlos Jobim – Lead Sheet (tom, "One Note Samba" July 23rd, 2007 via Wikifonia [Fou10], © reserved by Musicopy) 60
A.2 One Note Samba – Arrangement 61
A.3 The Girl from Ipanema by Antonio Carlos Jobim – Lead Sheet (benoit, "The Girl from Ipanema" November 5th, 2006 via Wikifonia [Fou10], © reserved by Musicopy) 63
A.4 The Girl from Ipanema – Arrangement 65
A.5 Corcovado by Antonio Carlos Jobim – Lead Sheet (Musicdad, "Corcovado – Quiet Nights Of Quiet Stars" August 9th, 2010 via Wikifonia [Fou10], © reserved by Musicopy) 67
A.6 Corcovado – Arrangement 68
A.7 Desafinado by Antonio Carlos Jobim – Lead Sheet (lasconic, "Desafinado – Slightly out of Tune" December 18th, 2008 via Wikifonia [Fou10], © reserved by Musicopy) 70
A.8 Desafinado – Arrangement 72
A.9 A Felicidade by Antonio Carlos Jobim – Lead Sheet (TxRx, "A Felicidade" January 24th, 2009 via Wikifonia [Fou10], © reserved by Musicopy) 75
A.10 A Felicidade – Arrangement 77
A.11 How Insensitive by Antonio Carlos Jobim – Lead Sheet (Mauro58, "How Insensitive – Insensatez" September 19th, 2009 via Wikifonia [Fou10], © reserved by Musicopy) 80
A.12 How Insensitive – Arrangement 81

List of Listings

3.1 MusicXML file example; header information, metadata. 13
3.2 MusicXML file example; notes, rests and chord symbols. 14
3.3 MusicXML file example; tablature. 14

List of Tables

5.1 Specification of the maximum finger span from [Rad06] 37

6.1 Overview of test songs – result reference 49
6.2 One Note Samba – Segmentation 50
6.3 The Girl from Ipanema – Segmentation 51
6.4 Corcovado – Segmentation 52
6.5 Desafinado – Segmentation 52
6.6 A Felicidade – Segmentation 53
6.7 How Insensitive – Segmentation 53

A.1 Overview of test songs – origin information 59

Part I

Introduction

CHAPTER 1

Motivation

Ein **A**nfänger **d**er **G**itarre **h**at **E**ifer.
(A beginner of guitar playing has enthusiasm.)

Mnemonic often found in german guitar teaching books to remember the tuning of the strings.

This thesis is about arranging songs that are given as lead sheets (containing melody and chord symbols) with the help of a computer software. The goal of this arrangement process is that the result, which already contains fingering information, can be played by a single performer on the guitar. For this purpose not only the melody is included in the arrangement, but also chord notes on musically reasonable positions.

Although the guitar is an instrument most commonly used to accompany singers or instrumental musicians, there comes a time for each guitar player when she/he performs a piece on her/his own. Since in the Jazz or Pop genre a song is notated in the form of a lead sheet one has to not only play the melody of the piece but also use the given chords to enhance the arrangement. That is often a tedious intellectual task, because one has to find the correct fingering for the melody and also insert other harmonically interesting notes that have to fit the rhythmic but also the cognitive and bio-mechanical context.

The practical aspects about having a software that can arrange songs from lead sheets are the following: It can be used to save a lot of time for the musician, even if its results are interpreted as suggestions or ideas rather than carved-in-stone. Teachers can use it to provide their students' favorite songs without having too much trouble doing the work themselves. Also said students can use it for arrangement without having enough knowledge to do it themselves, they could then use them as a basis to learn about harmonic concepts and guitar fingering. A necessary feature for this would be the possibility to adjust the level of difficulty.

From a scientific perspective automatic arrangement is a rather unexplored problem (for a literature review see Chapter 2.1) and can still originate interesting results from a musicology but also a computer science point of view. The problem poses a huge challenge to algorithms due to its many degrees of freedom (not only the complexity of the guitar fingering problem –

see Section 2.2 for a more detailed discussion – but also the harmonic and rhythmic possibilities are nearly endless). Moreover the point of evaluating the resulting arrangement is an open issue, as there is neither an exactly defined "correct" solution nor a database of lead sheets and corresponding arrangements from multiple human "experts" to compare automatically generated results to. Furthermore the evaluation is context and genre dependent.

In the remainder of this part the discussed problem in all its facets and used technology to solve it is described in more detail including a state-of-the-art literature review. This leads to Part II (Methodology), where the prototype implementation is introduced. At the end the resulting generated arrangements are discussed and a line of future work is proposed.

CHAPTER 2

State of the art

The following sections describe problems that come up on the way to automatic arrangement. A snapshot on current literature is provided as well as an overview on the terminology used in this thesis. First the topic of (automatic) music arrangement is covered, then the guitar is introduced together with the guitar fingering problem. The chapter concludes with melodic segmentation.

2.1 Music arrangement

According to [Wik10] the American Federation of Musicians[1] defines arranging as

> "the art of preparing and adapting an already written composition for presentation in other than its original form. An arrangement may include reharmonization, paraphrasing, and/or development of a composition, so that it fully represents the melodic, harmonic, and rhythmic structure".

This describes exactly the process of music arrangement as understood in this thesis. Here, an already existing composition (given in the form of a lead-sheet – see Section 3.1.1) is transformed to another musical form (solo-guitar instrumentation) while preserving the musical idea of the piece. In an extended sense this could also include reharmonization or other compositional techniques, which, however, is not within the scope of this work. An arranger could use the following steps to carry out the process of arrangement:

1. Decide on a musical genre.
2. Put the melody in the score.
3. Decide on where to put additional harmony notes in the score (the rhythm patterns are determined by the genre).

[1] http://en.wikipedia.org/wiki/American_Federation_of_Musicians

4. Decide on chord voicings (not only determined by the sound but also by the playability on the instrument).

5. Extend the chord progression where appropriate (transition chords) or change the given chord information (reharmonization, chord substitution).

6. Put the bass notes independently of the other chord notes (bassline).

7. Find an appropriate fingering for the performer.

One issue of rather philosophic nature is the question what the musical idea of a piece is and whether it exists at all. Diverging opinions about the musical idea of a given piece suggest that it is a more subjective rather than objective thing. Is it even distinct from the piece of music itself? Here we view any form of musical notation as a guideline for the performer, as we assume that the piece itself exists only as the interpretation in the performance.

Turning to the literature about automatic arrangement, there is one master thesis by Daniel Tuohy [Tuo06] and published extracts thereof [TP06b, TP06a] that discuss this topic particularly for the guitar. In this work a genetic algorithm (GA) or a greedy hill-climber (HC) arranges music in arbitrary form (e.g. Symphony No. 2 Mvt. II by Rachmaninoff) for solo-guitar. For that purpose all notes in the original score are annotated (in this case manually – but the author points out potential published algorithms to perform this task) by a weight that assesses their importance in the piece. This is how the composition is preserved, however other factors like playability[2] are also taken into account. The author then presents different kinds of GAs and an artificial neural network (ANN) to find the fingering (tablature) of the arrangement.

Other work describes arrangement in another context: In [NK97] the authors examine the ability of chaotic neural networks to generate variations of an original melody. The system presented in [CV06] re-arranges (or rather re-mixes) music segments according to a target affective state, thus it is not consistent with the term arrangement as used here in a more narrow sense.

Related work in automatic arrangement for the piano has been done by Emura et al. [EMY08]. They present a system that arranges Jazz songs given by lead-sheets on the basis of Jazz harmony theory considering the concept of voicing (assignment and ordering of chord notes based on a given chord symbol). The user is required to choose voicing type and matching approach strategies (smooth connection of neighboring chords) and the algorithm inserts chord notes on every melody note position. The system is compared against commercially available arrangement software like "Band in a box" by means of an expert survey. Chiu and coworkers [CSH09] propose a similar arrangement system for piano as Tuohy et al. [TP06b] do for guitar. The original score is analyzed to identify phrases and estimate their importance, then they are selected for inclusion in the final piano arrangement considering playability.

When considering the literature in the field of computer music, it is surprising that a lot of work was done in automatic composition or improvisation (representative selection: [GB91, RIHL93, Bil94, Jac95, Mar96, TI00, Mir01, GJC03, KM07, Ari09]) whereas the more restricted process of automatic arrangement, being presumably a more suitable task for computers, was given little thought so far in comparison [NK97, CV06, TP06b, TP06a, Tuo06, EMY08,

[2] Playability in this thesis always assumes a performer having an average sized hand with 5 fingers.

CSH09]. Although of course these two tasks are heavily related and the solutions and results are beneficial to one another.

2.2 The guitar

The guitar (see Figure 2.1) is a stringed instrument consisting of a body (9) and a neck (7) to which the strings (usually six) are attached. The pitch is changed by pressing down a string on the fretboard (4) and the sound is produced by plucking the same string.

Definition 1. *To denote pitch the* scientific pitch notation[3] *is used, as defined in [You39]. The note name is composed of a letter* $\{C, D, E, F, G, A, B, C\}$*, accidentals* $\{\sharp, \flat\}$ *and an octave number, starting with the reference note* $C_0 = 16.352\,Hz$*. In addition to that the MIDI number may be given as defined in [Ass96], starting at MIDI note 0,* $C_{-1} = 8.1758\,Hz$*, and extending above MIDI note 127,* $G_9 = 12543.875\,Hz$*.*

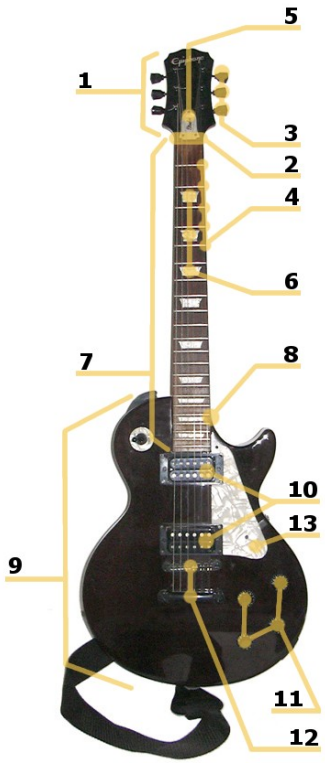


Figure 2.1: A prototypical electric guitar. (GreyCat, "File:Electric guitar parts.jpg" May 22, 2010 via Wikimedia Commons, Creative Commons Attribution-Share Alike 2.5 Generic license)

Definition 2. *Two different notations for fingered position on the guitar fretboard are used:*

[3] http://en.wikipedia.org/wiki/Scientific_pitch_notation

1. *2-tuples in the form* $(string, fret)$
2. *3-tuples in the form* $(string, fret, finger)$

$string = \{1 \dots 6\}$ *ordered from highest to lowest pitch.* $fret = \{0 \dots n\}$*, 0 denoting plucking the empty string,* n *denoting the number of frets on the guitar.* $finger = \{1 \dots 4\}$*, denoting the fingers from index to pinky.*

The standard tuning of the six strings is from lowest to highest pitch (MIDI number given in parentheses): E_2 (40), A_2 (45), D_3 (50), G_3 (55), B_3 (59), E_4 (64). The diagram in Figure 2.2 shows the pitches of all playable notes on the first 12 frets of a guitar in standard tuning. Since one fret alters the pitch by one half-tone, the 12th fret of a string sounds one octave above the tuning note. From the diagram you can also see that due to the overlap of the string's pitch ranges one given note can be played on multiple positions on the fretboard. For example the note F_3 (53) can be played on the following positions: $(6, 13)$, $(5, 8)$, $(4, 3)$.

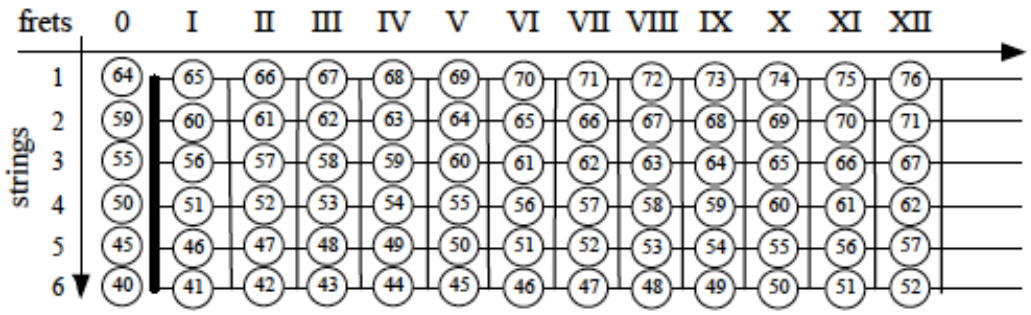


Figure 2.2: The first 12 frets of a guitar fretboard. The pitch is indicated in MIDI number. See Figure 3.2 in [Rad06].

If we ask how many different positions there are for a given pitch we can consider the diagram in Figure 2.3. It shows that a given note on average can be played on 2.94 different positions if we consider a guitar with a fretboard that has 22 frets (like the one depicted in Figure 2.1).

A sub problem in automatic arrangement for guitar is the guitar fingering problem (finding fingering positions $(string, fret)$ or $(string, fret, finger)$ for each note in a given score), that is addressed in [WL97, MY02, RD04, TDSR04, TP05, TP06c, RAL04, RL05a, RL05b, Rad06, RL07]. Since there are up to 5 positions for each given note (see Figure 2.3), a score containing n notes can generate up to 5^n different fingerings. If you additionally consider that each of those positions can be played by 4 different fingers in the worst case, the complexity can grow up to 20^n [Rad06].

In order to arrange music for solo-guitar one has to consider the playability on the instrument, which is governed by cognitive as well as bio-mechanical constraints. The most comprehensive work in this field can be found in [Rad06], where a complete model for the performer's

search for fingering positions is devised. All possible fingerings of a given note in the score are seen as vertices of a graph, which are connected to the subsequent vertices. Polyphonic passages are modeled as constraint satisfaction problems (CSPs) and are connected to neighboring notes the same way. Furthermore the edges of this graph are annotated by a difficulty score, which has also been used in this work to assess playability, because the validity of this algorithm has been proven experimentally. The solution to the fingering problem is then to find the shortest path through the graph. There is also work describing solutions to the fingering problem for other instruments – in this example [AKNR07] for the piano – that is also modeled as a graph search problem.

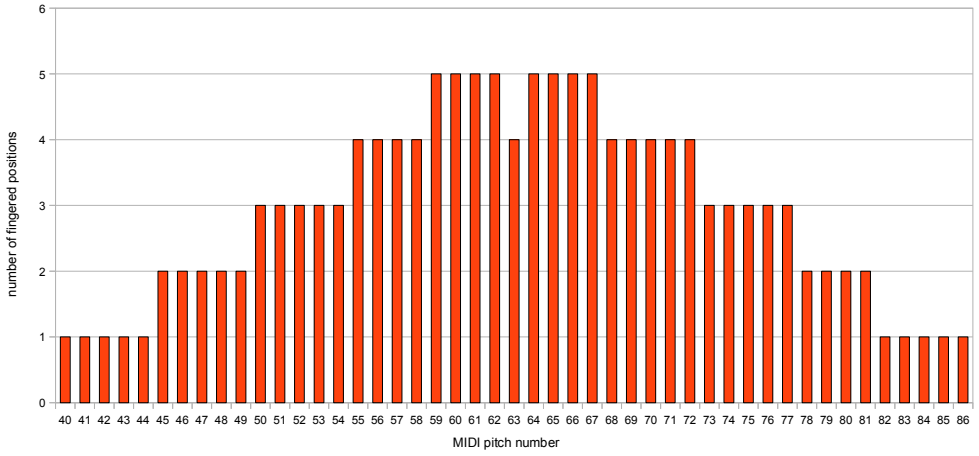


Figure 2.3: This diagram shows the number of different position on the fretboard for playing the indicated pitch (given as MIDI number) on the x-axis. See Figure 3.3 in [Rad06].

2.3 Segmentation

Segmentation as discussed here means to identify perceptual boundaries in a given melody. It allows us to break down a song into musically meaningful pieces such that each can be processed individually by an arrangement algorithm, which is required to reduce complexity. The input to such a segmentation algorithm is a representation of the melody and the output is a set of boundaries for segments. In this work the local boundary detection model (LBDM), developed by Emilios Cambouropoulos [Cam97, Cam01, MO02, Cam06], is adapted.

The LBDM considers three change features (differences between two subsequent notes in the melody), namely pitch intervals, inter-onset intervals (IOI) and rests. The design of the algorithm makes it also possible to include differences of another kind like harmonic intervals (distances between successive chords). Between consecutive feature values a degree-of-change function is computed and then the sum of this degree-of-change function of neighboring intervals is multiplied by the absolute feature value of the current interval. This function is then normalized

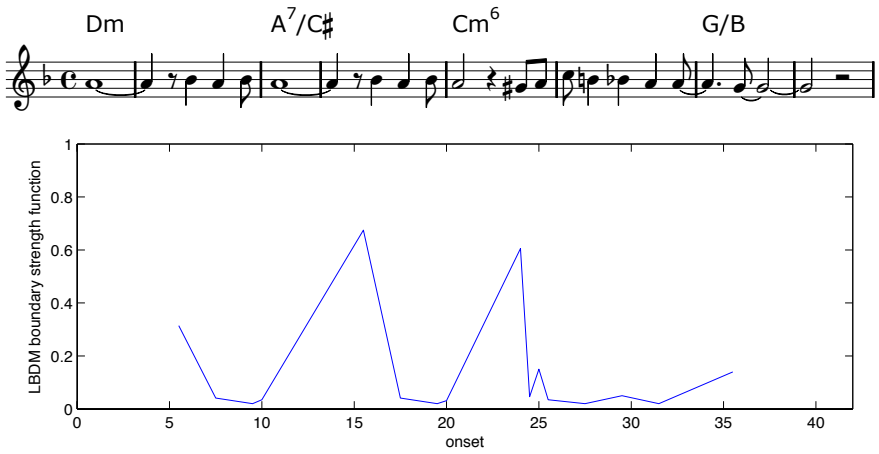


Figure 2.4: The LBDM applied to the first 8 bars of the song "How Insensitive" by Antonio Carlos Jobim. The blue line shows the resulting LBDM boundary-strength function.

and the three traces are combined using a weighted average. Local maxima in the resulting one-dimensional boundary-strength function represent potential segment boundaries. The detailed algorithm can be found in Section 5.1, Algorithm 3. An example of the boundary-strength function is depicted in Figure 2.4, where the LBDM is applied to the first 8 bars of the song "How Insensitive". Using the information about the boundary strength given by the LBDM an optimization algorithm finds segments, such that their lengths do not vary too much. For this purpose all possible segments are represented by a tree, with the full melody as the root node. The boundary-strength values determine where the segment nodes are split. A weighting function evaluates the segment nodes by their length and the optimal combination of segments is found by taking the maximum node on the path from each leaf node to the root. The optimization algorithm is described in Section 5.1, Algorithm 4.

CHAPTER 3

Algorithms and data representation

In this chapter the basic concepts of the technology that is used in the prototype implementation are introduced. The first Section 3.1 deals with music notation systems and representation thereof in the computer. Lead-sheets are used as the input to the arrangement algorithm presented in part II and tablature allows to display guitar fingering along with the arrangement. A format for symbolic music that represents both is MusicXML, which is described thereafter. The next sections describe genetic algorithms and constraint satisfaction problems. The arrangement process is basically a GA that evolves full arrangements, already including guitar fingering information. Whenever more than one note is played simultaneously multiple constraints apply to the fingering of the chord [Rad06]. Each such instance can be modeled as a constraint satisfaction problem (CSP) and the solution of which provides feasible fingering positions for each chord note.

3.1 Music notation

There are 3 basic music notation systems used in this work. Modern western music notation is used within lead sheets, which we assume that the reader is familiar with. Lead sheets are a good way to tightly hold all information that is used as an input to the arranging system (melody and chord symbols). The resulting arrangements will be notated again in modern western music notation together with the so-called tablature, that is able to display and store guitar fingering. Luckily there exists a data format that supports all the used notation forms (MusicXML), which is described thereafter.

3.1.1 Lead sheets

Lead sheets are a common music notation form especially in Jazz and popular music. As you can see in Appendix A, they usually contains 3 types of information:

- Melody (in modern western music notation)

- Harmony (in chord symbols, above the melody staff)
- Lyrics (optional, below the melody staff)

In some cases lead sheets contain also suggestions for accompaniment or tempo, however the main purpose of lead sheets is to give the musician enough information to improvise, perform or arrange the piece. In Appendix A lead sheets of famous Bossa Nova songs by Antonio Carlos Jobim are depicted. They are available at an online collection of lead sheets called Wikifonia[1] [Fou10] where users can upload and share music. There are also printed collections of lead sheets called "Real book" or "Fake book" which are published by several vendors. The included songs are sometimes referred to as "Jazz Standards", which underlines their importance to Jazz musicians.

3.1.2 Tablature

Tablature is a common notation system amongst guitarists and other string instrument players. The example in Figure 3.1 consists of two staffs; the first shows the part played by the guitar in modern western music notation, the second indicates fingering positions in tablature format. The tablature staff consists of one line for each string on the instrument, starting (top-down) with the highest pitched one. The numbers on those lines indicate the fret positions, but there are also other closely guitar-related symbols like bends, hammer-ons, pull-offs, slides, etc.

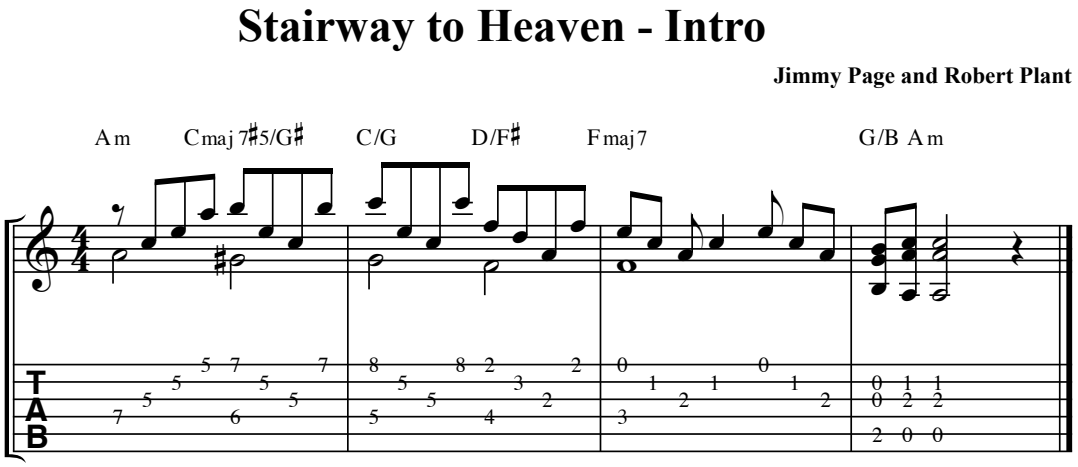


Figure 3.1: An example tablature. (Guitar intro from "Stairway to Heaven" by Jimmy Page and Robert Plant)

The score in Figure 3.1 shows the beginning of the song "Stairway to Heaven" by the british band Led Zeppelin as played by Jimmy Page. It is one of the classic benchmark songs for rock guitar players. Tablature is in general easier to read than modern western music notation,

[1] http://www.wikifonia.org

that is why the format is very popular amongst hobby musicians, that cannot or do not want to read "normal" scores. However, to show all the information necessary to perform a song it is indispensable to include also the part in modern western music notation, because for example the representation of rhythmic information is not well defined in tablature. To overcome this weakness people often include stems, flags and beams in tablature instead of adding another staff.

3.1.3 MusicXML

MusicXML [Goo06] is an open format for sheet music developed by the company Recordare LLC[2]. The design goals were to make an internet-friendly and human-readable (thus XML) interchange format that would be suitable for a lot of different applications – not only notation, but also optical music recognition (OMR), sequencing, digital libraries – and at the same time supported by all major vendors thereof. The DTD[3] and XSD[4] definitions of MusicXML (available under a royalty-free license) along with tutorials, examples and related publications can be found in [LLC10]. As mentioned earlier MusicXML supports all notation forms related to guitar arrangement of lead sheets.

Listing 3.1: MusicXML file example; header information, metadata.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD␣MusicXML␣2.0␣Partwise//EN" "http://www.musicxml.
       org/dtds/partwise.dtd">
3  <score-partwise version="2.0">
4          <movement-title>Stairway to Heaven - Intro</movement-title>
5          <identification>
6                  <creator type="composer">Jimmy Page and Robert Plant</creator>
7          </identification>
8          <part-list>
9                  <score-part id="P1">
10                         <part-name>Guitar</part-name>
11                         <part-abbreviation>Gtr.</part-abbreviation>
12                         <score-instrument id="P1-I2">
13                                 <instrument-name>Guitar</instrument-name>
14                         </score-instrument>
15                 </score-part>
16                 <score-part id="P2">
17                         <part-name>Guitar</part-name>
18                         <part-abbreviation>Gtr.</part-abbreviation>
19                         <score-instrument id="P2-I3">
20                                 <instrument-name>Guitar</instrument-name>
21                         </score-instrument>
22                 </score-part>
23         </part-list>
```

The Listing 3.1 shows the beginning of the MusicXML file corresponding to the music depicted in Figure 3.1. You can see a standard XML header referencing the MusicXML partwise DTD. The two different top-level elements (`score-partwise` and `score-timewise`) determine the overall structure of parts and measures. If `score-partwise` is chosen, a part is defined first, containing all the measures. The `score-timewise` implements the exact opposite ordering. In the example in Listing 3.1 the 2 parts are defined in the part-list element (they are both guitar parts – one for standard staff and one for tablature staff).

[2] `http://www.recordare.com/`

[3] `http://www.recordare.com/dtds/index.html`

[4] `http://www.recordare.com/xsd/index.html`

Listing 3.2: MusicXML file example; notes, rests and chord symbols.

```
<part id="P1">
        <measure number="1">
                <attributes>
                        <divisions>2</divisions>
                        <key>
                                <fifths>0</fifths>
                                <mode>minor</mode>
                        </key>
                        <time>
                                <beats>4</beats>
                                <beat-type>4</beat-type>
                        </time>
                        <clef>
                                <sign>G</sign>
                                <line>2</line>
                        </clef>
                        <transpose>
                                <diatonic>0</diatonic>
                                <chromatic>0</chromatic>
                                <octave-change>-1</octave-change>
                        </transpose>
                </attributes>
                <harmony>
                        <root>
                                <root-step>A</root-step>
                        </root>
                        <kind>minor</kind>
                </harmony>
                <note>
                        <rest/>
                        <duration>1</duration>
                        <voice>1</voice>
                        <type>eighth</type>
                </note>
                <note>
                        <pitch>
                                <step>C</step>
                                <octave>5</octave>
                        </pitch>
                        <duration>1</duration>
                        <voice>1</voice>
                        <type>eighth</type>
                        <stem>up</stem>
                        <beam number="1">begin</beam>
                </note>
                ...
```

In the Listing 3.2 it can be seen that parts contain measures. At the beginning of the staff attributes like key and time signatures are defined. Duration of notes in MusicXML are defined as fractions, where the denominator (number of divisions per quarter note) is defined in the `divisions` element and the `duration` element is specified within the `note` elements. It can also be seen what other elements are commonly defined per note: pitch, voice number, type (which should only affect the note's display and could also be deduced from the duration), stem and beam. Chord symbols are given before the note that they are associated with. They consist of a root note, kind (33 different kinds are supported) and bass note element (if applicable).

Listing 3.3: MusicXML file example; tablature.

```
<part id="P2">
        <measure number="1">
                <attributes>
                <divisions>2</divisions>
                <key>
                        <fifths>0</fifths>
                        <mode>minor</mode>
                </key>
```

```
                <clef>
                        <sign>TAB</sign>
                        <line>5</line>
                </clef>
                <staff-details>
                        <staff-lines>6</staff-lines>
                        <staff-tuning line="1">
                                <tuning-step>E</tuning-step>
                                <tuning-octave>2</tuning-octave>
                        </staff-tuning>
                        <staff-tuning line="2">
                                <tuning-step>A</tuning-step>
                                <tuning-octave>2</tuning-octave>
                        </staff-tuning>
                        <staff-tuning line="3">
                                <tuning-step>D</tuning-step>
                                <tuning-octave>3</tuning-octave>
                        </staff-tuning>
                        <staff-tuning line="4">
                                <tuning-step>G</tuning-step>
                                <tuning-octave>3</tuning-octave>
                        </staff-tuning>
                        <staff-tuning line="5">
                                <tuning-step>B</tuning-step>
                                <tuning-octave>3</tuning-octave>
                        </staff-tuning>
                        <staff-tuning line="6">
                                <tuning-step>E</tuning-step>
                                <tuning-octave>4</tuning-octave>
                        </staff-tuning>
                        <staff-size>150</staff-size>
                </staff-details>
        </attributes>
        <note>
                <rest/>
                <duration>1</duration>
                <voice>1</voice>
                <type>eighth</type>
        </note>
        <note>
                <pitch>
                        <step>C</step>
                        <octave>4</octave>
                </pitch>
                <duration>1</duration>
                <voice>1</voice>
                <type>eighth</type>
                <stem>none</stem>
                <notations>
                        <technical>
                                <string>3</string>
                                <fret>5</fret>
                        </technical>
                </notations>
        </note>
        ...
```

In Listing 3.3 the beginning of the tablature part is shown. The basic structure is the same, only the `clef` element changes and the tuning of the string-lines is specified in the `attributes` element. The same notes as in the previous part have to be included again, specifying the string and fret number within the `notations` element.

3.2 Genetic algorithms

GAs are search and optimization algorithms inspired by biological evolution. Solutions to the optimization problem (*candidates*) are encoded in *chromosomes* (a set of parameters that define

a proposed solution to the optimization problem). The difference between candidate and chromosome is equivalent to the difference between *genotype* (genetic information) and *phenotype* (observable properties like morphology) in natural evolution. A *population* of chromosomes is evolved over *generations* to originate chromosomes with a hopefully higher *fitness* score. The first population is filled with randomly initialized chromosomes to cover a wide range in the search space. From this population chromosomes are selected (the ones with higher fitness score should be selected more likely) for *crossover* (combination of chromosomes). This process is repeated to fill a new generation of chromosomes. Sometimes also (the fittest) chromosomes from the previous generation are copied to the new generation unchanged (*elitism*). After that the population is subjected to random *mutation*. This cycle is repeated until a termination criterion is reached. For an overview on GAs and much more consult [Mit98].

In an example we analyze a GA that breeds the string `HELLO WORLD`. An implementation of this GA can be found in [Dc10].

3.2.1 Representation

The first thing that is needed for a GA to operate is the representation of the solution to the problem as a chromosome. In most cases it is an array or list of arbitrary type (to facilitate crossover and mutation operations), but also tree structures (genetic programming) or graphs are used. In our example the chromosome is an array of characters (fixed length: 11).

3.2.2 Fitness

A chromosome needs to be assigned a fitness score, which measures its quality and enables a comparison between candidates. In some problems (especially art related) a fitness function is hard to define and thus in some cases the user assigns it (interactive fitness function). In the example we measure the difference to the target string (by counting non-matching characters).

3.2.3 Operators

Several operators are used in the evolution process. They guide the flow of the GA targeted at improving the score but also keeping dispersion to not get caught in local optima.

Initialization

The initialization process fills the first generation of chromosomes with more or less random candidates, which ensures a uniform sampling of the search space. In the example implementation the characters in the chromosomes are assigned randomly to one of 26 characters of the alphabet or the whitespace character.

Selection

Selection is a process guided by chance that is supposed to select candidates proportional to their fitness score for crossover. Small samples of less fit chromosomes that are likely to be

selected along with fitter chromosomes keep the population diverse and prevent premature convergence. The most popular methods are *roulette wheel selection* or *tournament selection*. In the example roulette wheel selection is used, which selects candidates according to the following probability (f_i denotes the fitness score of the ith chromosome and N denotes the total number of chromosomes in the population):

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3.1}$$

Crossover

The crossover operation breeds 2 offsprings from 2 parent chromosomes. The procedure is dependent on the chromosome representation, but in general the parent chromosomes are cut at a random point and the new offsprings are made of one slice of each parent (see Figure 3.2). This procedure is called one-point crossover, but it can easily be extended to n-point crossover. Crossover is intended to produce on average fitter candidates for the new population.

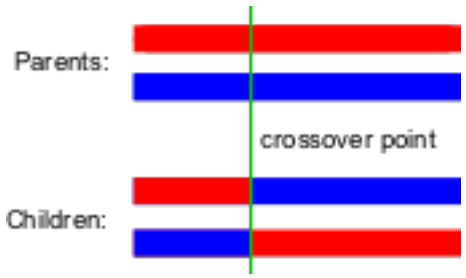


Figure 3.2: Single point crossover. (Rgarvage, "File:SinglePointCrossover.png" May 30, 2010 via Wikipedia, Creative Commons Attribution-ShareAlike 3.0 license)

Mutation

Mutation changes random elements in a chromosome which allows to introduce new ideas to the population. In an array-like chromosome representation, like in the example, each element is reinitialized in a random fashion with a certain probability (here $p_M = 0.02$). The purpose of mutation is to preserve and introduce diversity, such that local minima are avoided and the chromosomes of the population do not become too similar to each other.

Termination

It depends on the problem at hand which termination criterion makes the most sense. Typically a GA is terminated if a certain number of generations is reached or when no change (improvement) over a certain number of generations in the fitness score of the best chromosome in the population is observed. In the example we know exactly the fitness score we want to achieve, so the GA stops when the target string is found.

This is the output of the example GA (`StringsExample` from the Watchmaker framework in [Dc10]). It uses a population size of 100 and 5% elitism (which means that the fittest 5 chromosomes are retained in the next population). One can see beautifully how the chromosomes converge to the target string, which is reached in the 33rd generation:

```
Generation 0: HHKTHOFZVLT
Generation 1: YQLHLQQOYB
Generation 2: VAXXOYJOOKD
Generation 3: VAXXOYJOOKD
Generation 4: HGXROKXVRFW
Generation 5: HGXROZIXVND
Generation 6: WEFL IWPDAF
Generation 7: HGITBCWPDLK
Generation 8: HGXROVWPDLK
Generation 9: HGXLYYJOOKD
Generation 10: HBLLYYJOOKD
Generation 11: JEFLOYJOOQD
Generation 12: DEQROKWOOKD
Generation 13: HBLLYYJODLK
Generation 14: HEQROXWOOKD
Generation 15: HHLLYYWOOKD
Generation 16: HHLLYKWOOKD
Generation 17: HHLLYYWOOKD
Generation 18: HBLLOKWOOZD
Generation 19: HBLLOKWORMC
Generation 20: HEOLYKWORBD
Generation 21: HHLLYYWORBD
Generation 22: JELLOKWORBD
Generation 23: HELLOKWOOKD
Generation 24: HELLOUWORKD
Generation 25: HELLOOWORQD
Generation 26: HELLOAWORLC
Generation 27: HELLOAWORLU
Generation 28: HELLOOWORYD
Generation 29: HELLOAWORLD
Generation 30: HELLOAWORLD
Generation 31: HELLOAWORLD
Generation 32: HELLOVWORLD
Generation 33: HELLO WORLD
Evolution result: HELLO WORLD
```

3.2.4 Constraint handling in GAs

An open research topic is the handling of constraints in GAs (how can the generation of invalid chromosomes be avoided), which is relevant to the problem at hand (not all combinations of guitar fingerings are possible). The easiest solution would be to assign a penalty in the fitness score of an invalid candidate (or even 0 or ∞, which is called death-penalty). Another solution is to avoid generating invalid chromosomes in the initialization, crossover and mutation operations. A good overview on this subject can be found in [Coe02].

3.3 Constraint satisfaction problems

A constraint satisfaction problem consists of variables, a domain for each variable (finite set of possible values) and a set of constraints. The goal is to find an assignment of values to all variables that satisfies all given constraints. In the situation where CSPs are used in this thesis the variables are chord notes and the corresponding domains consist of the possible fingered positions for each note. To simultaneously played notes constraints like "only one note can be played on a given string" apply, so when generating the first population in the arrangement GA all such CSP instances are solved using the algorithms described here. CSPs are defined formally in [Kum92] as follows:

Definition 3. *A CSP $P = \{X, D, C\}$ is defined by a set of variables, $X = \{X_1, X_2, \ldots, X_n\}$, and a set of constraints, $C = \{C_1, C_2, \ldots, C_m\}$. Each variable X_i has a domain D_i of possible values. Each constraint C_i involves a subset of the variables and specifies the allowed combinations of values for the current subset. An assignment that satisfies all constraints is a* consistent *assignment, and a* solution *to a CSP is an assignment to all the variables, such that all constraints are satisfied.*

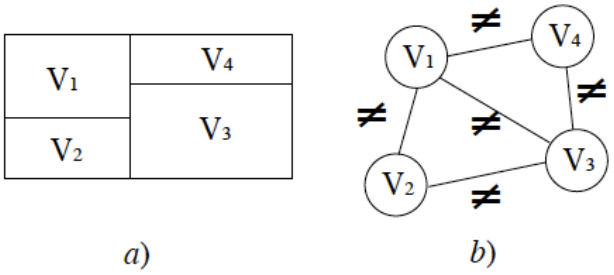


Figure 3.3: The map coloring problem (a): Each region $V_1 \ldots V_4$ has to be painted with one color such that no two neighboring regions have the same color; (b) shows the equivalent constraint graph. See Figure 3.9 from [Rad06].

A CSP can be visualized using a *constraint graph*, where the vertices represent the variables and an edge (or arc) represents a constraint between two variables. For one example see Figure 3.3. One step to solving a given CSP is to eliminate all values from all domains that will never

be part of the final solution. This leads hopefully to a simplified problem which can be solved more efficiently. This technique is called *constraint propagation*. The resulting CSP satisfies the properties of *arc-consistency*, which is defined in [Rad06] in the following way:

Definition 4. *Given a CSP $P = \{X, D, C\}$ with $C_{ij} \in C$, a variable X_i is* arc-consistent *relatively to X_j if and only if $\forall d_i \in D_i$, $\exists d_j \in D_j$ such that $(d_i, d_j) \in C_{ij}$. The arc defined by $\{X_i, X_j\}$ is arc-consistent if and only if X_i is arc-consistent relatively to X_j and X_j is arc-consistent relatively to X_i. A CSP is called* arc-consistent *if and only if all of its arcs are arc-consistent.*

Algorithm 1 AC-3 as defined in [Kum92].

```
1:  Q ← {(X_i, X_j) ∈ arcs(G), i ≠ j}
2:  while Q not empty do
3:     select and delete any arc (X_k, X_m) from Q
4:     revised ← false
5:     for all d_k ∈ D_k do
6:        found ← false
7:        for all d_m ∈ D_m do
8:           if (d_k, d_m) satisfies all constraints C_km ∈ C then
9:              found ← true
10:             break
11:          end if
12:       end for
13:       if ¬ found then
14:          remove d_k from D_k
15:          revised ← true
16:       end if
17:    end for
18:    if revised then
19:       Q ← Q ∪ {(X_i, X_k): (X_i, X_k) ∈ arcs(G), i ≠ k, i ≠ m}
20:    end if
21: end while
```

Algorithm 1 is called AC-3 and can be used to make a given CSP arc-consistent. It deletes every value $d_i \in D_i$ for which the condition given in Definition 4 doesn't hold. Of course if the domain of a variable X_i changes all previously revised arcs (X_j, X_i) have to be checked again. Instead of revising all domains that have been inspected already the AC-3 algorithm only revises domains that can possibly be affected, thus reducing complexity.

Given the now arc-consistent CSP it is still necessary to perform a search on the remaining values as the now revised domains only reduce search complexity, but the CSP can still have 0, 1 or more solutions [Kum92].

In this case the search can be performed by Algorithm 2 called Backtracking. This algorithm can be thought of as a depth-first-search (DFS) in a tree where the nodes correspond to an

Algorithm 2 Backtracking as defined in [RN03]

```
1: Backtrack(assignment, CSP)
2: if assignment is complete then
3:    return assignment
4: end if
5: X_i ← select unassigned variable from CSP
6: for all d_j ∈ D_i do
7:    if d_j is consistent with assignment then
8:       add {X_i = d_j} to assignment
9:       result ← Backtrack(assignment, CSP)
10:      if result ≠ failure then
11:         return result
12:      end if
13:      remove {X_i = d_j} from assignment
14:   end if
15: end for
16: return failure
```

assignment of a value to a variable and each level denotes a variable. As soon as an assignment is not found to be consistent the whole subtree is pruned and the search is continued in the level above.

Part II

Methodology

CHAPTER 4

Framework

In order to perform experiments on symbolic musical data one needs a framework to represent all the information that is dealt with. This chapter describes the framework on which the prototype implementation is based.

4.1 Goals

The prototype implementation is used to conduct first experiments in automatic arrangement based on lead-sheets. The major goal was to incorporate all available information in the input lead-sheet on a melodic, rhythmic and harmonic level. Regarding the results it was attempted to focus more on the playability aspect than on musical sophistication. Therefore the arrangement rules are kept rather simple:

1. Notes shall be inserted at fixed rhythmic positions, governed by a given rhythmic pattern.
2. The melody note shall be the highest note at any given chord in the arrangement.
3. The bass or root note, if present, shall be the lowest note at any given chord in the arrangement.
4. The arrangement shall not contain fingerings that are impossible to play (defined by the criteria found in [Rad06] and Section 5.2).
5. Both input lead-sheets and output arrangements shall be read/written in MusicXML format.

Because of point 1 the test songs have to belong to one musical genre that fits the rhythmic accompaniment pattern. A GA was chosen to optimize both the arrangement and fingering angle of the problem. This gives the search space an enormous size, but the nature of the instrument as well as bio-mechanical properties also pose constraints on the result (see also point 4). GAs are unconstrainted search techniques and are thus very bad at dealing with constraints [Coe02]. However various strategies to resolve this problem exist, see Section 5.2 for more details.

4.2 Design

Both the framework and the prototype have been implemented in the Java programming language. Various libraries are used to reduce the implementation effort: `JFreeChart 1.0.13` for data plotting, `Watchmaker-framework 0.7.1` for GAs, `Proxymusic 2.0` for reading and writing MusicXML and `commons-math 2.0` for fraction arithmetics. There are two main entry points to the framework (package `gArranger.main`). One program `GArranger-GenerateCandidates` has the task to generate candidates (see Section 5.2) and save them to disk, the other program `GArrangerGenerateSolutions` executes the GA for arrangement by reading the previously saved candidates from disk. This has the advantage that the part where candidates are randomly generated doesn't have to be executed every time a new arrangement is generated, which saves a significant amount of computation time during the experiments.

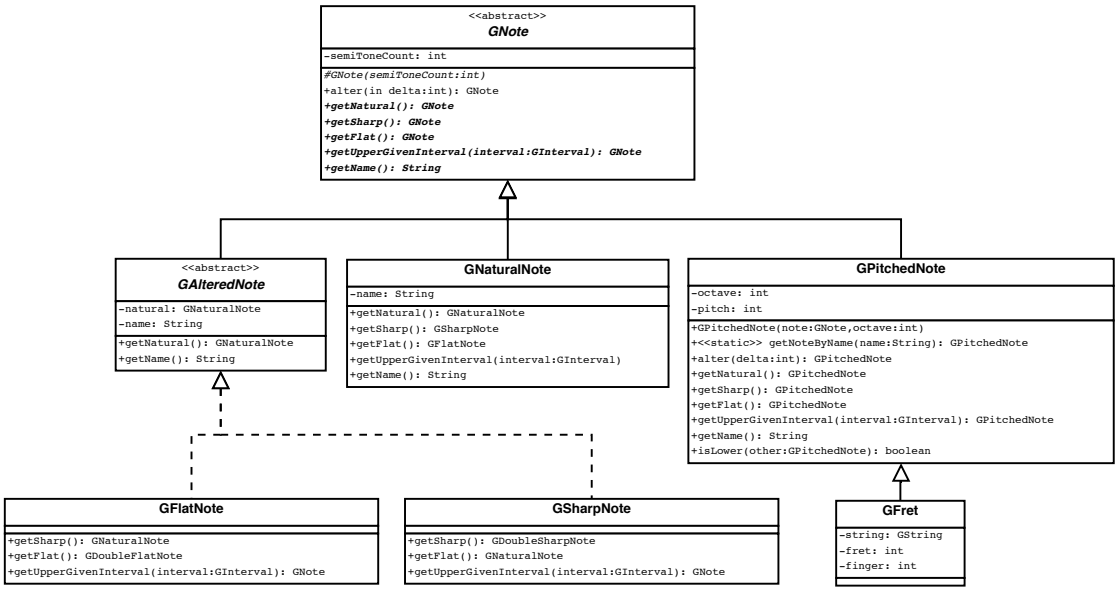


Figure 4.1: UML diagram of the classes representing notes.

The first kind of objects that are needed for a program to deal with symbolic music is notes. The object oriented design of all note representations is based on the work described in [Pac94] and the UML diagram can be found in Figure 4.1. In this prototype three different types of notes are required: notes representing pitch-classes (octave independent), pitched notes (octave dependent) and fingered positions. The base class for all types of notes is `GNote`. Together with its derived classes it represents pitch-classes (`GNote`) that are mostly needed to describe intervals and chords.

The `GNote` abstract base class is split into the classes `GNaturalNote` and `GAltered-Note`; the leaves of this inheritance tree are `GFlatNote` and `GSharpNote`. Implemented note arithmetics include computing neighboring natural, sharp and flat notes from any given note and returning the upper note of an interval, all in consideration of enharmonic equiva-

lence[1]. Pitched notes (`GPitchedNote`) are derived from `GNote` and occur in melodies. The additional information needed to pinpoint the specific pitch is the octave in which the note is played. The combination of pitch class and octave is also the foundation of the *scientific pitch notation* described in Definition 1. Finally the class representing fingered positions (`GFret`) is derived from `GPitchedNote`. As mentioned in Section 2.2 a pitched note alone is not a unique specification of a position on the fretboard of a guitar, therefore the guitar string, fret number and finger number as additional attributes are needed.

Other data structures that are based on `GNote` derived classes are `GInterval` and `GChord`, together they are assembled in the package `gArranger.data.harmony`. The main functionality of those classes is to transform the chord symbol in the lead sheet (e.g. C^{maj7}, $E\flat^{7\sharp 9}$) into a set of notes. The `GInterval` class mainly provides the building blocks for this process.

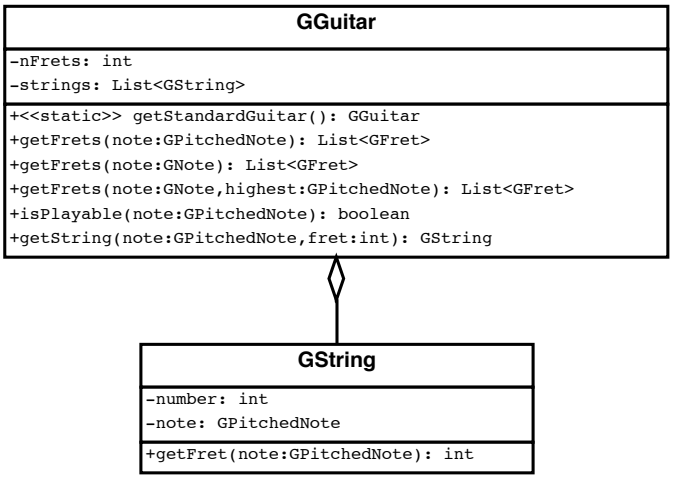


Figure 4.2: UML diagram of the classes representing the guitar.

The modeling of the guitar can be seen as an UML diagram in Figure 4.2. It consists of a list of strings and the number of playable frets. The factory method constructs a standard electric guitar as depicted in Figure 2.1 with 22 frets. A guitar string `GString` holds a number and a pitch. The methods in `GGuitar` can be used to compute possible frets for pitched notes and also pitch-classes as well as determine if a given note is playable with this guitar instance.

The last component for modeling symbolic music is rhythm. For this purpose the class `Event<T>` is created to assign position and duration properties to any timed object (can be chord symbols as well as notes in a melody). Position and duration are stored as fractions, which corresponds to the common way of denoting rhythmical information ("*sixteenth* note"), but also how duration is represented in MusicXML (see also Section 3.1). In various algorithms (segmentation for instance) event progressions are represented as a sequence of onsets and offsets (measure-independent format), which is not stored explicitly but computed on demand, accessible via methods (like `getOnsets`) in classes representing measures (`GMeasure`) and

[1]The implemented prototype is required to be able to differentiate between a minor sixth (C – A♭) and an augmented fifth (C – G♯), two different names for the same pitch difference. More on enharmonic equivalence can be found in `http://en.wikipedia.org/wiki/Enharmonic`

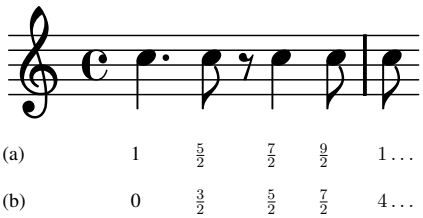


Figure 4.3: A demonstration of the difference between position (a) and onset (b) representation.

sequences of measures (`GPart`). Figure 4.3 demonstrates the difference between the two formats. The line (a) shows the position information stored in a `Event<T>` object. The counter starts at 1 and it is reset at each new measure. Onsets however start at 0 and continue across measure boundaries, thus this representation does not depend on a measure-context.

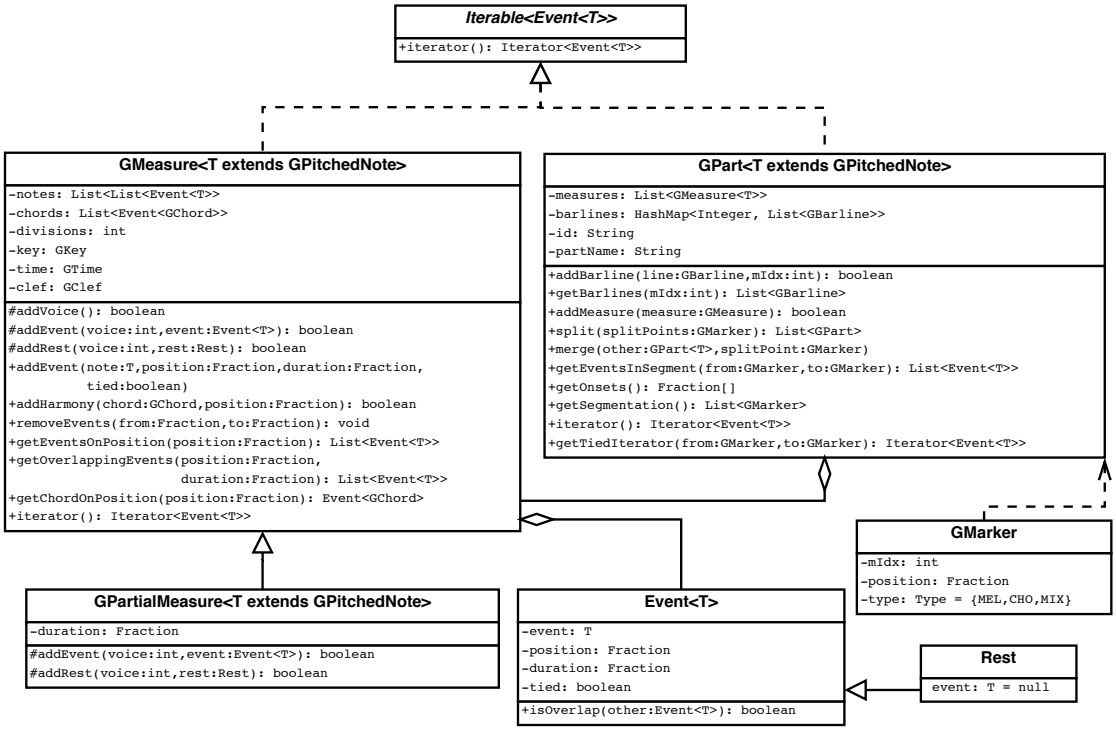


Figure 4.4: UML diagram of the classes representing higher level data structures.

The UML diagram of classes mentioned above can be seen in Figure 4.4. `GMeasure` is implemented in a self-organizing way. For example: a measure can have multiple voices, which is a way to handle polyphonic passages on a single staff. The `GMeasure` object decides transparently when it is necessary to open new voices according to the following rule: If there is a note in the same position, having the same duration or free space (rest) in any of the existing voices the

new note will be added there, otherwise a new voice has to be created. `GMeasure` also stores a chord symbol track. The protected methods `addEvent(voice, event)` and `addRest` are low-level implementations that are overridden in `GPartialMeasure` and called in the public interface method `addEvent(note, position, duration, tied)`. The `GMeasure` class also manages `Rest` objects (that are derived from `Event<T>`) within voices, thus the user never has to care about inconsistent or invalid states of the measure. Rests are also not included when the iterator is used to read event data from a `GMeasure` object.

More methods to query/manipulate data are also available that are mostly called by the wrapping class `GPart`, like `removeEvents` or `getEventsOnPosition`. `GPart` maintains a sequence of `GMeasure` objects and offers higher-level access/manipulation methods. Positions within `GPart` objects are denoted by `GMarker` objects, that include a measure index. Further use of this class in association with the method `getSegmentation` is also discussed in Section 5.2. `GPart` objects can also be iterated either with consideration of ties or without. Important operations for segmentation (see Section 5.1) and crossover (see Section 5.2) are `split` and `merge`.

CHAPTER 5

Arrangement algorithm

This chapter describes the workings of the main arrangement intelligence. The sequence is as follows:

1. The melody and chord symbols are read from the input lead sheet.
2. The melody is segmented into n parts.
3. Each part is processed by the arrangement GA separately.
4. The result parts are merged into the final output arrangement.

5.1 Segmentation

Breaking the song into several pieces before proceeding with arrangement by a GA is necessary to reduce the search space, as also suggested in [Tuo06], Section 4.4.2 or [TP06b]. A break is best introduced when the performer is free to move his/her hand. Also criteria on a semantic level are relevant, such as end of phrases or motives. Both considerations are well addressed in the local boundary detection model (LBDM) by Cambouropoulos [Cam01], which is discussed in the subsequent subsection.

All segmentation related classes are assembled in the package `gArranger.segmentation`.

5.1.1 Local boundary detection model

The LBDM is based on two rules [Cam01]:

Change rule Boundary strengths shall be proportional to the degree of change between two consecutive intervals.

Proximity rule The boundary assigned to the larger interval is proportionally stronger.

The feature values on which the LBDM is based are differences between the consecutive notes of a melody on various levels: The implementation outlined in Algorithm 3 uses pitch intervals, inter-onset intervals (IOI) and rest intervals (the difference between current onset and previous offset). A big advantage of the LBDM is that it can easily be adapted to include other features such as chord progressions or double bar-lines.

After feature extraction a degree-of-change function is computed on all three levels (implementing the *change rule*) and then multiplied by the absolute value of each interval (implementing the *proximity rule*), so if in two instances the degree-of-change function has equal values (as in note successions sixteenth to eighth and quarter to half) the boundary value on the second transition will be greater. The degree-of-change function is denoted as R_k in Algorithm 3, and as S_k after the multiplication with the absolute interval value P_k. This trace is then normalized to the range $[0, 1]$. The final boundary function b is a weighted average over the strength sequences S_k, here the following weight values are used: $w_{pitch} = 0.15$, $w_{ioi} = 0.5$ and $w_{rest} = 0.35$. Local maxima in the final boundary function indicate suitable segment boundaries and the optimal segmentation is searched in a subsequent step.

Algorithm 3 Algorithm computing the local boundary detection model as defined in [Cam01].

$n \leftarrow$ number of notes - 1
for $i = 1 \ldots n$ **do**
  $P_{pitch}(i-1) \leftarrow |pitch(i) - pitch(i-1)|$
  $P_{ioi}(i-1) \leftarrow onset(i) - onset(i-1)$
  $P_{rest}(i-1) \leftarrow onset(i) - offset(i-1)$
end for
for all $k \in \{pitch, ioi, rest\}$ **do**
  $R_k(i, i+1) \leftarrow \begin{cases} \frac{|P_k(i) - P_k(i+1)|}{P_k(i) + P_k(i+1)} & \text{if } P_k(i) + P_k(i+1) \neq 0 \\ 0 & \text{if otherwise} \end{cases}$
  $S_k(i) \leftarrow P_k(i) \cdot (R_k(i-1, i) + R_k(i, i+1))$
  $S_k \leftarrow \frac{S_k - \min(S_k)}{\max(S_k) - \min(S_k)}$
end for
$b(i) \leftarrow \sum_{k \in \{pitch, ioi, rest\}} w_k \cdot S_k(i)$

5.1.2 Finding optimal segments

The LBDM provides a quantification of the potential of each interval to become a segment border, however the optimal combination of segment borders to form the final segmentation has still to be determined. The criterion to define this optimality is based on the segment length. It shall be around the same value for all segments and not vary too much. Therefore a function is defined to assess segments. This weight function is derived from the gaussian function shown in Equation 5.1 (the argument x denoting the segment length). Without the normalization factor that is part of the gaussian probability density function this weight takes values in the range $(0, 1]$

and reaches its maximum when $x = \mu$.

$$f(x) = e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{5.1}$$

Algorithm 4 Algorithm that searches optimal segments from LBDM data.

$borders \leftarrow \{(i, b(i)) : b(i-1) < b(i) \wedge b(i+1) < b(i)\}$
$borders \leftarrow borders \cup \{(-1, 1.0), (n, 1.0)\}$
$tree \leftarrow$ buildTree$(borders)$
for all $leaves \in tree$ **do**
  $max \leftarrow \{node_i \in$ pathToRoot$(leaf) : w_{node_i} > w_{node_k} \wedge i \neq k\}$
  $segm \leftarrow segm \cup max$
end for
for all $nodes \in segm$ **do**
  $segm \leftarrow segm - (segm \cap$ pathToRoot$(node))$
end for
for all $nodes \in segm$ **do**
  if $w_{node_i + node_{i+1}} > w_{node_i} + w_{node_{i+1}}$ **then**
    $segm \leftarrow segm - node_i - node_{i+1} \cup (node_i + node_{i+1})$
  end if
end for

With the help of this function we can find the optimal segmentation using the method shown in Algorithm 4. Figure 5.1 visualizes this process by means of an example. First the local maxima of the LBDM function are extracted as potential segment borders. From this initial segmentation a tree is built (the root node being the segment that represents the whole song). Each node is further broken down according to the maximum boundary values in the segment. This process is repeated for every new segment node until there are no more local maxima left. Then the path from each leaf node (corresponding to the initial segments built directly from all LBDM maxima) to the root is searched for the segments having the highest weight according to the function in Equation 5.1. Those are stored in the list $segm$.

Subsequently all segments are eliminated from $segm$ that are already included in another segment in the same list, this is the case if there is a common element in $segm$ and the path from the current node to the root. To avoid over-segmentation a second pass merges neighboring segments if the merged segment has a bigger weight than the sum of the two single weights. In the example shown in Figure 5.1 this is the case with the boundary marked in green on the plot, which is eliminated due to this process.

We now want to specify an acceptable range of segment lengths, thus determining suitable values for parameters μ and σ in Equation 5.1. The parameter values shall be chosen such that segment lengths in the acceptable range are assigned a weight of $\frac{1}{2}$ or higher. The center of this range will be the parameter μ, but we need to find out what value to assign to parameter σ such

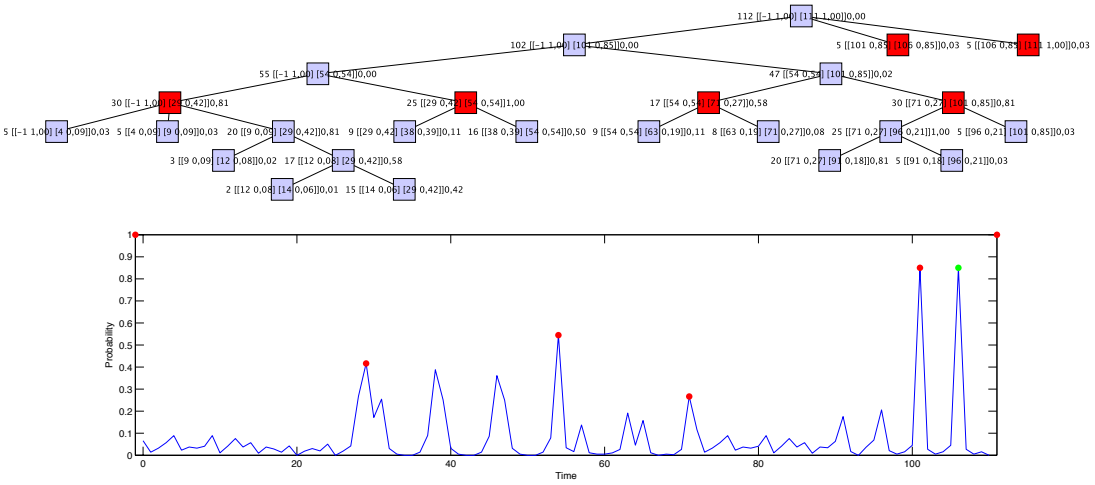


Figure 5.1: This figure shows how the segmentation tree (above) is built from the LBDM boundary-strength function (in the plot below). The root node represents the whole song, in this case "The Girl from Ipanema". Each segment is split (by adding child nodes) on the maximum boundary-strength value (in the case of the root node it is the 2 peaks at x-axis values 101 and 106, so it has 3 child nodes). The segmentation tree in this figure is not complete because of the limited space. Each node in the tree is annotated with the following information: *segment length* [[*left boundary*] [*right boundary*]] *weight*; the boundary being composed of (x, y) coordinates of the local maximum in the plot. The nodes marked in red correspond to the segment boundaries marked in red and green on the plot and represent the final segments. The boundary in green was removed due to the second pass merging procedure.

that the function 5.1 $f(x) \geq \frac{1}{2}$. So, if we solve Equation 5.1 with $f(x) = \frac{1}{2}$ for σ we get:

$$\sigma = \pm \frac{x - \mu}{\sqrt{-2 \log \frac{1}{2}}}. \tag{5.2}$$

According to Equation 5.2 the value of σ is only dependent on the deviation of the segment length from μ. So a new parameter δ is introduced that specifies the acceptable deviation from the mean segment length μ. This substitution can then be incorporated into Equation 5.1 to get the final weight function with two parameters μ and δ shown in Equation 5.3. The parameter values $\mu = 25$ and $\delta = 9$ have been determined experimentally. They mean that segment lengths around 25 are assigned a higher weight, especially if $x \in [16, 34]$, $f(x)$ will be greater or equal to $\frac{1}{2}$. Figure 5.2 depicts a plot of Equation 5.3 with the parameter values specified above.

$$f(x) = e^{\frac{\log \frac{1}{2}(x-\mu)^2}{\delta^2}} \tag{5.3}$$

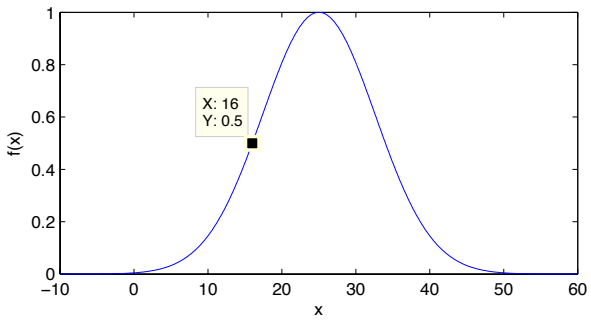


Figure 5.2: Illustration of the weight function as defined in Equation 5.3 with $\mu = 25$ and $\delta = 9$.

5.2 Genetic algorithm

Each lead sheet segment is processed by the arrangement GA. The optimization is already carried out in full "fingering" domain, this means that evolving candidates are represented as full arrangements including fingering. The only time where (chord) notes are selected or added to the candidate is when they are generated to fill the first population of the GA. The melody is always inserted fully with randomly chosen fingering and chord notes are added on the positions that are given by a rhythmic pattern. Hereby constraints that apply to simultaneously played notes are considered by solving a CSP on every chord, this makes sure that the first population only contains "valid" candidates (in terms of not violating any constraint). Constraints are also included in the fitness function where every violation causes the candidate to be penalized, because every mutation operation on a chord note is very likely to cause constraint violations. Mutation furthermore only changes the fingered position of a randomly selected note, it does not add or remove notes. Of course by combining candidates (crossover operation) the arrangement can be thinned out or enriched with chord notes. The fitness function optimizes mainly the fingering of the arrangement by computing a difficulty score (from [Rad06]) that means that the fingering within chords as well as transitions between chords and notes is evaluated. The main focus of this work is the playability aspect of arrangement that's why no musical constraints or attributes whatsoever are implemented in the fitness function.

The GA was implemented using the *Watchmaker Framework for Evolutionary Computation* version 0.7.1 by Daniel W. Dyer [Dc10]. It provides high-performance, multi-threaded and extensible classes that implement different operators and engines. There are no restrictions on candidate representations, its type is completely decoupled from the framework. If a custom implementation of evolutionary operators is needed, one can easily integrate own classes but also use provided standard algorithms. Operators that have been implemented for the arrangement task are a `CandidateFactory` and various `EvolutionaryOperator` classes (Crossover, Mutation and Evaluation), all of which are discussed in subsequent subsections.

5.2.1 Chromosome representation and generation

Evolved chromosomes represent full arrangements of lead sheet segments (including fingering – hence type `GPart<GFret>`) that were determined using the segmentation algorithm described above. There are two implementations that extend `AbstractCandidateFactory< GPart<GFret> >`. Those are needed because the process of generating candidates is executed independently from the rest of the GA execution (see Section 4.2). `GTabConstrainted-Factory` actually generates candidates that are usually written to disk by the program `GArrangerGenerateCandidates`. When the rest of the GA is executed those previously generated candidates have to be read from disk and provided to the other evolutionary operators, for this purpose `GTabStoredFactory` is used. The inputs to the candidate generation algorithm are the melody `GPart<GPitchedNote>` that also already includes the chord symbols and a pattern `GPattern` that determines on which metrical positions chord notes can be inserted.

First the melody notes are included in the candidate `GPart`. Here all possible fingered positions $(string, fret, finger)$ for the given note are computed and a randomly chosen one is inserted. It is also taken care that tied notes end up with the same 3-tuple. Then for each metrical position in the given `GPattern` a random subset of the notes in the current chord is computed (always containing the bass/root note). Now the CSP of assigning fingered positions to chord notes is solved using first Algorithm 1 to make the CSP arc-consistent and then finding a random but valid assignment with the backtracking Algorithm 2 in Section 3.3.

There are however a few refinements to this process: The melody note has already been assigned a fingered position, but it also needs to contribute to the CSP, because it can limit the domain of other variables (notes). So, the melody note together with possible other already inserted, simultaneously played notes are also part of the CSP variables, but with a domain limited to one element. Furthermore two new constraints are introduced: The melody note needs to be the highest in the chord according to point 2 in Section 4.1 and the bass or root note the lowest (point 3). Of all possible fingerings, if any, a random one is selected as final output. This is achieved by shuffling the remaining elements in the domains after making the CSP arc-consistent (Algorithm 1). The first found assignment by the backtracking Algorithm 2 is taken.

Constraints are represented as implementing classes of the `GConstraint` interface, which specifies a method `boolean isSatisfied(GFret fret1, GFret fret2)`. The constraints are checked on every pair of fingered positions that are played simultaneously (thus this method is called on every pair of fingered positions). However this representation is too limited to cover for example a proper formulation of the barré constraint: The same finger on the same fret is allowed if only notes on the right side of the barré finger are played additionally. Checking constraints of that kind would require access to the whole set of fingered positions, but the representation used here in the process of generating as well as evaluating candidates is powerful enough for the constraints listed hereafter. See also Section 3.3.1 in [Rad06].

OneNotePerString It is physically impossible to play more than one note at once on the same string.

HigherNoteOnLowerString Higher notes shall be played on lower strings (assuming strings are numbered from highest to lowest pitch).

NoHorizontalOverlap The order of finger numbers shall be the same as the order of fret numbers, e.g. two fingers cannot intersect. Furthermore barré chords are allowed with any finger (which is a major difference to [Rad06]).

MaxFingerSpan The fret number difference must not exceed a predefined maximum span between each distinct pair of fingers defined in table 5.1.

Table 5.1: Specification of the maximum finger span from [Rad06]

	index	middle	ring	pinky
index	-	2	3	4
middle	2	-	1	3
ring	3	1	-	1
pinky	4	3	1	-

If no solution to the CSP exists the process of generating (almost) random subsets of chord notes is repeated until a solution is found. Failed subsets are stored for the purpose of not repeating unnecessary computations. If no possible chord note combination is found the bass/root note is not a required component any more. This way in the worst case no chord note at all is inserted if the initial condition (melody note) made the satisfaction of all constraints impossible. All inserted chord notes get the duration of the melody note, if there is one, otherwise the `GPattern` object determines the duration (difference between subsequent and current metrical position). All note durations are limited by the duration of one quarter note (to enable long melody notes with underlying chord accompaniment pattern).

The CSP was implemented in this step to ensure a first population of valid (in terms of the satisfaction of all constraints) chromosomes. During the remaining execution of the GA it is very likely that invalid chromosomes are produced by mutation, so a strategy to handle constraints throughout the remaining process is needed nevertheless. More on this subject can be found in the next subsection.

5.2.2 Fitness function

As stated in Section 4.1 the main goal is to optimize playability of the resulting arrangement. This aspect is covered using the ideas presented in [Rad06], where the guitar fingering problem (assigning fingered positions to a given score) is modeled and a solution is proposed. This model consists of a graph at which the vertices correspond to fingered positions and the edges model transitions between them. The author then uses a difficulty score for transitions between fingered positions to solve the guitar fingering problem by finding the minimal path through the graph. The same difficulty score can therefore easily be used to estimate the overall playability of a given arrangement, which is exactly the task at hand.

The implemented fitness function is shown in Equation 5.4. The first component f_1 denotes playability (or rather difficulty - the lower the value the better). If f_1 was the only objective, the

final arrangement would end up with less notes, as any additional note also increases difficulty. Therefore f_2 is added, which acts as a counter balance and grows proportional to the number of notes in the arrangement. Since it behaves inversely to f_1 it is placed below the fraction line. n denotes the number of notes in the arrangement and has been introduced to make the fitness score independent of the arrangement size. Both components are eventually weighted by factors α_1 and α_2 and summed up, which is the easiest way to achieve multi-objective optimization. More on the detailed computation of f_1 and f_2 can be found subsequently.

$$f = \alpha_1 \frac{f_1}{n} + \alpha_2 \frac{n}{f_2 + 1} + penalty \tag{5.4}$$

Turning to the difficulty score and its computation: according to [Rad06] (Section 3.2) three different types of movements in guitar performance can be distinguished:

melody (MEL) The performer plays one note at a time.

chord (CHO) The performer plays more than one note at a time.

mixed (MIX) The performer holds at least one note and plays others at the same time.

All three cases are defined and exemplified in Figure 5.3. In summary, a monophonic passage without any simultaneously played notes is classified as a MEL block. If multiple notes are played at the same time, starting and ending at the same time, the CHO case is applied. Everything else involving overlapping notes (one or more held notes with a sequence of chords or a melody played at the same time) is a MIX case. Algorithm 5 is used to implement the rules given in Definitions 5 – 10 in Figure 5.3. This segmentation is not only used to compute the difficulty score according to [Rad06] but also to find suitable split points for crossover. Moreover the second component of the fitness score f_2 is computed as the number of notes within CHO or MIX blocks.

Overall difficulty f_1 is composed of difficulty within each MEL, CHO or MIX block (intra block score) and difficulty of transition between blocks (inter block score). The same basic function given in Equation 5.5 from [Rad06] is employed within each inter or intra block score. It is based on the two directions of hand movement on the fretboard – *horizontally* (along the fretboard) and *vertically* (across the fretboard). Therefore difficulty score between two fingered positions $p = (string_p, fret_p, finger_p)$ and $q = (string_q, fret_q, finger_q)$ is defined as follows in [Rad06]:

$$weight_{(p,q)} = along_{(p,q)} + across_{(p,q)} \tag{5.5}$$

This is further decomposed to:

$$along_{(p,q)} = fret\_stretch_{(p,q)} + locality_{(p,q)} \tag{5.6}$$

$$across_{(p,q)} = vertical\_stretch_{(p,q)} \tag{5.7}$$

$fret\_stretch$ is a function of the directed distance measure $\Delta fret$, which is defined as

$$\Delta fret_{(p,q)} = fret_q - fret_p. \tag{5.8}$$

Algorithm 5 Algorithm to segment an arrangement into MEL, CHO and MIX blocks.

$blocks \leftarrow \emptyset$
$block \leftarrow$ new MEL block
for all $notes \in arrangement$ **do**
  if $type_{block} = MEL$ **then**
    if $onset_{prev} = onset_{curr} \wedge offset_{prev} = offset_{curr}$ **then**
      if $onset_{block} = onset_{prev}$ **then**
        $type_{block} \leftarrow CHO$
      else
        $blocks \leftarrow blocks \cup block$
        $block \leftarrow$ new CHO block
      end if
    else if $offset_{prev} \leq onset_{curr}$ **then**
      do nothing
    else
      if $onset_{block} = onset_{prev}$ **then**
        $type_{block} \leftarrow MIX$
      else
        $blocks \leftarrow blocks \cup block$
        $block \leftarrow$ new MIX block
      end if
    end if
  else if $type_{block} = CHO$ **then**
    if $onset_{prev} = onset_{curr} \wedge offset_{prev} = offset_{curr}$ **then**
      do nothing
    else if $offset_{prev} \leq onset_{curr}$ **then**
      $blocks \leftarrow blocks \cup block$
      $block \leftarrow$ new MEL block
    else
      $type_{block} \leftarrow MIX$
    end if
  else if $type_{block} = MIX$ **then**
    if $onset_{curr} \geq offset_{prev}$ **then**
      $blocks \leftarrow blocks \cup block$
      $block \leftarrow$ new MEL block
    end if
  end if
end for
$blocks \leftarrow blocks \cup block$

Positive values indicate movement towards the body of the instrument, negative values the other way. The actual $fret\_stretch$ function is now dependent on the involved finger pair, as difficulty varies with fingers and movement direction. Figure 5.4 shows an example for the $fret\_stretch$ functions for the index finger. [Rad06] defines all possible $fret\_stretch$ functions in appendix A.

The $fret\_stretch$ function employs the same concept of the *comfortable span* as the constraint *MaxFingerSpan* defined above in table 5.1. The distance of the comfortable span is assigned the minimum value of 0.5. The second component of the $along$ score is $locality$, which accounts for the fact that playing near the neck is easier than playing near the body of the instrument.

$$locality_{(p,q)} = \alpha \cdot (fret_p + fret_q) \tag{5.9}$$

[Rad06] suggests a value of $\alpha = 0.25$ for the moderately skilled player, but it can be adjusted accordingly. One special case that has to be considered are empty strings. If one of the fingered positions denotes an empty string no $fret\_stretch$ is added and $locality_{(p,q)} = \alpha \cdot fret_q$.

Next, $across$ is defined. It depends on the string distance $\Delta string$ and as for $fret\_stretch$ a comfortable span can be derived. This is evident from the function $vertical\_stretch$ as defined in [Rad06] and below.

$$\Delta string_{(p,q)} = |string_q - string_p| \tag{5.10}$$

$$\Delta finger_{(p,q)} = |finger_q - finger_p| \tag{5.11}$$

$$vertical\_stretch_{(p,q)} = \begin{cases} 0.25 & \Delta finger_{(p,q)} = 0 \ \wedge \ \Delta string_{(p,q)} = 0 \\ 0.25 & \Delta finger_{(p,q)} = 1 \ \wedge \ (\Delta string_{(p,q)} = 0 \\ & \qquad \vee \Delta string_{(p,q)} = 1) \\ 0.25 & \Delta finger_{(p,q)} = 2 \ \wedge \ (\Delta string_{(p,q)} = 0 \\ & \qquad \vee \Delta string_{(p,q)} = 2 \\ & \qquad \vee \Delta string_{(p,q)} = 3) \\ 0.25 & \Delta finger_{(p,q)} = 3 \ \wedge \ (\Delta string_{(p,q)} = 0 \\ & \qquad \vee \Delta string_{(p,q)} = 4) \\ 0.5 & \text{otherwise} \end{cases} \tag{5.12}$$

The function $weight_{(p,q)}$ is directly used as a MEL intra block score (applied to every pair of subsequent notes). We will now demonstrate how to evaluate the intra block score of the example MEL block depicted in Figure 5.5a. The first two fingered positions are $p = (3, 3, 1)$ and $q = (3, 5, 3)$ and this is how the $weight$ function is computed:

$$\begin{aligned} weight_{(p,q)} &= along_{(p,q)} + across_{(p,q)} \\ weight_{(p,q)} &= fret\_stretch_{(p,q)} + locality_{(p,q)} + vertical\_stretch_{(p,q)} \\ weight_{(p,q)} &= 0.5 + (0.25 \cdot (3 + 5)) + 0.25 \\ weight_{(p,q)} &= 2.75 \end{aligned}$$

The $fret\_stretch$ function in this example can be evaluated by examining Figure 5.4. The finger pair involved in positions p and q is index-ring, so we have to look up the lower left function. $\Delta fret_{(p,q)} = 2$, so $fret\_stretch_{(p,q)} = 0.5$. $vertical\_stretch$ evaluates to 0.25

because $\Delta string_{(p,q)} = 0$ and $\Delta finger_{(p,q)} = 2$. The full intra block score is then simply computed by summing up all $weight_{(p,q)}$ function values for all consecutive pairs of fingered positions.

$$\begin{array}{rll}
weight_{((3,3,1),(3,5,3))} = & 0.5 + 0.25 \cdot (3+5) + 0.25 & 2.75 \\
weight_{((3,5,3),(3,6,4))} = & 0.5 + 0.25 \cdot (5+6) + 0.25 & 3.50 \\
weight_{((3,6,4),(2,4,2))} = & 0.5 + 0.25 \cdot (6+4) + 0.5 & 3.50 \\
weight_{((2,4,2),(3,6,4))} = & 0.5 + 0.25 \cdot (4+6) + 0.5 & 3.50 \\
weight_{((3,6,4),(3,5,3))} = & 0.5 + 0.25 \cdot (6+5) + 0.25 & 3.50 \\
weight_{((3,5,3),(3,3,1))} = & 0.5 + 0.25 \cdot (5+3) + 0.25 & 2.75 \\
weight_{((3,3,1),(4,6,4))} = & 0.5 + 0.25 \cdot (3+6) + 0.5 & 3.25 \\
\sum = & & \overline{22.75}
\end{array}$$

The same way as for MEL intra block score, $weight$ is used to assign a score to a CHO block:

$$weight_{(c_1,\ldots,c_n)} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} weight_{(c_i,c_j)} \tag{5.13}$$

Here the weight of every distinct pair of chord notes is summed up. The same score can be applied to MIX blocks, because they can for this purpose be treated like a sequence of CHO blocks. This means that every set of simultaneously played notes is merged to a CHO block, which requires the held notes being split at the same position as the moving notes in the MIX block. In this case also the transition weight is added that is defined hereinafter.

Considering the CHO block example in Figure 5.5b the same procedure as demonstrated above is employed:

$$\begin{array}{rll}
weight_{((4,0,0),(3,2,2))} = & 0 + 0.25 \cdot 2 + 0.25 & 0.75 \\
weight_{((4,0,0),(2,1,1))} = & 0 + 0.25 \cdot 1 + 0.25 & 0.50 \\
weight_{((4,0,0),(1,1,1))} = & 0 + 0.25 \cdot 1 + 0.25 & 0.50 \\
weight_{((3,2,2),(2,1,1))} = & 0.5 + 0.25 \cdot (2+1) + 0.25 & 1.50 \\
weight_{((3,2,2),(1,1,1))} = & 0.5 + 0.25 \cdot (2+1) + 0.5 & 1.75 \\
weight_{((2,1,1),(1,1,1))} = & 0.5 + 0.25 \cdot (1+1) + 0.5 & 1.50 \\
\sum = & & \overline{6.50}
\end{array}$$

As transitions between MEL and another MEL are not possible by Definition 5, MIX blocks are transformed into a sequence of CHO blocks and transition weights are symmetric, only MEL→CHO and CHO→CHO transitions have to be considered. The following method was proposed by Daniele Radicioni after inquiry by email, as transitions are not covered in [Rad06]:

$$trans\_weight_{(p,q)} = \begin{cases} \beta_1 \Delta fret_{(p,q)} & \Delta fret_{(p,q)} > 0 \\ -\beta_2 \Delta fret_{(p,q)} & \text{otherwise} \end{cases} \tag{5.14}$$

The factors β_1 and β_2 reflect the different difficulty of moving the hand towards the body or in opposite direction. Therefore they were set to $\beta_1 = 0.5$ and $\beta_2 = 0.3$. Otherwise only the hand repositioning is penalized. In case of CHO blocks where multiple notes can be used to compute $trans\_weight$ from, the fret of the index finger is chosen or estimated if not employed.

This is the way the transition weight is evaluated when going from the MEL block in Figure 5.5a to the CHO block in Figure 5.5b ($p = (4, 6, 4)$, $q = (2, 1, 1)$):

$$\begin{aligned} trans\_weight_{(p,q)} &= -\beta_2 \Delta fret_{(p,q)} \\ trans\_weight_{(p,q)} &= -0.3 \cdot (-5) \\ trans\_weight_{(p,q)} &= 1.5 \end{aligned}$$

Any CHO block, including the ones generated from MIX blocks is checked for compliance with the same constraints as discussed above. Per violated constraint a penalty of 1000 is added to f. Penalizing is the easiest and most obvious way of dealing with constraints in GAs as candidates with (in this GA) higher fitness are less likely to reproduce. Also so-called death penalty has been implemented where an invalid (at least one constraint is violated) candidate is assigned a fitness of ∞. As candidates are very likely to violate constraints after mutation (discussed below) this would lead to very low variation in the population and all (possibly useful) information in invalid candidates is lost immediately. This is why death penalty was eventually not used in the GA.

5.2.3 Selection

The simplest fitness-proportionate selection strategy for GAs (roulette wheel selection) described in Section 5.2 is used, also because there is a ready made implementation in the Watchmaker Framework in class `RouletteWheelSelection`. Of course more sophisticated selection strategies like `SigmaScaling` or `StochasticUniversalSampling` exist, but the selection operation was not a primary object of research in this work.

5.2.4 Crossover

Standard n-point crossover has been implemented for `GPart<GFret>` objects in class `GFret-Crossover`. Chromosomes cannot be split at any point, because notes in the arrangement can overlap. Therefore first a set of possible split points has to be found in both parents and combined to include common ones to both. The final crossover points are chosen randomly among them. The same segmentation algorithm as used in the fitness function can also be used for this purpose, because boundaries between MEL, CHO and MIX blocks as well as all note boundaries within MEL blocks fulfill the criterion that no note overlaps them.

In rare cases, where there might not be enough common split points to perform the crossover operation, the n parameter is gradually reduced until enough split points can be found. If none are found (which should never be the case, except for extremely short or complicated parts) the parents are returned without change.

5.2.5 Mutation

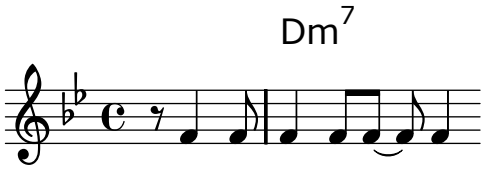
Any fingered position in a chromosome can be changed by the mutation operation implemented in class `GFretMutation` with probability $p_M = 0.1$. If chosen, all possible fingering tuples are computed for the underlying note and the mutated fingered position is substituted for a

random one among them. Also here it has to be taken care of preservation of tied notes. This operation possibly violates constraints, but could also be implemented to include only compatible notes using CSP solution methods.

5.2.6 Termination conditions

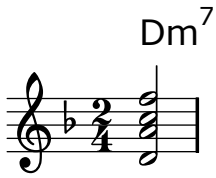
All termination conditions are implementing classes of interface `TerminationCondition` in the Watchmaker Framework. The GA is executed until the fitness value of the best chromosome of each generation stagnates for a certain amount of generations (found in class `Stagnation`). Furthermore the best found candidate must satisfy all constraints.

MEL



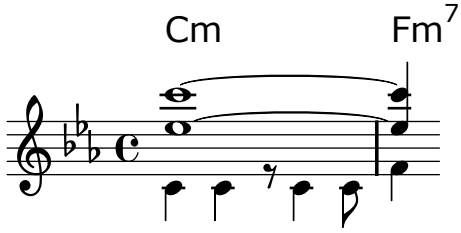
Definition 5. *a finite sequence of n notes form a melody $S = \{s_1, s_2, \ldots, s_n\}$, if for each pair $(s_i, s_j) \in S$ where $j = i + 1$, $offset_{s_i} \leq onset_{s_j}$.*

CHO

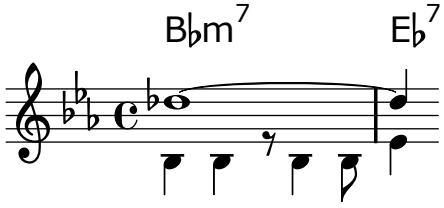


Definition 6. *a set of $k \in \{2, 3, \ldots, 6\}$ simultaneous notes form a chord $C = \{c_1, \ldots, c_k\}$, if for each and every pair $(c_i, c_j) \in C$ $onset_{c_i} = onset_{c_j} \wedge offset_{c_i} \leq offset_{c_j}$.*

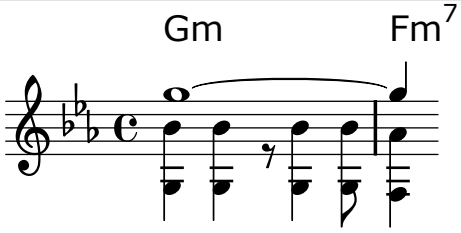
MIX



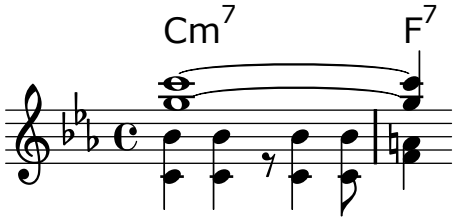
Definition 7. MIX 1*: a finite set of notes form a mixed passage if given a CHO set C of k notes, a MEL sequence S of n notes exists such that for all n, $onset_c \leq onset_{s_n} \wedge offset_c \geq offset_{s_n}$.*



Definition 8. MIX 2*: a finite set of notes form a mixed passage if given an individual MEL note p, a MEL sequence S of n notes exists such that for all n, $onset_p \leq onset_{s_n} \wedge offset_p \geq offset_{s_n}$.*



Definition 9. MIX 3*: a finite set of notes form a mixed passage if given an individual MEL note p, a sequence of q CHOs exists such that for all q, $onset_p \leq onset_{c_q} \wedge offset_p \geq offset_{c_q}$.*



Definition 10. MIX 4*: a finite set of notes form a mixed passage if given a CHO set C of k notes, a sequence of q CHOs exists such that for all q, $onset_c \leq onset_{c_q} \wedge offset_c \geq offset_{c_q}$.*

Figure 5.3: Definition and visualization of different movement types in guitar performance according to [Rad06].

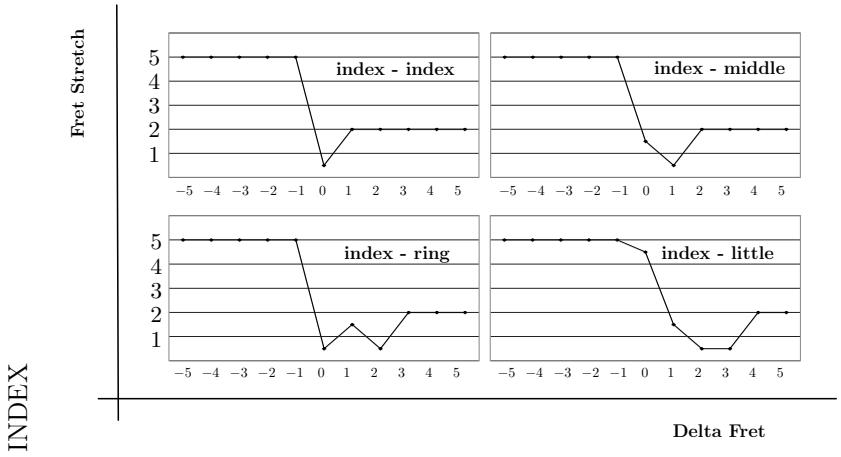
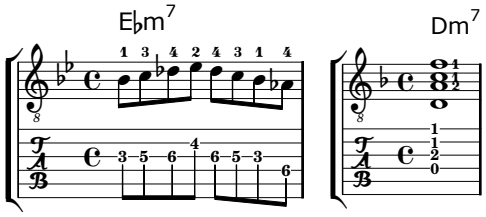


Figure 5.4: $fret\_stretch$ functions for the index finger (from [Rad06]).



(a) MEL block (b) CHO block

Figure 5.5: This figure shows examples of a MEL and a CHO block including fingering that is used to illustrate the functionality of the fitness function. Guitar string and fret information are contained in the lower tablature staff and fingers are indicated in the upper staff.

Part III

Results

CHAPTER 6

Executing the arrangement algorithm

This chapter presents the results obtained by executing the algorithms outlined in Chapter 5. Six test songs from the Bossa Nova genre that are listed in Table 6.1 have been chosen to demonstrate the operating mode of both the segmentation and arrangement algorithms. All test songs belong to the same genre because of the the reason discussed in Section 4.1: The same rhythmic pattern that governs the insertion of chord notes was used in all of them. These songs have been chosen precisely because of the following reasons:

- They are well known.
- The genre uses a variety of interesting chord structures.
- There is a strong predominant underlying rhythmic pattern in the melody.

Table 6.1: Overview of test songs – result reference

Name	Composer	Parts	Segmentation	Arrangement
One Note Samba	Antonio Carlos Jobim	6	Table 6.2	Figure A.2
The Girl from Ipanema	Antonio Carlos Jobim	5	Table 6.3	Figure A.4
Corcovado	Antonio Carlos Jobim	6	Table 6.4	Figure A.6
Desafinado	Antonio Carlos Jobim	7	Table 6.5	Figure A.8
A Felicidade	Antonio Carlos Jobim	6	Table 6.6	Figure A.10
How Insensitive	Antonio Carlos Jobim	3	Table 6.7	Figure A.12

All test songs were obtained from the online lead sheet collection Wikifonia [Fou10] in MusicXML format. Only errors were corrected that would have led to wrong interpretation of lead sheets like when chord symbols were only stored as text annotations and not as a `<harmony>` tag that denotes chord symbols or when chord kinds were wrongly spelled (minor-major → major-minor).

The results of the segmentation algorithm are discussed in Section 6.1 and the ones of the genetic algorithm for arrangement and fingering are discussed in Section 6.2. The complete final arrangements can then be found in Appendix A.

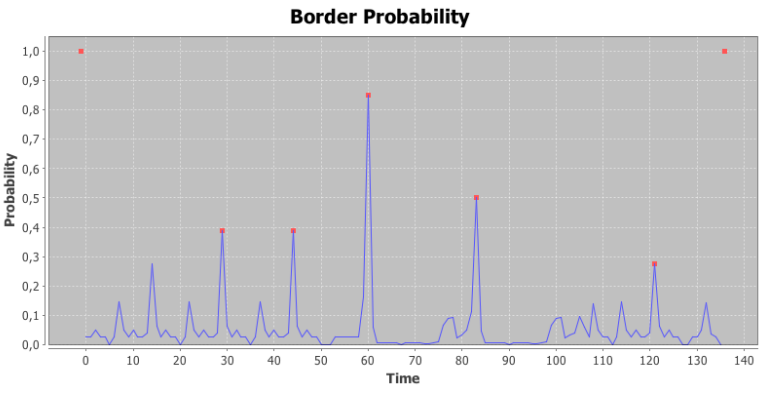
6.1 Segmentation

Tables 6.2 – 6.7 present the segmentation results for all test songs. The table on the left side contains one row per segment giving its starting position, ending position and length. Positions are given as measure index (always starting from 0 – contrary to the convention to denote the first non-upbeat measure with 1) combined with the beat within the corresponding measure. Beats are denoted using fractions, as in the MusicXML format, with 1 being equal to the duration of a quarter note. Therefore a position of $(0, \frac{7}{2})$ indicates the 2nd eighth note in the 3rd beat in the first measure of the piece (see also Figure 4.3). The segment length is given as the number of notes in the segment, where n tied notes are counted as 1 (as it would correspond to the onset-offset representation).

The plot on the right side shows the boundary function (blue line) denoted as b in Algorithm 3. Its maxima indicate suitable segment borders. The horizontal axis shows the indexes of boundaries between two successive notes. That is why the first boundary between the first and second note in a melody has index 0 and therefore the beginning of the song has index -1. The final segment boundaries are shown as red dots, including beginning and end of the song with probability 1, as they are by definition always part of the final segmentation. This is how the position of the red dots on the horizontal axis relates to the data given in the table: The position can be thought of as the cumulative sum of the 3rd column in the table that indicates segment lengths (minus 1).

Table 6.2: One Note Samba – Segmentation

Start		End		Length
m	b	m	b	
0	$\frac{7}{2}$	8	$\frac{7}{2}$	30
8	$\frac{7}{2}$	12	$\frac{7}{2}$	15
12	$\frac{7}{2}$	17	1	16
17	1	21	1	23
21	1	28	$\frac{7}{2}$	38
28	$\frac{7}{2}$	end		15

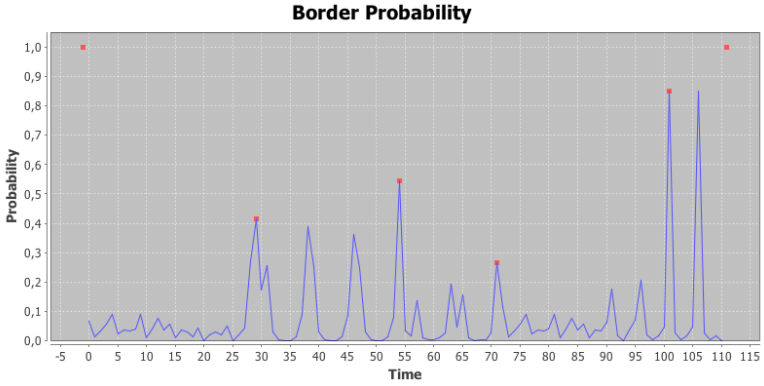


In "One Note Samba" (Table 6.2) the segment boundaries perfectly separate the 4-measure phrases (2 × 4 measures with melody note F, 4 measures with melody note B♭ and again F),

which corresponds to the first 3 found segments. Afterwards the B part (melody starting with chord symbol E♭m^7) is split in 2 pieces because the repeat bar-lines are ignored at the end of it (therefore the peak that starts the last segment is much more pronounced). The last segment is again a 4 measure phrase in B♭.

Table 6.3: The Girl from Ipanema – Segmentation

Start		End		Length
m	b	m	b	
0	1	8	2	30
8	2	21	$\frac{11}{3}$	25
21	$\frac{11}{3}$	26	1	17
26	1	33	$\frac{5}{2}$	30
33	$\frac{5}{2}$	end		10



The first segment of "The Girl from Ipanema" (Table 6.3) correctly comprises the 8 measure theme up to the repeat bar-line. The next 3 × 4 measure phrases are combined into the second segment. A little flaw is that part B is separated (the remaining 4 measures of part B are not included because the separating peak is much stronger due to the long note and rest than the boundary separating parts B and C). The next segment again correctly contains the same theme as the first, then the last 2 measures are repeated two times. Here the second pass merging heuristics eliminated the splitting of the last segment where an equally high peak (0.85) would have created 2 segments with only 5 notes each.

Also in "Corcovado – Quiet Nights Of Quiet Stars" (Table 6.4) all segment boundaries found by the LBDM correctly separate musical phrases. Only the last 2 segments could have been better separated by the peak on index 92, because the last 2 phrases are musically similar and would be better off in one segment together and the note distribution would be more balanced (14+20 instead of 24+10). The other peak is higher nevertheless because the LBDM does not consider musical parallelism.

The first 2 segments of "Desafinado – Slightly out of Tune" (Table 6.5) accurately separate part A until the repeat bar-line. The next boundary is not chosen between parts A and B but after the first phrase in part B because there is a longer rest and therefore a higher peak. The global maximum on index 117 (separation between parts B and C) was overruled by the second pass merging heuristics that preferred 3 segments (27+23+26) over 2 (38+38). The last segment boundary is again perfect.

In "A Felicidade" (Table 6.6) the segments could not have been chosen better, as they make perfect sense musically as well as technically.

The musical segmentation in "How Insensitive – Insensatez" (Table 6.7) is very straight-

Table 6.4: Corcovado – Segmentation

Start		End		Length
m	b	m	b	
0	$\frac{3}{2}$	8	$\frac{3}{2}$	24
8	$\frac{3}{2}$	12	$\frac{3}{2}$	14
12	$\frac{3}{2}$	18	$\frac{3}{2}$	23
18	$\frac{3}{2}$	24	$\frac{3}{2}$	18
24	$\frac{3}{2}$	32	$\frac{5}{2}$	24
32	$\frac{5}{2}$	end		10

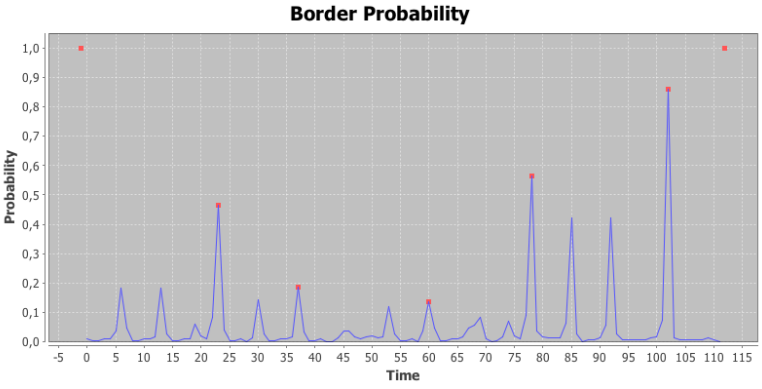


Table 6.5: Desafinado – Segmentation

Start		End		Length
m	b	m	b	
0	$\frac{3}{2}$	8	$\frac{3}{2}$	22
8	$\frac{3}{2}$	16	$\frac{3}{2}$	25
16	$\frac{3}{2}$	28	$\frac{3}{2}$	32
28	$\frac{3}{2}$	35	$\frac{7}{2}$	27
35	$\frac{7}{2}$	44	$\frac{3}{2}$	23
44	$\frac{3}{2}$	51	4	26
51	4	end		33

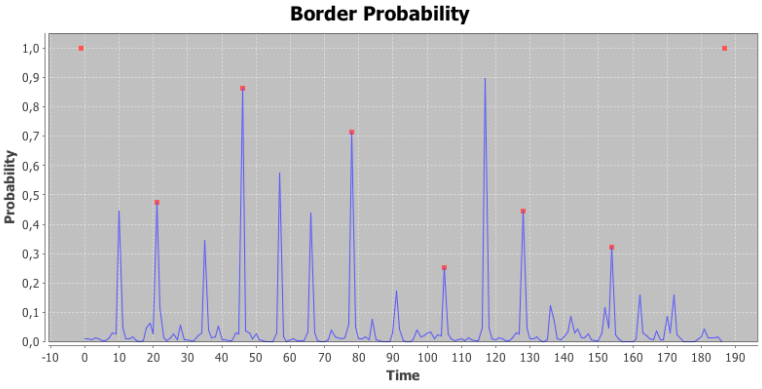


Table 6.6: A Felicidade – Segmentation

Start		End		Length
m	b	m	b	
0	1	17	2	32
17	2	25	1	22
25	1	32	4	21
32	4	44	4	39
44	4	48	4	11
48	4	end		28

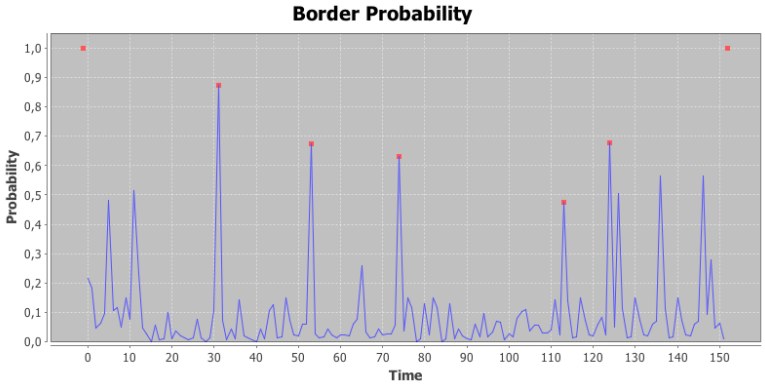
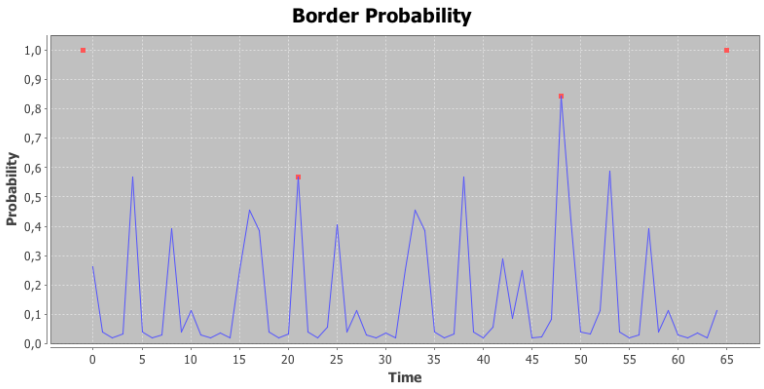


Table 6.7: How Insensitive – Segmentation

Start		End		Length
m	b	m	b	
0	1	11	$\frac{5}{2}$	22
11	$\frac{5}{2}$	24	1	27
24	1	end		17



forward and would have produced 4 segments (17+17+15+17). However, due to the matching pitches of the last note of each segment and the first of the next the peaks in the boundary functions are not as high as the ones in the middle of those phrases. That is why this 3 segment division (22+27+17), with the last segment corresponding to the last musical phrase, is generated. The peaks on index 4 and 38 have again been ignored due to the second pass merging.

6.2 Genetic algorithm

All arrangements in Appendix A have been produced using the underlying rhythmic pattern depicted in Figure 6.1. This pattern has replaced the 2 measure Bossa Nova clave[1] used in earlier experiments because repeats or pickup measures are not considered by the arrangement algorithm, so occasionally the pattern would flip and misalign the main accent on the first beat in the first measure.

Figure 6.1: (Pseudo) Bossa Nova pattern used for potential chord note positions.

In Figure 6.2 (upper plot) the development of the fitness score in the first segment of Corcovado is shown. The fitness score of the best candidate in the population is drawn in blue and improves from 38.79 to 34.33. At the same time the mean fitness (green) in the population almost resembles gaussian noise ($\mu = 252.99$, $\sigma = 6.69$), but in generation 0, where all candidates in the population are valid, the mean fitness is significantly lower (73.29), which can be seen in the lower plot of Figure 6.2, which shows the first 700 generations of the same data. It immediately increases to the same level that is maintained throughout the optimization process. This is due to the high penalty in the fitness score and high probability of constraint violation in the mutation of the candidates, such that only few fully valid candidates remain in the population.

The following parameter values have been used in the genetic algorithm to produce the results in Appendix A. A generational approach was chosen (as opposed to an island model) with $n_P = 2000$ chromosomes in the population. An elite percentage (the percentage of the population with the best fitness score that are preserved unchanged in the next generation) of $p_E = 20\%$ and 2-point crossover was applied. The evolution process was stopped if no better chromosome was found for $n_G = 700$ generations.

The weights of the fitness function were set as $\alpha_1 = 1$ and $\alpha_2 = 30$. Those values have been found to be a good compromise between playability and the number of notes in the arrangement. If α_1 was too dominant the resulting arrangement would be very thin and vice versa if α_2 was too high the arrangement would be very difficult to play.

"One Note Samba" is a predestinated test song as its melody consists of the same note throughout long passages such that the contribution of chord notes can be seen and heard easily. The first segment was unfortunately not well optimized in terms of guitar fingering although the

[1] `http://en.wikipedia.org/wiki/Bossa_nova`

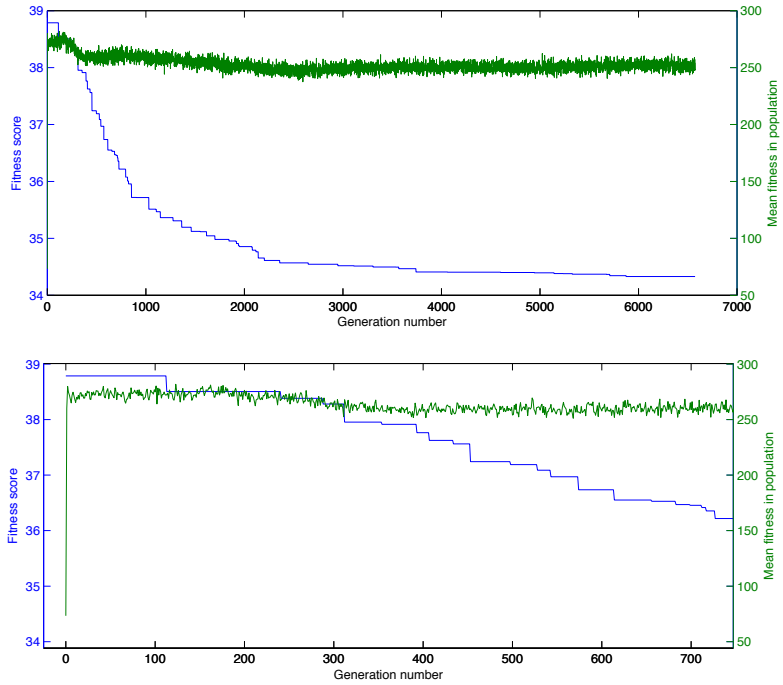


Figure 6.2: Demonstration of the optimization process of the first segment in the song "Corcovado". The blue line in both plots shows the fitness score of the best candidate in the population over generations, the green line indicates the mean fitness in the population. The plot below zooms in the same data of the first 700 generations, such that the increase in the mean fitness after the first generation can be seen distinctly.

middle parts could very well be played right away. Big differences can be seen in the fingering of CHO or MIX passages and neighboring MEL sections as melody positions are often very messy ("One Note Samba", measures 0 and 1) and the transitions between them are sometimes abrupt (measure 3), which means that the performer's hand would have to move quickly along the fretboard. Very often ("One Note Samba", segment 1, second ending; "The Girl from Ipanema", beginning, part B; etc.) the problem is that fingerings are not consistent locally and especially fingering in MEL blocks could be optimized easily.

A second shortcoming is the lack of consideration of harmonic information in the fitness score. This is reflected in the chord voicings that seem random (which they are!) and most significantly in the bass line (D♯ or A on $B^{7\flat5}$ in measures 8 and 15 of "One Note Samba"; A on $Fmaj^7$ in measure 8-9 of "Girl from Ipanema"; etc.). "A Felicidade" lacks bass notes in the first part altogether. On the other hand the last part of "Corcovado" among others is very well done in terms of bass notes.

In "Girl from Ipanema" the approach of inserting chord notes at fixed positions fails in some parts due to the triplets in the melody (measures 22, 24, 31, 33, 35) which creates a very interesting polyrhythm. Further examples can be found in "A Felicidade".

When inserted chords are not bound by the melody note this can result in musical "outliers" as can be seen in the first chord on "Corcovado", further examples are in measures 15 and 17 of "How Insensitive". But those chords can also fill empty measures with an accompaniment pattern (measure 16 in "One Note Samba") or by chance create nice final chords (as in "Corcovado", last measure).

Due to constraints in the MusicXML format the chord symbol changes in the last line of "A Felicidade" where one melody note spans over 4 measures and two chord symbols are given at each one are not correctly reflected in the arrangement since chord positions are represented implicitly (the sequence of `<note>` and `<harmony>` tags determines the position). Therefore if two chord symbols have to be written over a whole note, both chord symbols will be inserted successively at the same position and the second chord symbol will be moved graphically (`relative-x` attribute) to the right. That is why only the first chord symbol ended up in the arrangement.

The main problem of the arrangement algorithm as presented here is the absence of musical criteria. Most significantly when listening to the result arrangements the bass or root note of chords is often missing in critical positions (first beat of every measure), which disrupts the listening experience. One method to overcome this would be to add penalties for missing bass or root note in the fitness function. Furthermore the chord note selection could also be evaluated therein as not all chord notes are created equal (thirds and sevenths are more salient features than fifths).

The segmentation could also be optimized: considering bar-lines, repeats and chord progressions (cadences) would improve finding significant musical boundaries in the music. Furthermore when repeats are unfolded the usage of two-measure patterns for chord note insertion would be possible without the danger of misalignment.

Last but not least, the GA's handling of constraints could also be improved by considering them also in mutation. For example when a chord note is selected for mutation the emerging CSP could be solved such that only valid candidates remain in the population. This would certainly make the optimization more targeted.

CHAPTER 7

Conclusion and Outlook

In this work the subject of automatic arrangement of lead sheets for solo-guitar was examined and a prototype system proposed that constitutes a first attempt towards solving this multifaceted problem. As mentioned before automatic arrangement is far from being exhausted in scientific discussion and as a result offers much space for new ideas and algorithms.

Here the arrangement together with the guitar fingering problem was solved using a genetic algorithm, where chord notes were inserted on given metrical positions and arrangements (already in fingered-positions-domain) were optimized mainly towards playability.

All in all the arrangements produced by this algorithm are a first step in exploration of the subject. They are not as advanced that a guitar performer could immediately pick up a guitar and play them right away, but they are also not completely impossible or musically far out. Some passages actually contain good arrangement ideas, that can be further used. The possibility of creating arrangements based on melody and harmony information with a genetic algorithm has been studied in this work among others. Nevertheless far more work can and should be invested to further understand the subject.

7.1 Future work

Studying the results it should be questioned whether genetic algorithms are the most suitable approach in automatic arrangement due to the highly constrained nature of the problem. For the arrangement part itself there may be found a fitness function that satisfactorily implements certain rules or models how arrangement is supposed to work. But the fingering problem is probably better solved in a post processing step using already well established and efficient algorithms like the one presented in [Rad06].

A next step in working on the problem would be to study arrangement theory from books or query a team of professional arrangers on how experts approach the problem. Also in other fields expert systems that were derived from expert knowledge have achieved great success. Maybe a more constructional or rule-based algorithm would be worth developing. Together with several experts a data set could also be collected or created that can be used for learning or evaluation.

How generated arrangements are compared against some gold standard (and what that might be) in a systematic way is anyway a completely unsolved problem.

More work needs to be done in feeding arrangement algorithms with harmonic knowledge. Some ideas from [PEB05, PEBB05, PEB06, Pai08] in the areas of modeling chord progressions and harmonization as well as [Pac99, Pac00, PR01, Che04] in harmonization and analysis/synthesis of jazz chord sequences might help here. Such models can be used to introduce transition chords or interludes in arrangements. One issue in arrangement is also the concept of chord voicing, that has been considered in [EMY08] for arranging piano music. Furthermore more advanced methods to represent and deal with rhythm are needed. Rhythm representation, analysis and modeling are the main subject-matter in [PGBE08, TT08b, TT08a, Tou02]; guitar accompaniment with a focus on Bossa Nova can be found in [DST$^+$03, DST$^+$04]. It is evident that further research would greatly benefit from results obtained in the fields of automatic accompaniment, improvisation, composition or harmonization.

New tools are also required to carry on with research: the symbolic music framework developed for this work is far from being complete and robust. An open issue is also visualization (notation) of music in MusicXML or other representations. Here a library for transforming music into a visual representation in conjunction with the framework mentioned above would make many things easier.

APPENDIX A

Lead sheets and arrangements

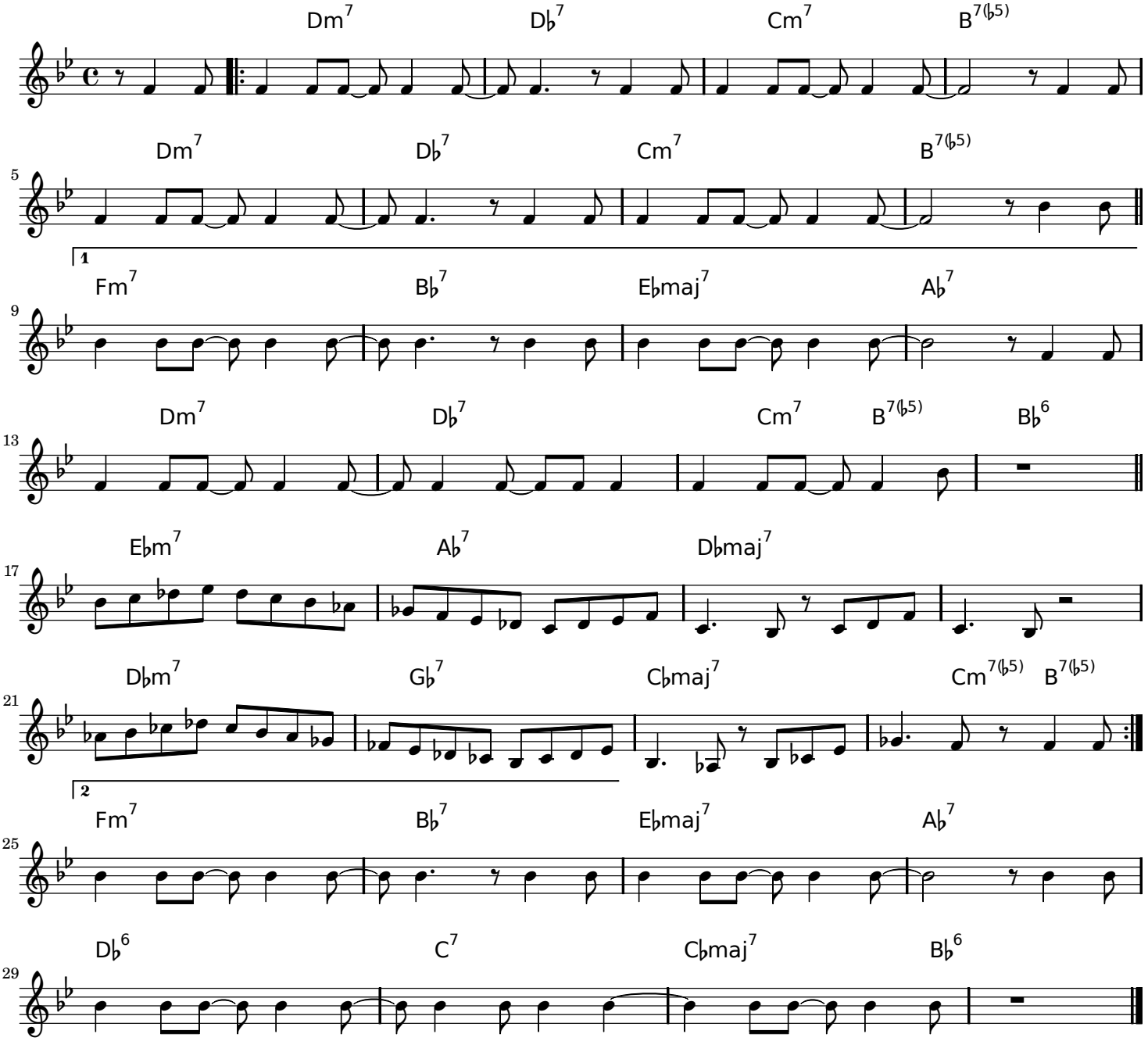
In this appendix all input lead sheets and resulting arrangements discussed in chapter 6 can be found. Table A.1 lists all test songs and gives additional origin information. All songs were composed by Antonio Carlos Jobim.

Table A.1: Overview of test songs – origin information

Name	Published By	Published on	Link
One Note Samba	tom	July 23rd, 2007	http://www.wikifonia.org/node/362
The Girl from Ipanema	benoit	November 5th, 2006	http://www.wikifonia.org/node/129
Corcovado	Musicdad	August 9th, 2010	http://www.wikifonia.org/node/6778
Desafinado	lasconic	December 18th, 2008	http://www.wikifonia.org/node/1025
A Felicidade	TxRx	January 24th, 2009	http://www.wikifonia.org/node/1167
How Insensitive	Mauro58	September 19th, 2009	http://www.wikifonia.org/node/2697

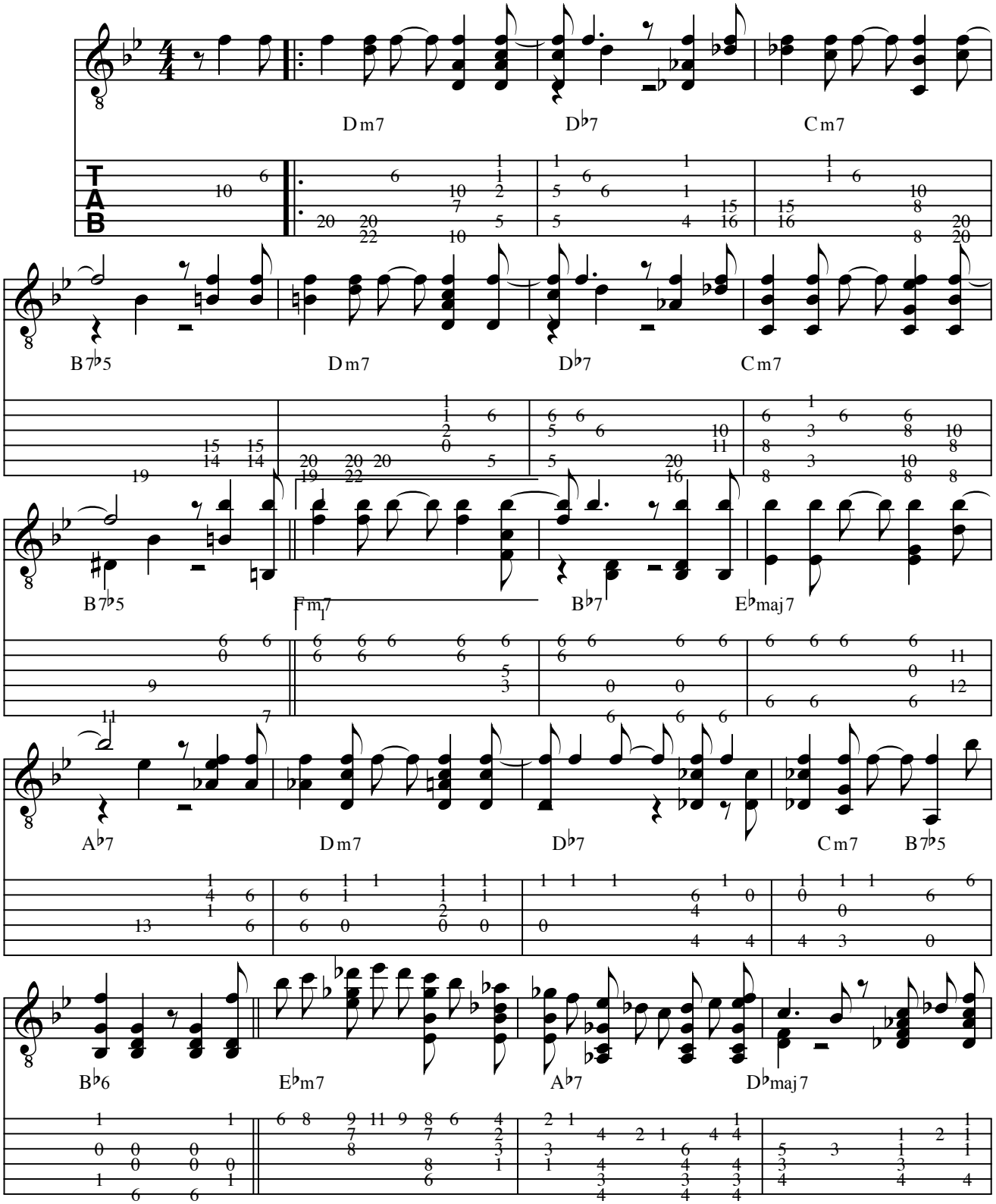
One Note Samba

Music by Antonio Carlos Jobim

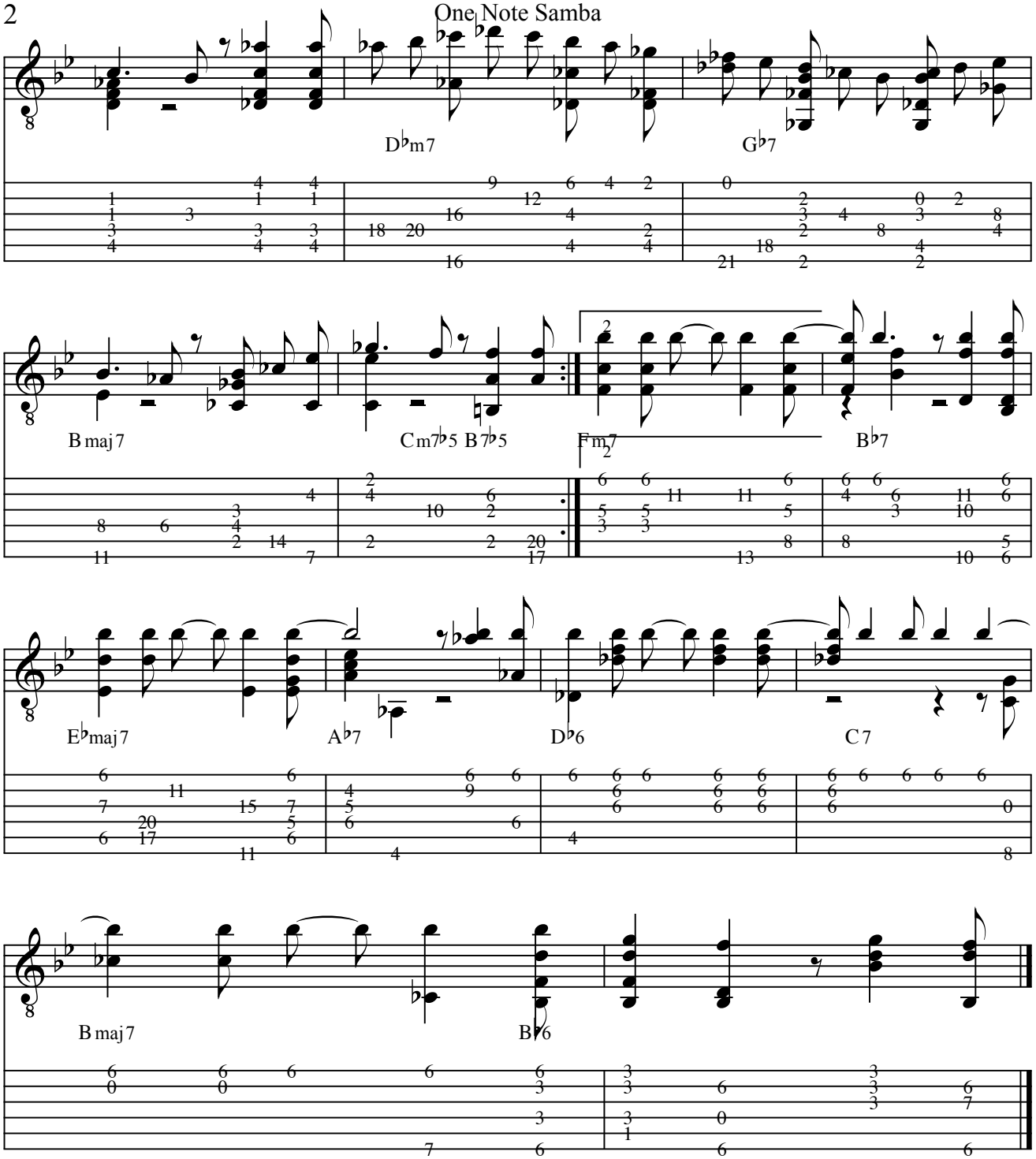


One Note Samba

Antonio Carlos Jobim

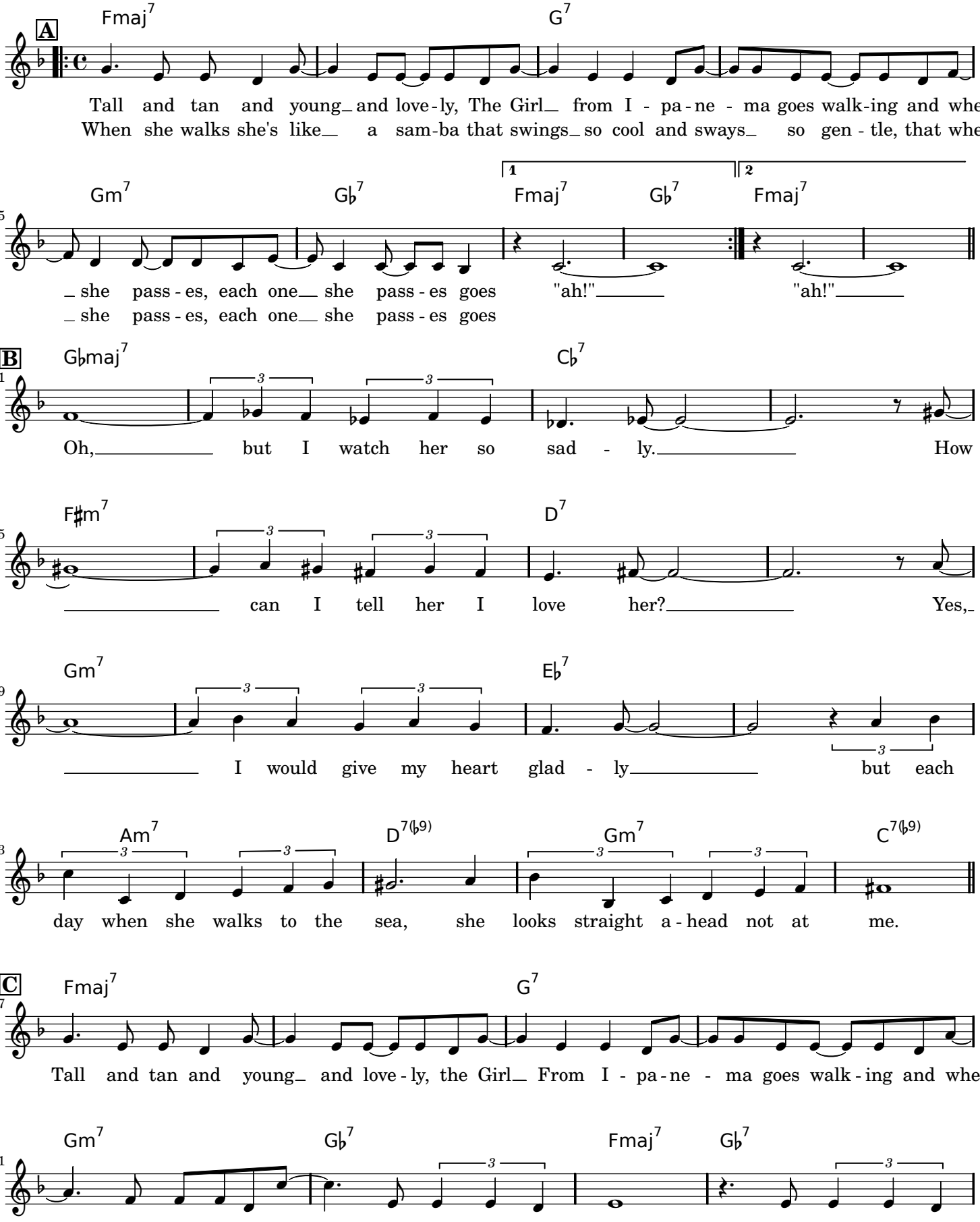


One Note Samba

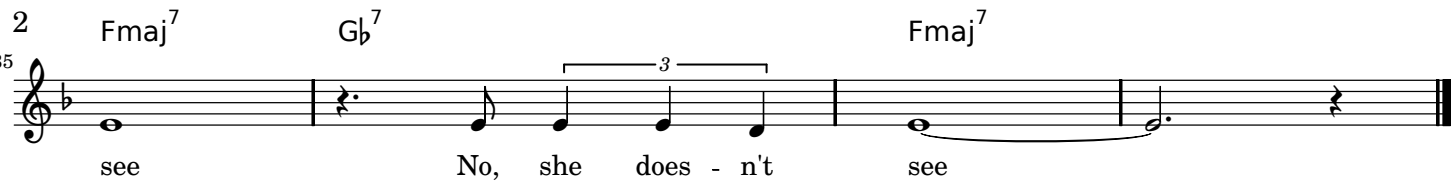


The Girl from Ipanema

Music by Antonio Carlos Jobim

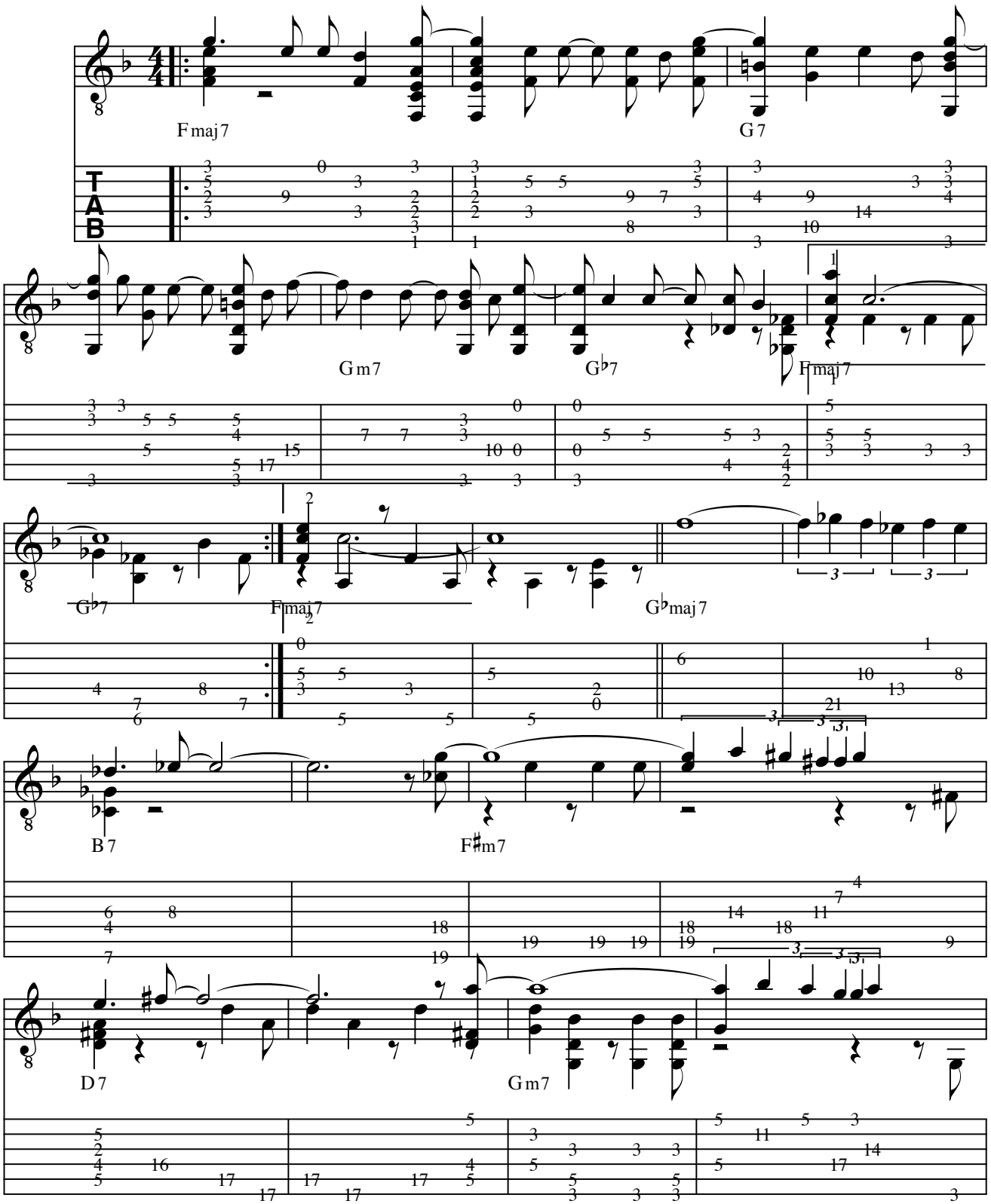


2

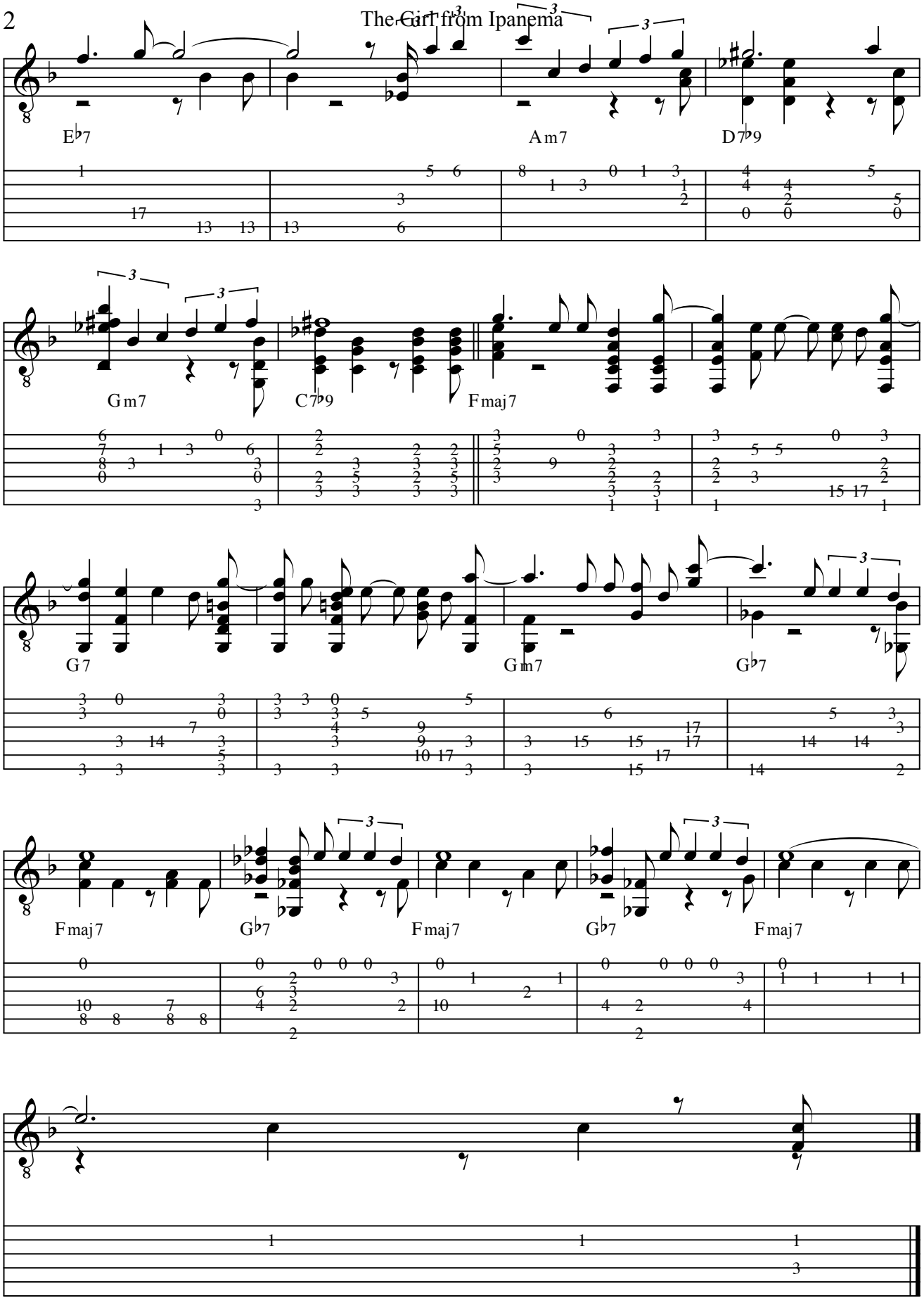


The Girl from Ipanema

Antonio Carlos Jobim



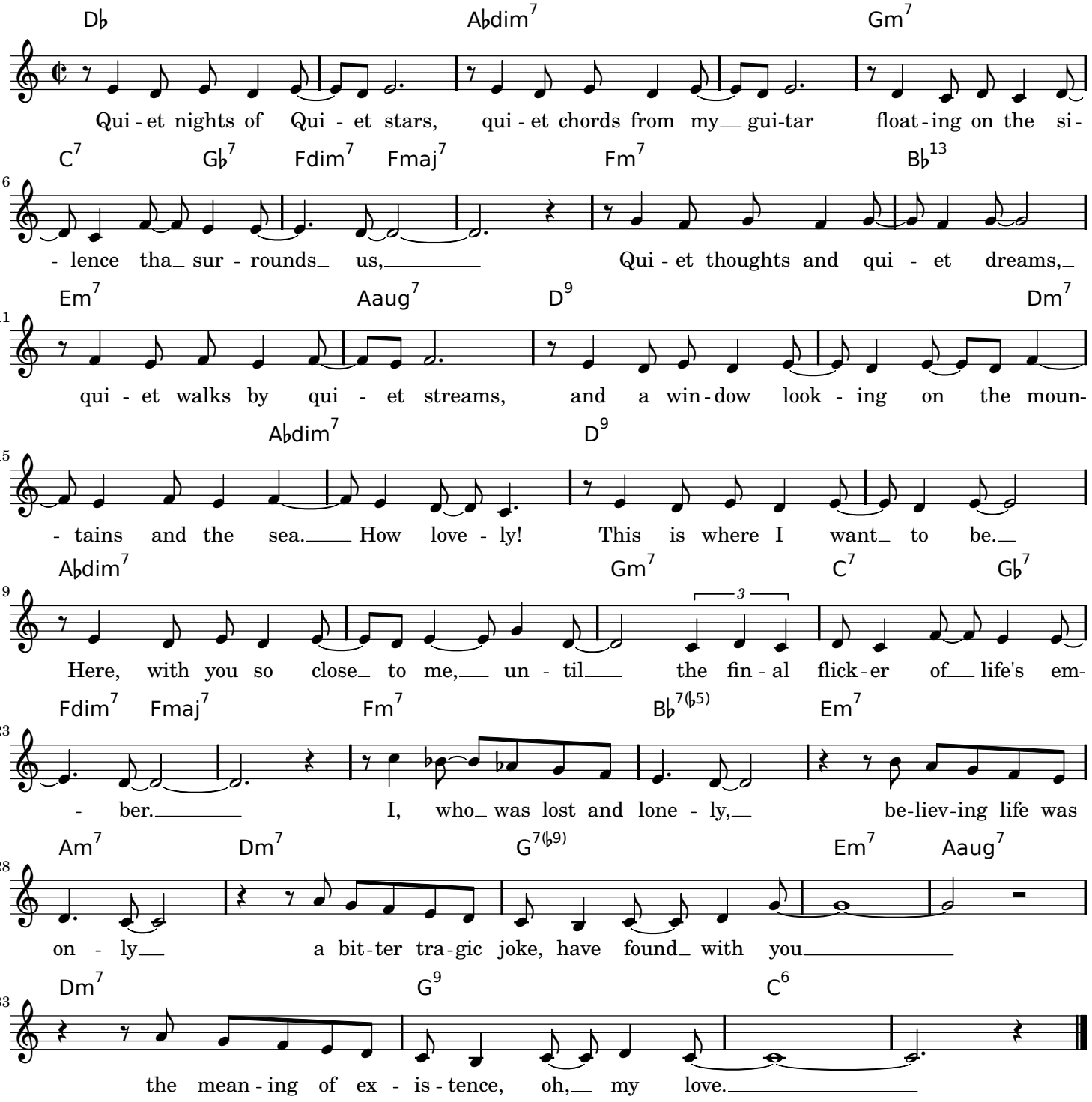
The Girl from Ipanema



Corcovado

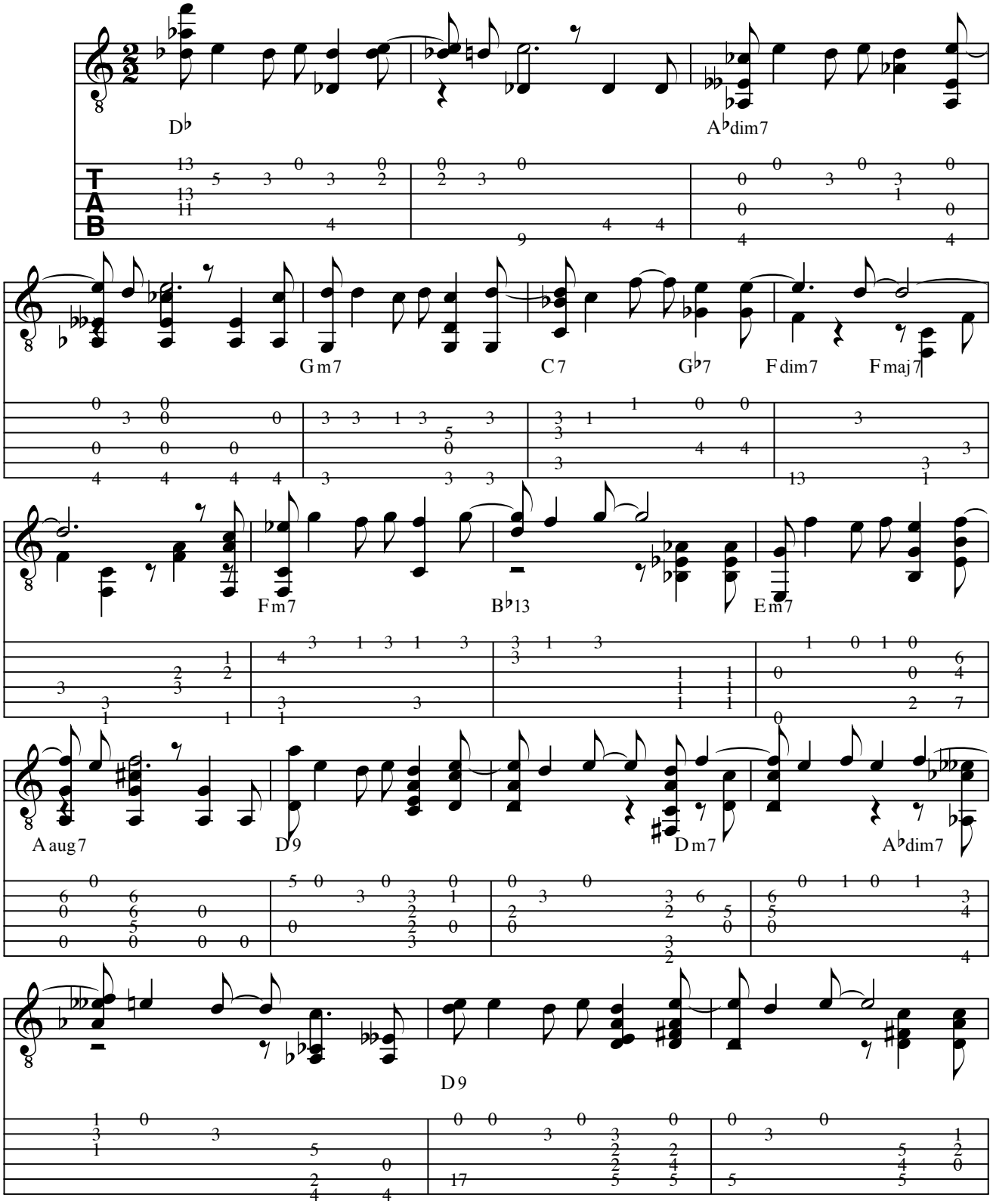
Quiet Nights Of Quiet Stars

Words & Music by Antonio Carlos Jobim, Gene Lees (English)

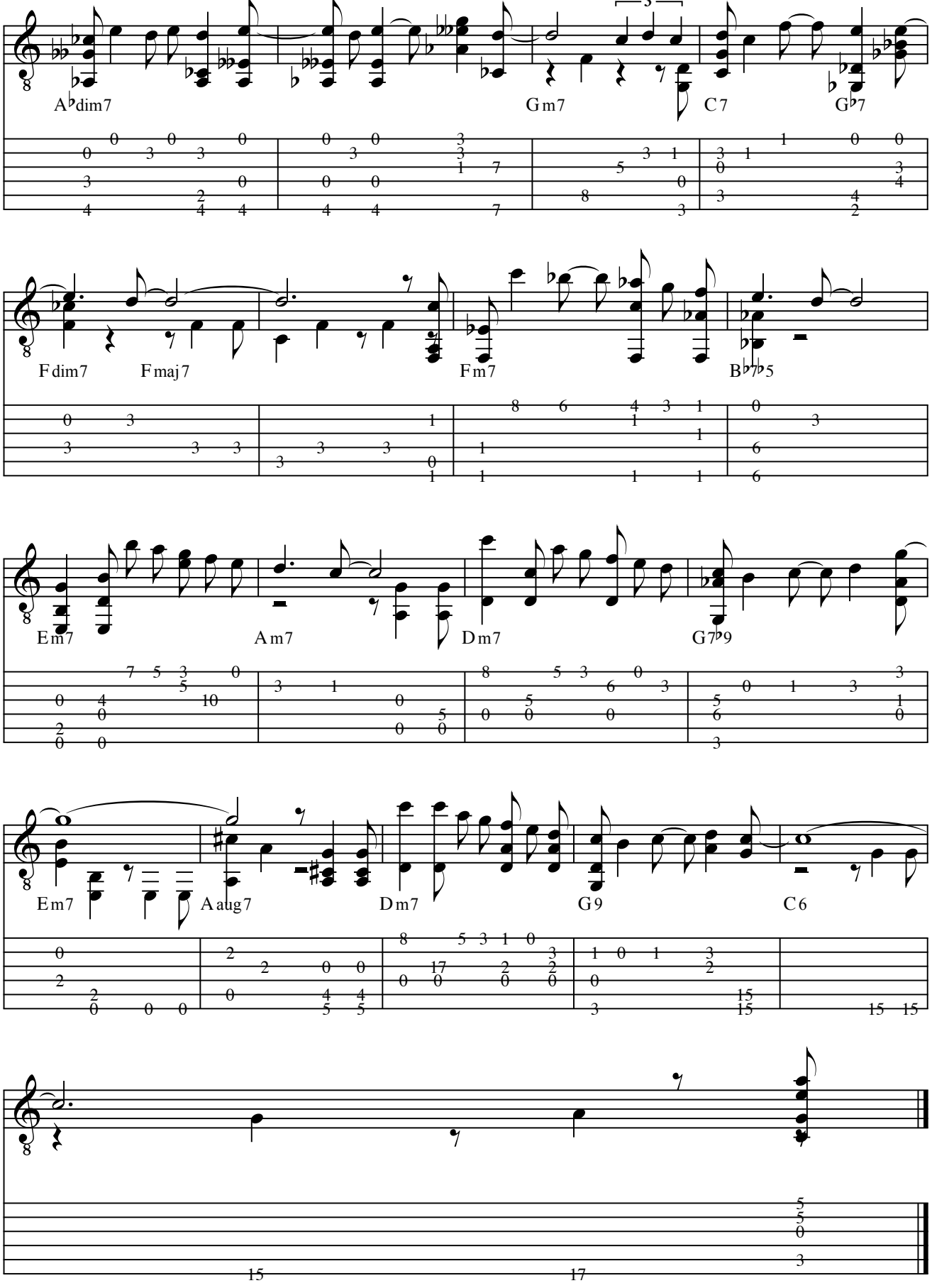


Corcovado

Antonio Carlos Jobim



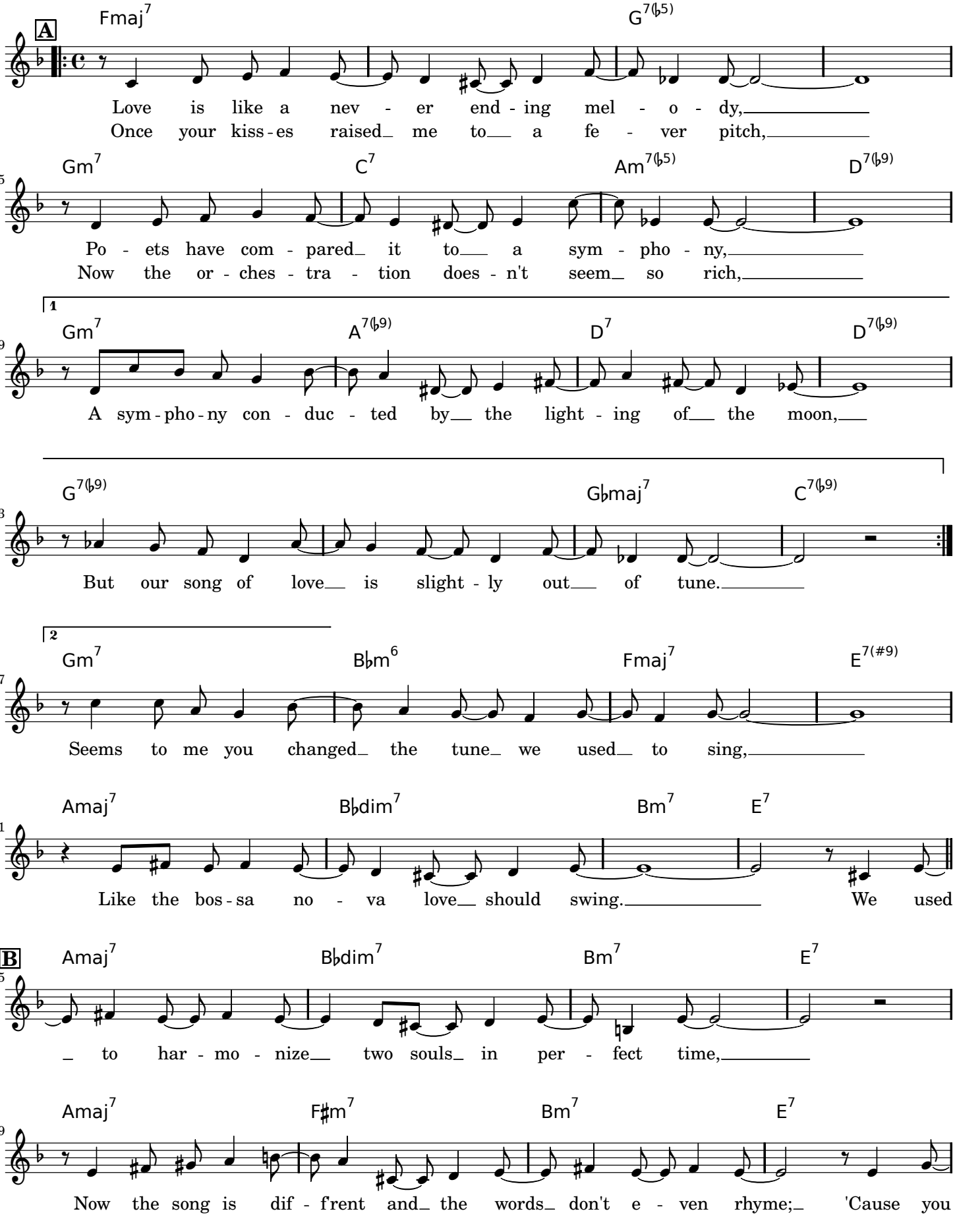
Corcovado

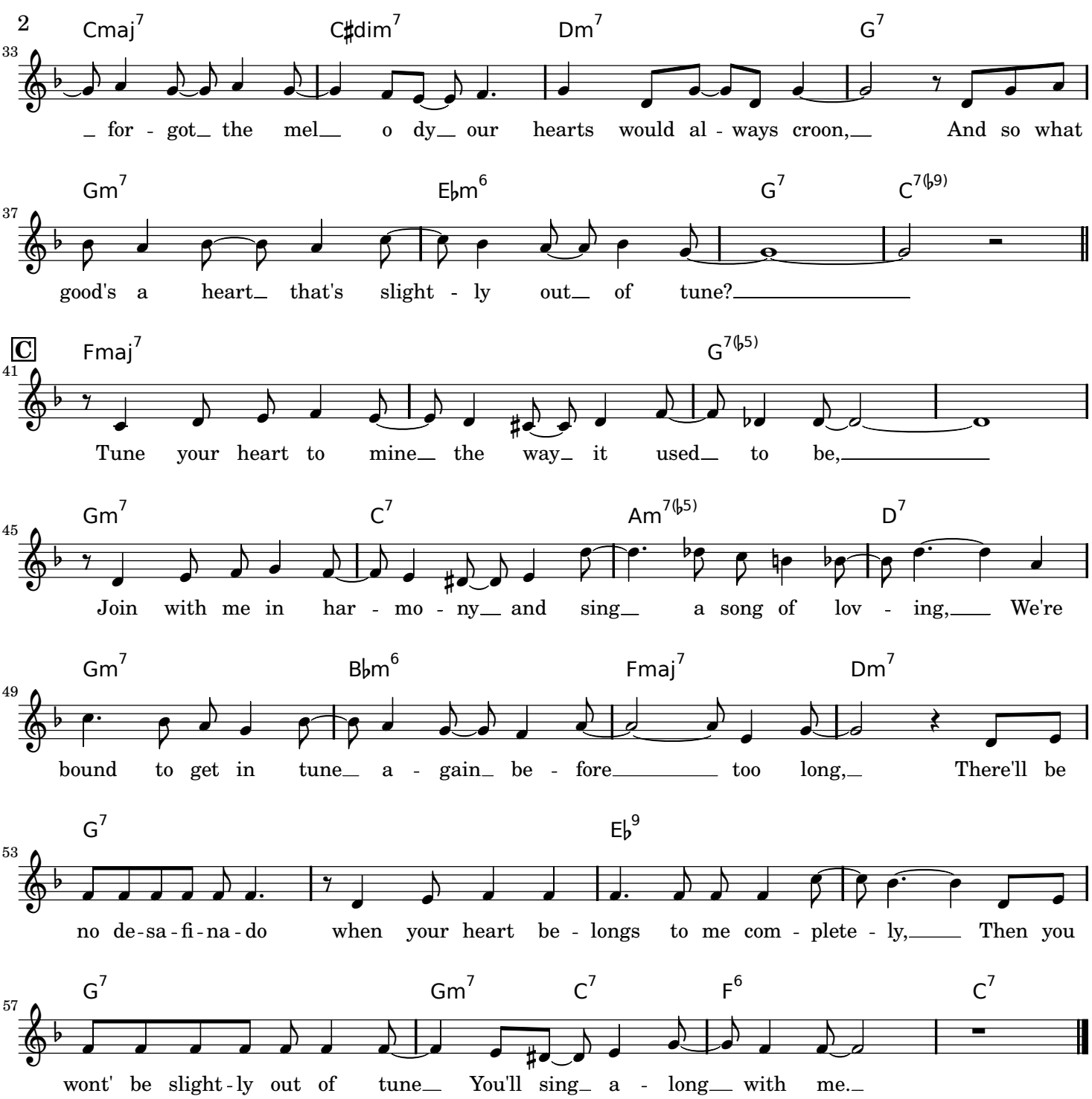

A♭dim7
Gm7
C7
G♭7
Fdim7
Fmaj7
Fm7
B♭♭5
Em7
Am7
Dm7
G7♭9
Em7
Aaug7
Dm7
G9
C6

Desafinado

Slightly out of Tune

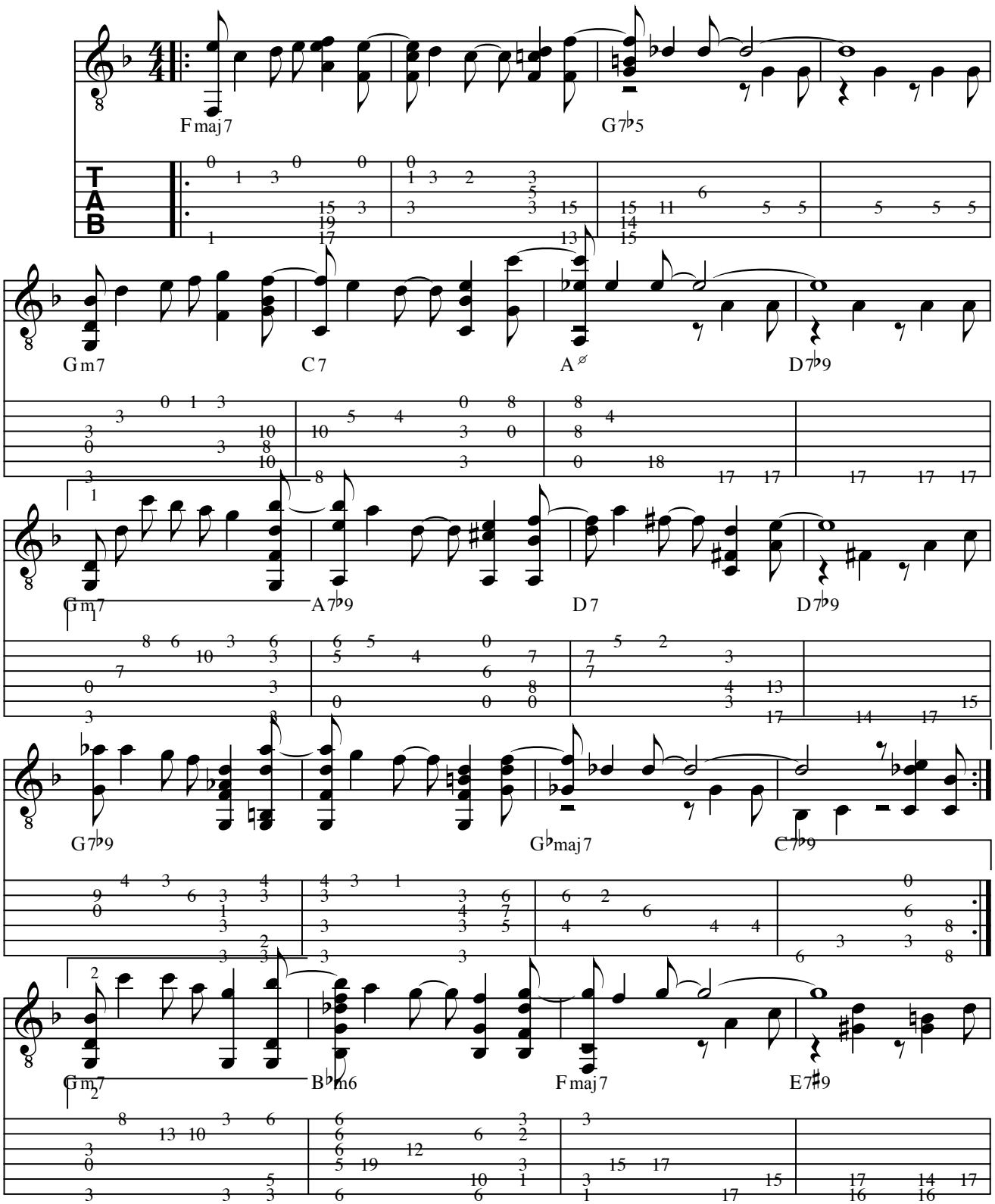
Words & Music by Antonio Carlos Jobim, Jon Hendricks



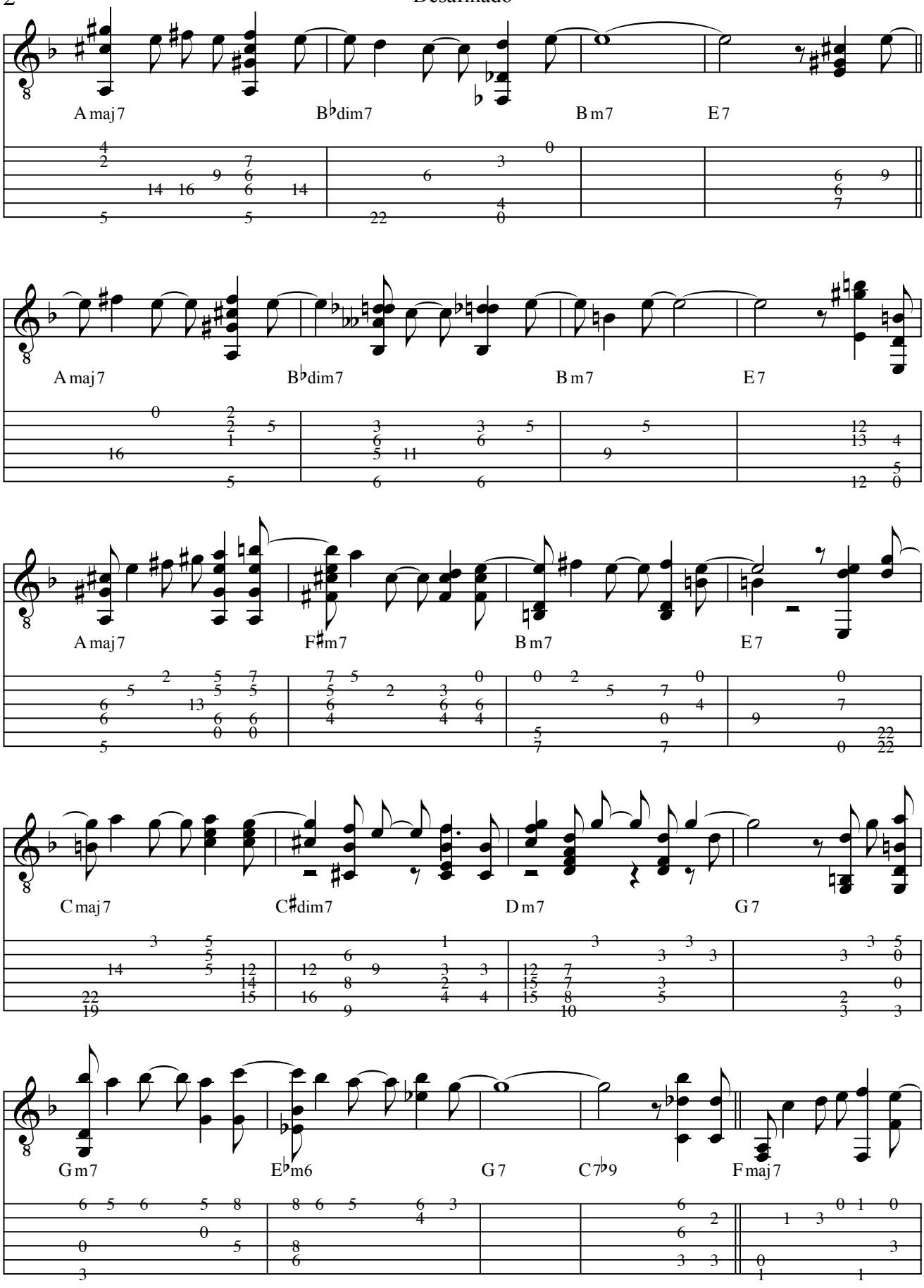


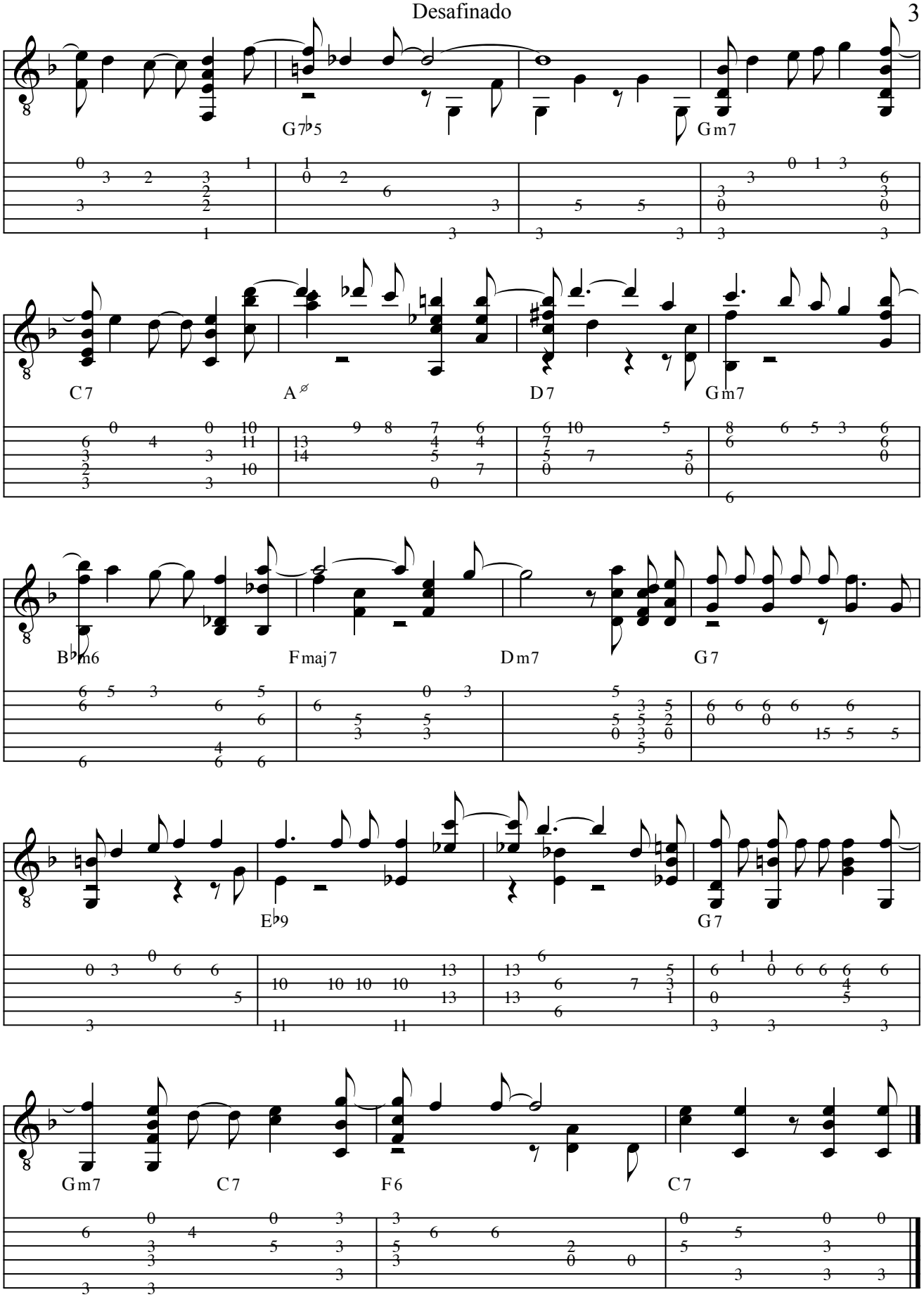
Desafinado

Antonio Carlos Jobim



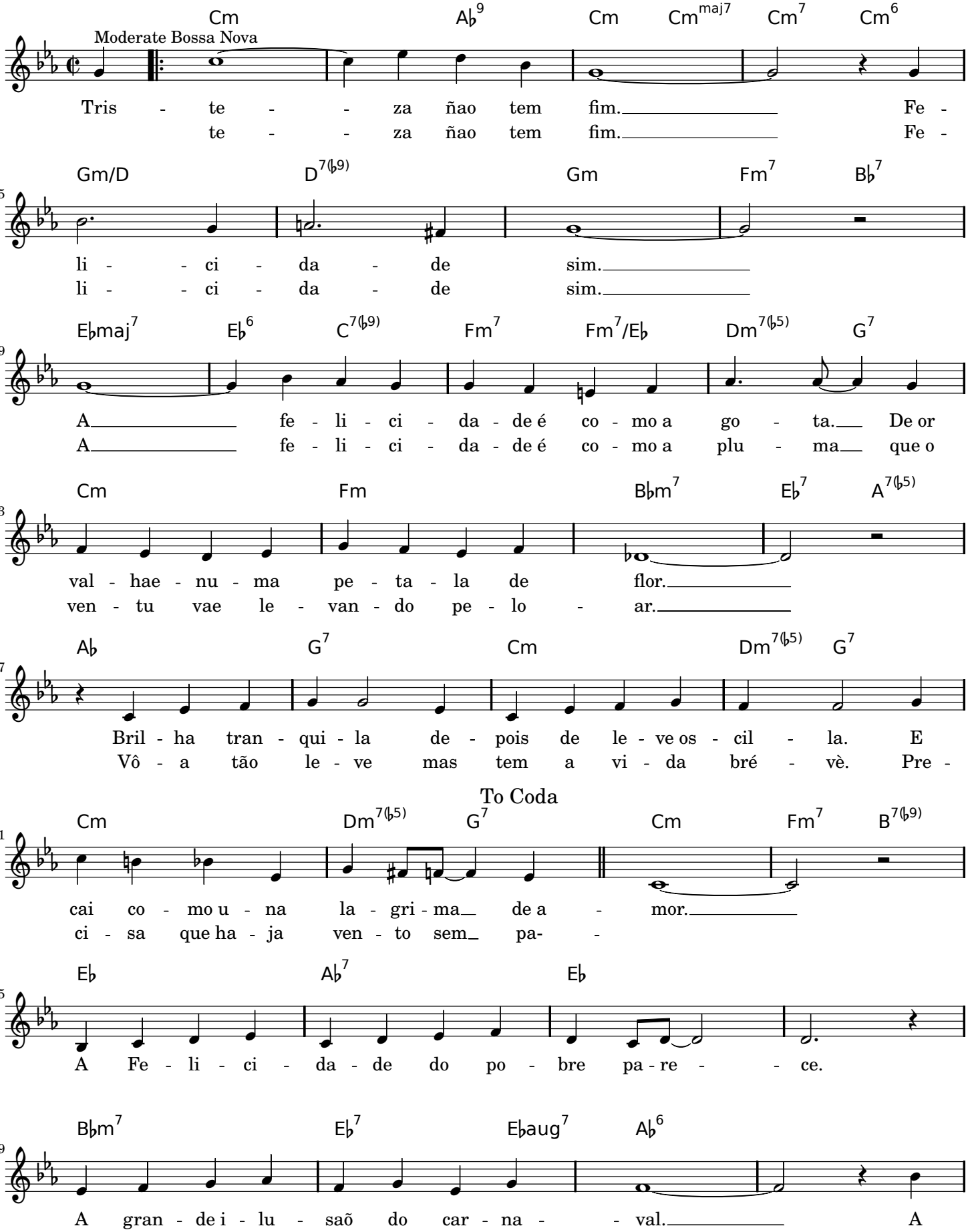
Desafinado


A maj7
B♭dim7
B m7
E 7
A maj7
B♭dim7
B m7
E 7
A maj7
F♯m7
B m7
E 7
C maj7
C♯dim7
D m7
G 7
G m7
E♭m6
G 7
C7♭9
F maj7


G7♭5
Gm7
C7
A∅
D7
Gm7
B♭m6
Fmaj7
Dm7
G7
E♭9
G7
Gm7
C7
F6
C7

A Felicidade

Words & Music by Antonio Carlos Jobim, Vinicius de Moraes



33 Fm7 B♭7 E♭

gen - te tra - bal - ha o a - ho en tei - ro. Por

37 Am7(♭5) Daug7 Am7(♭5) Daug7 Gm7(♭5) C7(♭9) Fm7(♭5) B♭7

un mo - men - to de son - ho. Pra - fa - zer a fan - ta - si - a. De

41 E♭ Dm7(♭5) G7

rei ou de pi - ra - ta ou jar - di - nei - ra. E

45 Cm Dm7(♭5) G7 Cm

tu - do se a - ca - bar na quar - ta fei - ra Tris-

49 Cm Dm7(♭5) G7

rar. Pre - ci - sa que ha - ja ven - to sem pa -

53 Cm Dm7(♭5) G7

rar. Pre - ci - sa que ha - ja ven - to sem pa -

57 Cm A♭9

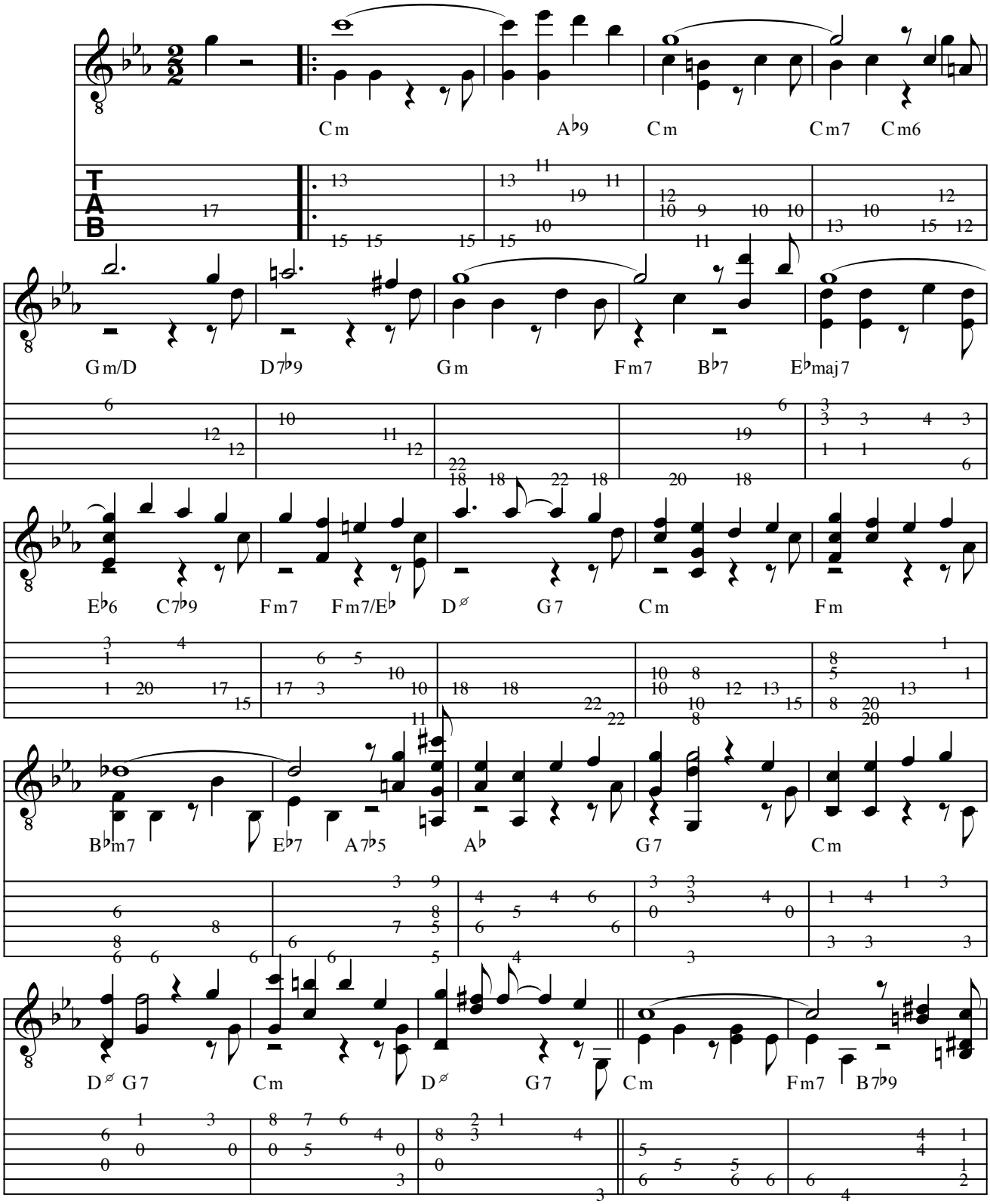
rar. Tris - te - - - za ñao tem

61 Cm Cmmaj7 Cm Cm6 Cm Cmmaj7 Cm7 Cm6 Cm

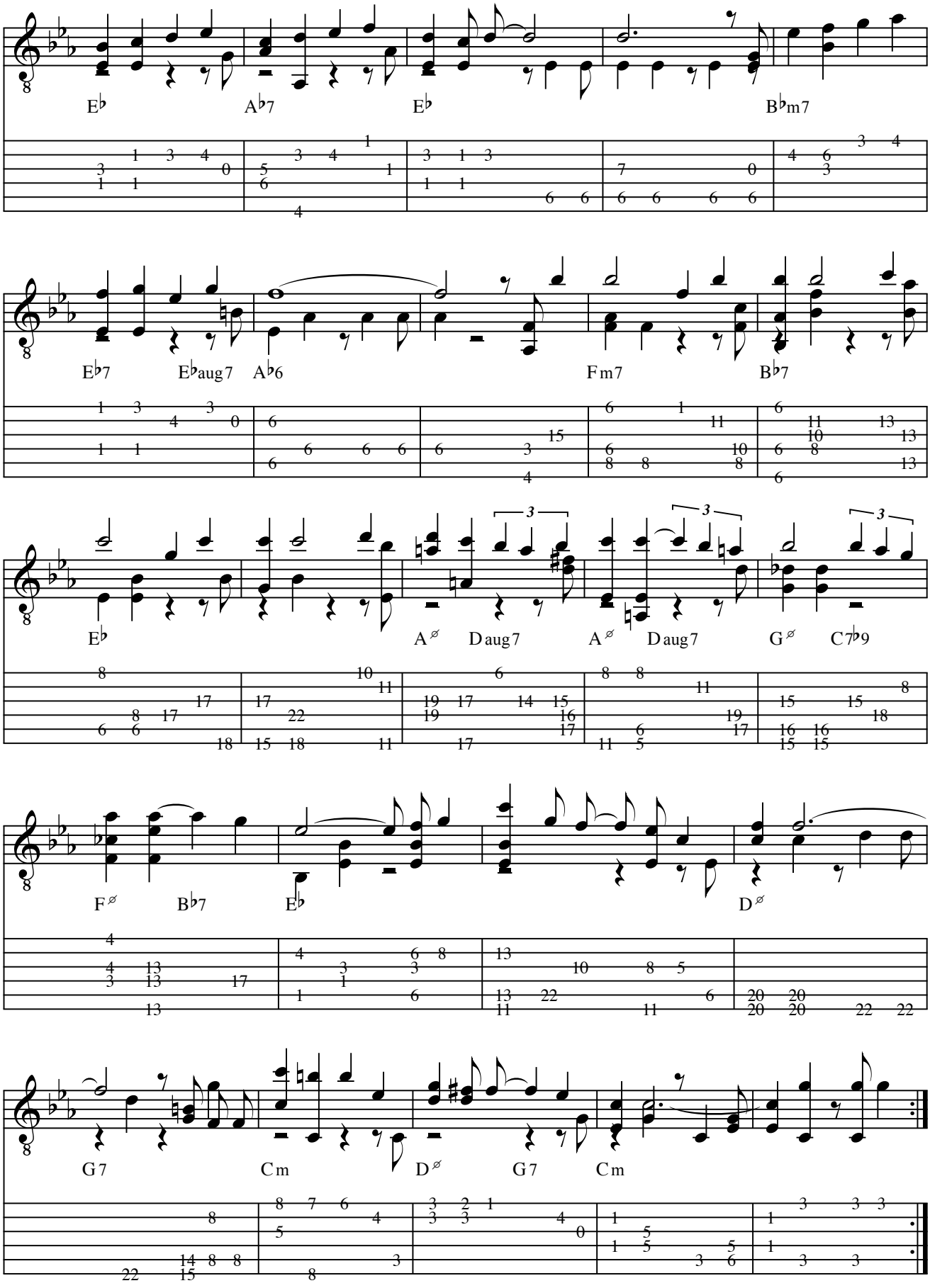
fim.

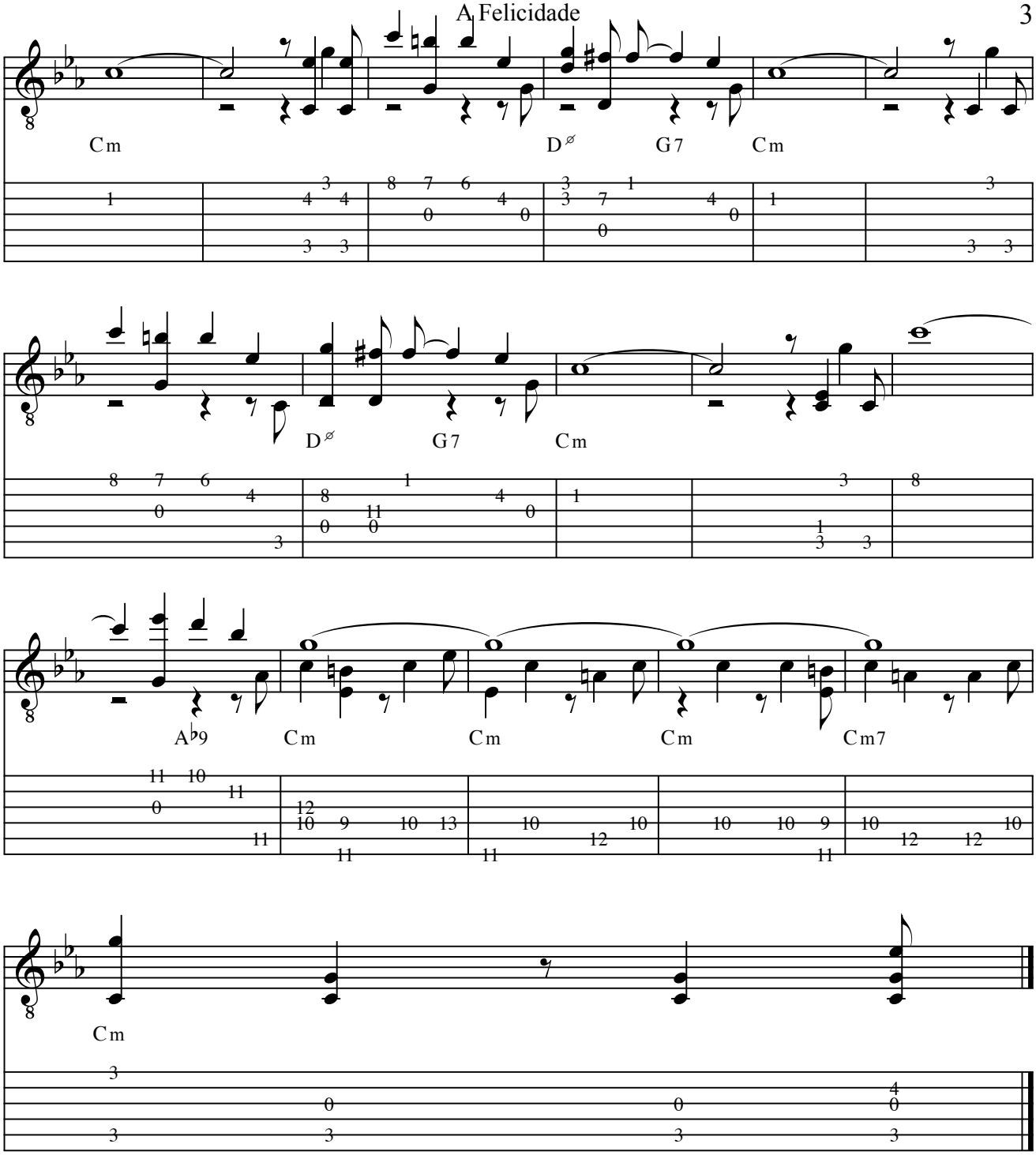
A Felicidade

Antonio Carlos Jobim



A Felicidade

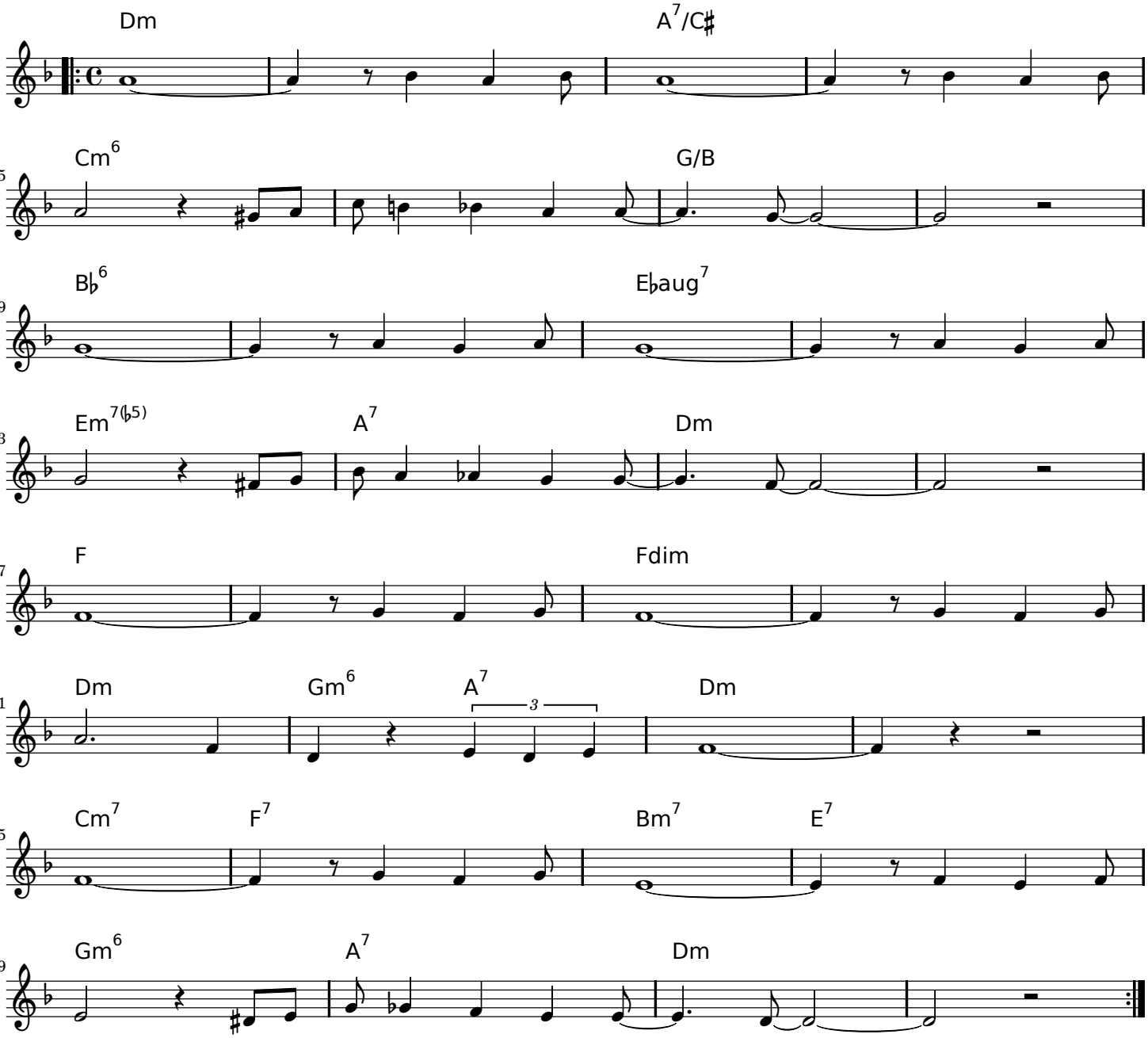



Cm
D∅
G7
Cm
D∅
G7
Cm
A♭9
Cm
Cm
Cm
Cm7
Cm

How Insensitive

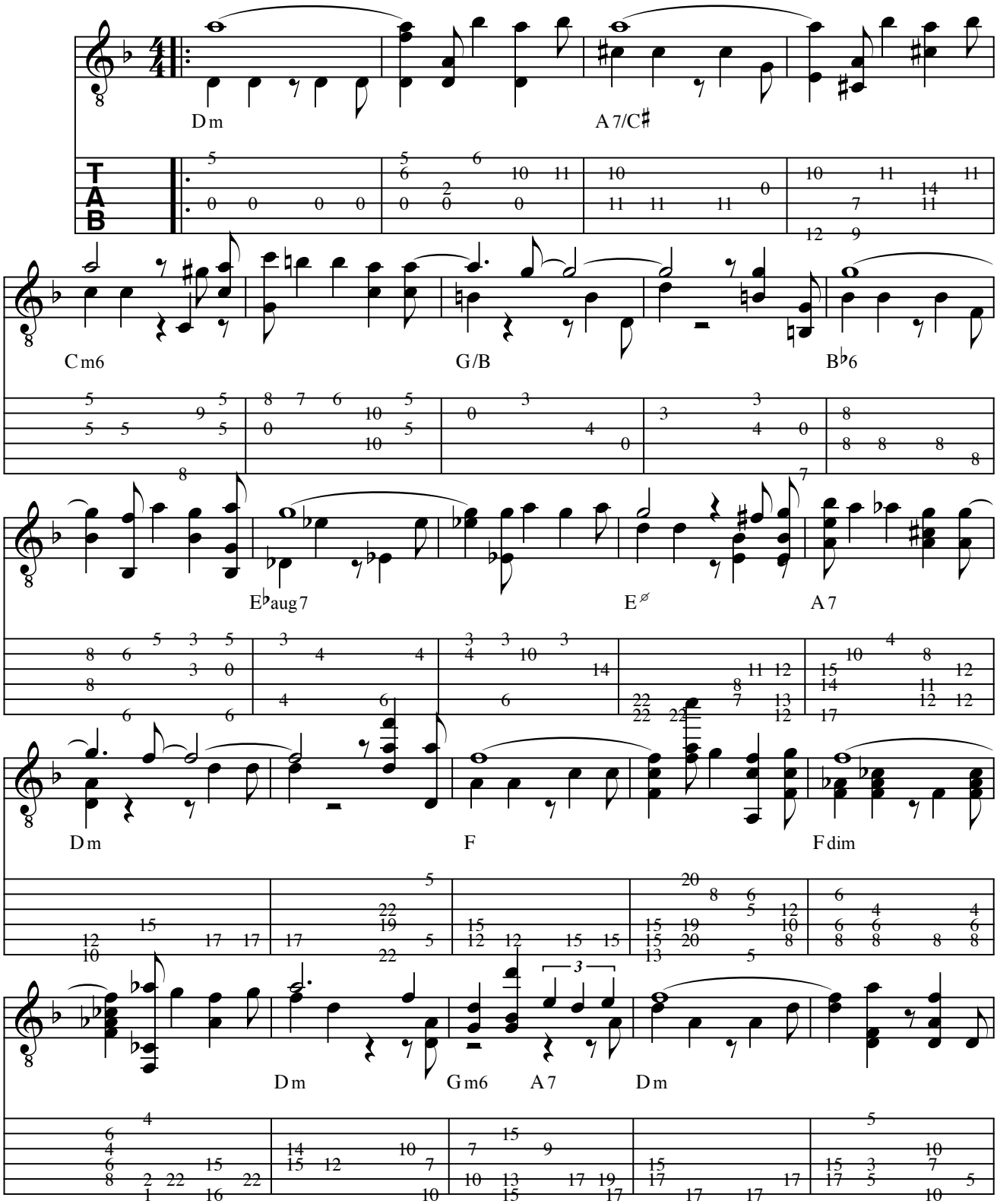
Insensatez

Music by Antonio Carlos Jobim

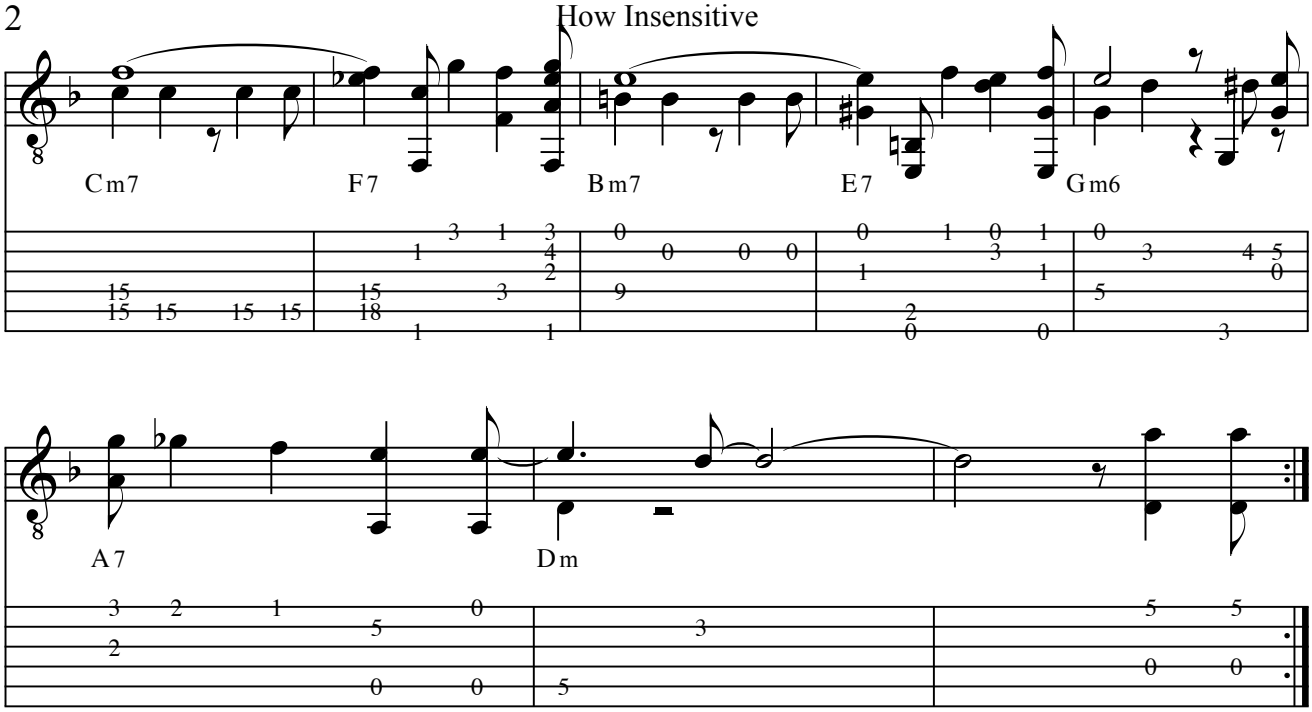


How Insensitive

Antonio Carlos Jobim



How Insensitive



Acronyms and abbreviations

MIDI Musical Instrument Digital Interface

GA Genetic algorithm

HC Hill climber

ANN Artificial neural network

CSP Constraint satisfaction problem

LBDM Local boundary detection model

IOI Inter-onset intervals

OMR Optical music recognition

DFS Depth first search

IOI Inter-onset interval

Bibliography

[AKNR07] Alia Al Kasimi, Eric Nichols, and Christopher Raphael. A simple algorithm for automatic generation of polyphonic piano fingerings. In Simon Dixon, David Bainbridge, and Rainer Typke, editors, *Proceedings of the 8th International Symposium on Music Information Retrieval (ISMIR)*, pages 355–356, Vienna, Austria, September 2007. Österreichische Computer Gesellschaft.

[Ari09] Christopher Ariza. The interrogator as critic: The turing test and the evaluation of generative music systems. *Comput. Music J.*, 33(2):48–70, 2009.

[Ass96] MIDI Manufacturers Association. The Complete MIDI 1.0 Detailed Specification, 1996.

[Bil94] J. A. Biles. Genjam: A genetic algorithm for generating jazz solos. In *Proceedings of the International Computer Music Conference*, pages 131–137, Aarhus, Denmark, 1994.

[Cam97] Emilios Cambouropoulos. Musical rhythm: A formal model for determining local boundaries, accents and metre in a melodic surface. In *Music, Gestalt, and Computing - Studies in Cognitive and Systematic Musicology*, pages 277–293, London, UK, 1997. Springer-Verlag.

[Cam01] Emilios Cambouropoulos. The local boundary detection model (LBDM) and its application in the study of expressive timing. In *Proceedings of the International Computer Music Conference (ICMC'01)*, Havana, Cuba, September 2001.

[Cam06] Emilios Cambouropoulos. Musical parallelism and melodic segmentation: A computational approach. *Music Perception: An Interdisciplinary Journal*, 23(3):249–267, February 2006.

[Che04] Marc Chemillier. Toward a formal study of jazz chord sequences generated by Steedman's grammar. *Soft Computing*, 8(9):617–622, 2004.

[Coe02] Carlos A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, 2002.

[CSH09] Shih-Chuan Chiu, Man-Kwan Shan, and Jiun-Long Huang. Automatic system for the arrangement of piano reductions. In *Multimedia, 2009. ISM '09. 11th IEEE International Symposium on*, pages 459 –464, 14-16 2009.

[CV06] Jae-woo Chung and G. Scott Vercoe. The affective remixer: personalized music arranging. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 393–398, New York, NY, USA, 2006. ACM.

[Dc10] Daniel W. Dyer and contributors. Watchmaker framework for evolutionary computation. `http://watchmaker.uncommons.org/`, 2010. Accessed on May 20th, 2010.

[DST+03] Márcio Dahia, Hugo Santana, Ernesto Trajano, Carlos Sandroni, and Geber Ramalho. Generating rhythmic accompaniment for guitar: the Cyber-João case study. In *IX Brazilian Symposium on Computer Music: Music as Emergent Behaviour*, Campinas, Brazil, 2003.

[DST+04] Márcio Dahia, Hugo Santana, Ernesto Trajano, Geber Ramalho, Carlos Sandroni, and Giordano Cabral. Using patterns to generate rhythmic accompaniment for guitar. In *Sound and Music Computing Conference*, Paris, 2004.

[EMY08] Norio Emura, Masanobu Miura, and Masuzo Yanagida. A modular system generating jazz-style arrangement for a given set of a melody and its chord name sequence. *Acoustical science and technology*, 29(1):51–57, 2008.

[Fou10] Wikifonia Foundation. Wikifonia. `http://www.wikifonia.org/`, 2010. Accessed on September 14th, 2010.

[GB91] P. M. Gibson and J. A. Byrne. NEUROGEN, musical composition using genetic algorithms and cooperating neural networks. In *Second International Conference on Artificial Neural Networks*, pages 309–313, Bournemouth, UK, November 1991.

[GJC03] Andrew Gartland-Jones and Peter Copley. The suitability of genetic algorithms for musical composition. *Contemporary Music Review*, 22(3):43–55, September 2003.

[Goo06] Michael Good. Lessons from the adoption of MusicXML as an interchange standard. In *XML 2006 Conference Proceedings*, Boston, MA, December 2006.

[Jac95] Bruce L. Jacob. Composing with genetic algorithms. In *Proceedings of the International Computer Music Conference (ICMC)*, Banff, Alberta, September 1995.

[KM07] Robert M. Keller and David R. Morrison. A grammatical approach to automatic improvisation. In C. Spyridis, A. Georgaki, G. Kouroupetroglou, and C. Anagnostopoulou, editors, *Proceedings of the 4th Sound and Music Computing Conference (SMC07)*, Lefkada, Greece, July 2007.

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: a survey. *AI Mag.*, 13(1):32–44, 1992.

[LLC10] Recordare LLC. MusicXML definition, version 2.0. `http://www.recordare.com/xml.html`, 2010. Accessed on May 20th, 2010.

[Mar96] Andrew Martin. Reaction–diffusion systems for algorithmic composition. *Org. Sound*, 1(3):195–201, 1996.

[Mir01] Eduardo Miranda. *Composing Music with Computers (Music Technology)*. Focal Press, June 2001.

[Mit98] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[MO02] Massimo Melucci and Nicola Orio. Evaluating automatic melody segmentation aimed at music information retrieval. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 310–311, New York, NY, USA, 2002. ACM.

[MY02] Masanobu Miura and Masuzo Yanagida. Finger-position determination and tablature generation for novice guitar players. In Catherine Stevens, Denis Burnham, Gary McPherson, Emery Schubert, and James Renwick, editors, *Proceedings of the 7th International Conference on Music Perception and Cognition (ICMPC)*, pages 701–704, Sydney, July 2002. AMPS and Causal Productions.

[NK97] Tomomasa Nagashima and Jun Kawashima. Experimental study on arranging music by chaotic neural network. *International Journal of Intelligent Systems*, 12(4):323–339, 1997.

[Pac94] François Pachet. An object-oriented representation of pitch-classes, intervals, scales and chords. In *Actes des 1res Journées d'Informatique Musicale, Bordeaux res Journées d'Informatique Musicale, Bordeaux*, pages 19–34, 1994.

[Pac99] François Pachet. Surprising harmonies. *International Journal of Computing Anticipatory Systems*, 4, February 1999.

[Pac00] François Pachet. Computer analysis of jazz chord sequence: Is Solar a blues? In Eduardo Reck Miranda, editor, *Readings in Music and Artificial Intelligence*, pages 85–113. Harwood Academic Publishers, 2000.

[Pai08] Jean-François Paiement. *Probabilistic models for music*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, 2008.

[PEB05] Jean-François Paiement, Douglas Eck, and Samy Bengio. A Probabilistic Model for Chord Progressions. In *Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR)*, 2005. IDIAP-RR 05-57.

[PEB06] Jean-François Paiement, Douglas Eck, and Samy Bengio. Probabilistic melodic harmonization. In Luc Lamontagne and Mario Marchand, editors, *Canadian Conference on AI*, volume 4013 of *Lecture Notes in Computer Science*, pages 218–229. Springer, 2006.

[PEBB05] Jean-François Paiement, Douglas Eck, Samy Bengio, and David Barber. A graphical model for chord progressions embedded in a psychoacoustic space. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 641–648, New York, NY, USA, 2005. ACM.

[PGBE08] Jean-François Paiement, Yves Grandvalet, Samy Bengio, and Douglas Eck. A distance model for rhythms. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 736–743, Helsinki, Finland, 2008. ACM. IDIAP-RR 08-33.

[PR01] François Pachet and Pierre Roy. Musical harmonization with constraints: A survey. *Constraints*, 6(1):7–19, 2001.

[Rad06] Daniele P. Radicioni. *Computational Modeling of Fingering in Music Performance*. PhD thesis, Centro di Scienza Cognitiva, Università degli Studi di Torino, Torino, Italy, 2006.

[RAL04] Daniele P. Radicioni, Luca Anselma, and Vincenzo Lombardo. A Segmentation-Based Prototype to Compute String Instruments Fingering. In R. Parncutt, A. Kessler, and F. Zimmer, editors, *Proceedings of the 1st Conference on Interdisciplinary Musicology (CIM04)*, Graz, Austria, April 2004.

[RD04] Aleksander Radisavljevic and Peter Driessen. Path difference learning for guitar fingering problem. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, 2004.

[RIHL93] K. Ricanek II, A. Homaifar, and G. Lebby. Genetic algorithm composes music. In *System Theory, 1993. Proceedings SSST '93., Twenty-Fifth Southeastern Symposium on*, pages 223 –227, 7-9 1993.

[RL05a] Daniele P. Radicioni and Vincenzo Lombardo. Computational Model of Chord Fingering. In B.G. Bara, L. Barsalou, and M. Bucciarelli, editors, *Proceedings of the 27th Annual Conference of the Cognitive Science Society*, pages 1791–1796, Mahwah, New Jersey, 2005. Lawrence Erlbaum Associates.

[RL05b] Daniele P. Radicioni and Vincenzo Lombardo. Fingering for Music Performance. In *Proceedings of the International Computer Music Conference (ICMC05)*, pages 527–530, Barcelona, Spain, 2005.

[RL07] Daniele P. Radicioni and Vincenzo Lombardo. A Constraint-based Approach for Annotating Music Scores with Gestural Information. *Constraints*, 12(4):405–428, 2007.

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[TDSR04] Ernesto Trajano, Márcio Dahia, Hugo Santana, and Geber Ramalho. Automatic discovery of right hand fingering in guitar accompaniment. In *Proceedings of the International Computer Music Conference (ICMC'04)*, pages 722–725, 2004.

[TI00] Nao Tokui and Hitoshi Iba. Music composition with interactive evolutionary computation. In *Proceedings of Generative Art 2000, the 3rd International Conference on Generative Art*, Milan, Italy, 2000.

[Tou02] Godfried T. Toussaint. A mathematical analysis of african, brazilian and cuban clave rhythms. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 157–168, Townson, MD, July 2002.

[TP05] Daniel R. Tuohy and W. D. Potter. A genetic algorithm for the automatic generation of playable guitar tablature. In *Proceedings of the International Computer Music Conference (ICMC '05)*, Barcelona, Spain, September 2005.

[TP06a] Daniel R. Tuohy and W. D. Potter. An evolved neural network/HC hybrid for tablature creation in GA-based guitar arranging. In *Proceedings of the International Computer Music Conference (ICMC'06)*, New Orleans, Louisiana, November 2006.

[TP06b] Daniel R. Tuohy and W. D. Potter. GA-based music arranging for guitar. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC '06)*, pages 3810–3815, Vancouver, BC, Canada, July 2006.

[TP06c] Daniel R. Tuohy and W. D. Potter. Guitar tablature creation with neural networks and distributed genetic search. In *Proceedings of the 19th International Conference on IEA/AIE'06*, Lecture Notes in Artificial Intelligence, pages 244–253, Annecy, France, June 2006. Springer-Verlag, Berlin.

[TT08a] Eric Thul and Godfried T. Toussaint. Analysis of musical rhythm complexity measures in a cultural context. In *C3S2E '08: Proceedings of the 2008 C3S2E conference*, pages 1–9, New York, NY, USA, 2008. ACM.

[TT08b] Eric Thul and Godfried T. Toussaint. Rhythm complexity measures: A comparison of mathematical models of human perception and performance. In Juan Pablo Bello, Elaine Chew, and Douglas Turnbull, editors, *Proceedings of the 9th International Conference on Music Information Retrieval*, Philadelphia, PA, USA, September 2008.

[Tuo06] Daniel R. Tuohy. Creating tablature and arranging music for guitar with genetic algorithms and artificial neural networks. Master's thesis, University of Georgia, Athens, Georgia, 2006.

[Wik10] Wikipedia. Arrangement — Wikipedia, the free encyclopedia, 2010. Accessed May 22nd, 2010.

[WL97] Jeng-Feng Wang and Tsai-Yen Li. Generating guitar scores from a MIDI source. In *Proceedings of 1997 International Symposium on Multimedia Information Processing*, 1997.

[You39] Robert W. Young. Terminology for logarithmic frequency units. *The Journal of the Acoustical Society of America*, 11(1):134–139, 1939.

Erklärung zur Verfassung der Arbeit

Arnaud Moreau
Boerhaavegasse 23/21
A-1030 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Oktober 2010 _____

Unterschrift