Integer Programming For Optimal Right Hand Guitar Fingerings

Mohit Dubey, Matthew Skarha, and Isabel Taylor
Department of Mathematics,
Oberlin College, Oberlin, OH 44074, USA
mdubey, mskarha, or itaylor@oberlin.edu

Abstract

Integer Programming is a mathematical technique that has been used to solve linear optimization problems in a wide variety of fields. Here, we present a method for using an integer program to find optimal right hand fingerings, which could be applied generally to notated classical or fingerstyle guitar music. Our approach uses soft constraints formulated to represent "proper" classical guitar technique and minimizes a sum of penalties incurred by violating these constraints. Finally, we used the Gurobi Optimizer to implement our method for a short piece of music (extracted from the 1921 piece "La Catedral" by Agustin Barrios).

Introduction

Integer Programming (IP) problems, the family of linear programming problems in which variables are constrained to take on integer values, crop up in fields as far reaching as production planning, scheduling, and the playing of musical instruments. Given an objective function to minimize or maximize, integer or binary valued variables, and a set of constraints on those variables corresponding to physical or temporal limitations, optimal solutions to these problems can be found using a variety of algorithms (i.e. branch-and-bound or cutting planes). Playing a musical instrument, while seemingly less concrete than problems such as scheduling or production planning, can be easily formulated as an IP. In this case, the objective is to play a piece of music using the most efficient technique possible with variables corresponding to the use of specific fingers to play each note and constraints based upon human anatomy and the music itself. Previous work has used IP methods to find optimal fingerings for piano music [1] as well as optimal left-hand fingerings for the guitar [2] [3]. However, the problem of finding optimal right hand fingerings for classical or fingerstyle guitar music has yet to be explored. In this paper, we formulate a basic IP to solve for optimal right hand fingerings for notated guitar music based on the principles of "proper" right hand technique.

Right Hand Technique

Throughout history, the instrument known as the guitar has been used to play a wide variety of music around the world including, but not limited to, European classical music, flamenco, jazz, and fingerstyle. In this paper, we focus on the case of European classical music, fingerstyle, and other guitar music in which a pick (or "plectrum") is not used, giving particular emphasis to the techniques developed to play music notated in Western staff notation. In these styles, the guitar is

played using the right hand thumb, index, middle and ring fingers but not the pinky. While there have been many differing approaches to playing (famously, Ida Presti [4] and Alexandre Lagoya [5]) that have evolved over the years, at present there is a notion of "proper right hand technique" which can be thought of as a set of rules that describe the physically optimal way of using the right hand to play the guitar. These rules can be summarized as follows:

Rule 1: Do Not Repeat Fingers

It is never optimal to use the same right hand finger to play two notes in a row. This is due to the nature of the finger's motion away from the string, which requires it to reset before it can play again. It is far more efficient to use another finger to pluck the next note, giving time for the initially used finger to relax. This is especially true for passages of long successive notes, particularly fast scalar passages.

Rule 2: Thumb Plays the Bassline

Based on the way the right hand naturally rests on the guitar (Figure 1), there is a natural correspondence between the thumb and the (spatially speaking) top three strings (usually tuned E, A, and D) of the guitar [1]. Due to this fact, it is far more optimal to use the thumb to play the lowest frequency pitches of a given passage of music, which is usually called the bassline.

Rule 3: Fingers Stay in Natural Position

In a similar manner, the natural resting position of the right hand provides a natural correspondence between the index, middle and ring fingers and the bottom three strings of the guitar respectively. While it is considered optimal to try to maintain this natural correspondence, this is often considered a more relaxed constraint, as it must often be broken in order to accomplish musical goals such as playing scales or tremolo sections in which two or three right hand fingers must be used on one or two strings in alternation.

Rule 4: Avoid Backwards Crossings

The fourth rule governing proper right hand technique pertains to the alternation of fingers. In order to maintain the natural resting position of the hand, it is natural for fingers further right on the right hand to play strings lower down (vertically) on the guitar. It is therefore more optimal to play a sequence of treble strings that get progressively lower (i.e. G, B, e) with the index, middle then ring finger (in that order). Violation of this directionality principle is referred to as a "backwards cross", as it requires the fingers to cross behind one another in order to played vertically higher strings that are outside of the natural resting position of the hand.

Rule 5: Avoid Ring-Middle-Ring Alternation

The final rule governing proper right hand technique pertains to the alternation of specifically the middle and ring fingers. Due to the shared tendons between the middle and ring fingers, it is far more difficult to alternate them in succession than the index and middle or index and ring fingers. Therefore, it is highly non-optimal to play with the ring and middle fingers in alternation.

[1] Throughout this paper we will refer to strings in terms of their vertical position, rather than their relative pitch "height" in order to avoid confusion.

Figure 1: *The natural resting position of the right hand for playing guitar using "proper" technique. The thumb can also equivalently be placed on any of the other three top (bass) strings.*

Mathematical Model

In our model we will assume that the left-hand fingering for the guitar has already been solved for. In this case, a piece of notated music can be reduced to a sequence of string numbers, s_i, that need to be played in a certain order by a sequence of fingers, f_i. We choose to number the strings of the guitar in ascending order vertically and the fingers of the right hand in ascending order from right-to-left (Figure 2), such that the numbers are aligned for the natural resting position of the hand (i.e. finger 1, the ring finger, naturally rests on string 1, the high e string).

Figure 2: *Diagrams of the chosen assignment of fingers and strings to numerical values.*

In summary:

- s_i is the string played on timestep i, where $1 \leq s_i \leq 6$
- x_i is the number of the finger played on timestep i, where $1 \leq f_i \leq 4$

In addition to these, we define "helper" variables as follows:

- t_i is a binary variable which is 1 iff $f_i = 4$, 0 otherwise
- p_i^{BC} is a binary penalty variable equal to 1 if we violate rule 4 by beginning a **B**ackwards **C**rossing at step i, 0 otherwise
- p_i^T is a binary penalty variable equal to 1 if we violate rule 2 by playing the treble notes with the **T**humb, 0 otherwise
- p_i^F is a penalty variable for violating rule 3, keeping **F**ingers in natural position, where $0 \leq p_i^F$. It is equivalent to the difference between the string being played and the finger playing it.

And one helper piece of data:

- b_i is binary data equal to 1 if $s_i \leq 3$, 0 otherwise

Integer Program

A few notes: we express certain values with absolute value signs. We recognize this is not linear, and respond that these can be linearized with well-known techniques. For the sake of clarity, we leave them as they are.

$$\min \sum_{i=1}^{n} p_i^{BC} + p_i^T + \frac{1}{2} p_i^F \tag{1}$$

subject to:

$$b_i * f_i < 4 + p_i^T \qquad \forall\, i \tag{2}$$

$$|f_i - s_i| - (t_i * M) \leq p_i^F \qquad \forall\, i \tag{3}$$

$$-(s_i - s_{i+1})(f_i - f_{i+1}) \leq M * p_i^{BC} \qquad \forall\, i < n-1 \tag{4}$$

$$f_i + f_{i+1} + f_{i+2} \geq 5 \forall\, i < n-2 \tag{5}$$

$$|f_i - f_{i+1}| \geq 1 \qquad \forall\, i < n-1 \tag{6}$$

Now, an explanation of the formulation.

(1) Objective Function To explain the objective function, I must also explain the theory behind our constraints. Guitar fingering is more an art-form than a strict rule adherence, and occasionally the need to break the rules we have previously outlined comes in handy, or is even necessary. Traditionally in integer programming, constraints are "hard," and to break a rule implies that an optimal solution is not possible. In our case, we wanted "soft" constraints - we don't want to break them, but it is still valid if we do occasionally. To mitigate this, we introduce the idea of penalty variables that get activated when we violate a constraint. This objective function is attempting to minimize the number of times we violate our soft constraints. Also note the weight of 1/2 in front of p_i^F, which biases the program towards breaking rule 3 over the others.

(2) "Thumb Plays Bassline" Constraint To encourage the thumb to play the bassline, we penalize the program only when the thumb fingers play the treble strings. To do so, we have the "activation" data b_i to trigger when we play on treble strings. Then, we simply want this to be less than 4 (aka played by the non-thumb fingers). If it is 4, meaning we play it with the thumb, we impose a penalty.

(3) "Fingers Stay in Natural Position" Constraint To encourage the fingers to stay in their "natural" position, we essentially want to minimize the difference between the string played on time-step i and the finger that plays it (since in "natural" position the fingers' numbers align with the strings') We do this by looking at the difference between the finger and the string, and equating that to the penalty. Since we don't want to penalize when the thumb plays, we add in the $-(t_i * M)$, where M is some large constant. This means when the thumb is used, this constraint is always satisfied, and no penalty will be enacted.

(4) "Avoid Backwards Crossings" Constraint A backwards crossing happens when, when given two different strings, you use fingers in the opposite order. In other words, if the difference between string i and string i+1 is positive, a backwards crossing happens when the difference between finger i and finger i+1 is negative, and vice versa. Since the strings are data, we can multiply the two differences $(s_i - s_{i+1})(f_i - f_{i+1})$. Then, we want this value to be positive (meaning no backwards crossing) and need to penalize it when it is negative (there exists a backwards crossing). To do so, we use an "activation" penalty variable, p_i^{BC}. When the value is negative, we want to activate the penalty. We do this by associating the penalty with a large constant M. When $(s_i - s_{i+1})(f_i - f_{i+1})$ is positive (aka no backwards crossings), we negate it, so it is a negative value and less than or equal to 0. When the value is negative (a backwards crossing exists), the negated value is positive, forcing us to activate the penalty to maintain the constraint. A few notes for this constraint: we do not care when the value of $(s_i - s_{i+1})(f_i - f_{i+1})$ is zero, as this would only happen when either string i and i+i are the same (and a backwards crossing is impossible), or when we use the same finger twice, which we prevent using another constraint.

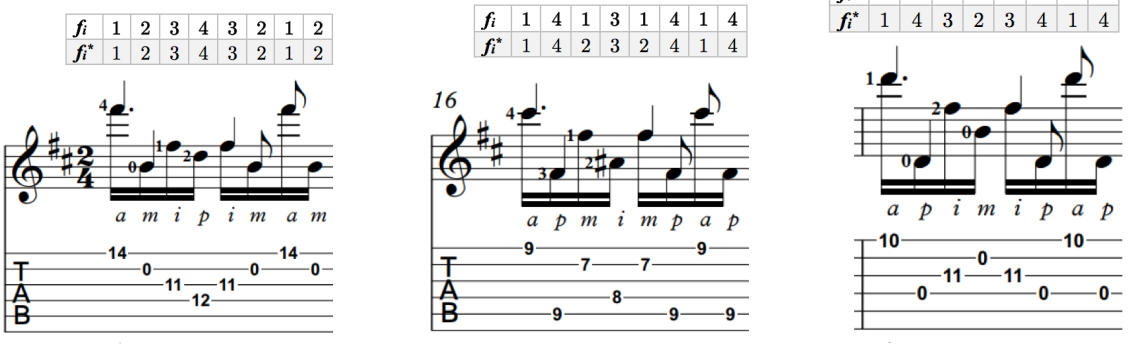
(5) "Avoid Ring-Middle-Ring Alteration" Constraint This constraint is fairly self-explanatory. Since we are limiting the smallest possible triplet of 1-2-1 fingerings, we simply force all solutions to have sequences of fingerings greater than or equal to 5, cutting off this pattern as a possibility.

(6) "Do Not Repeat Fingers" Constraint Once again, this constraint is fairly self-explanatory. For every time-step i and i+i, we want $f_i \neq f_{i+1}$ and so we ensure the difference between f_i and f_{i+1} is at least 1, once again using the absolute value as shorthand notation. We encode this as a hard constraint as we never want this to happen.

Implementation and Results

We implemented our model in Gurobi, using eight measure-long excerpts from "La Catedral" by Agustin Barrios as the input. Currently, our model is only optimized for scale-based guitar music where each timestep contains exactly one note of equal length. Barrios' "La Catedral" satisfies these conditions while also providing a wide range of musical approaches. Additionally, we used a "La Catedral" transcription by classical guitarist and teacher Daniel F. Savarese that included tablature and fingerings as the optimal version against which to test our model [6].

For each measure, the f_i correspond to the fingering of each note, as returned by our model, and the f_i^* correspond to the optimal fingering of each note, as notated by Savarese.



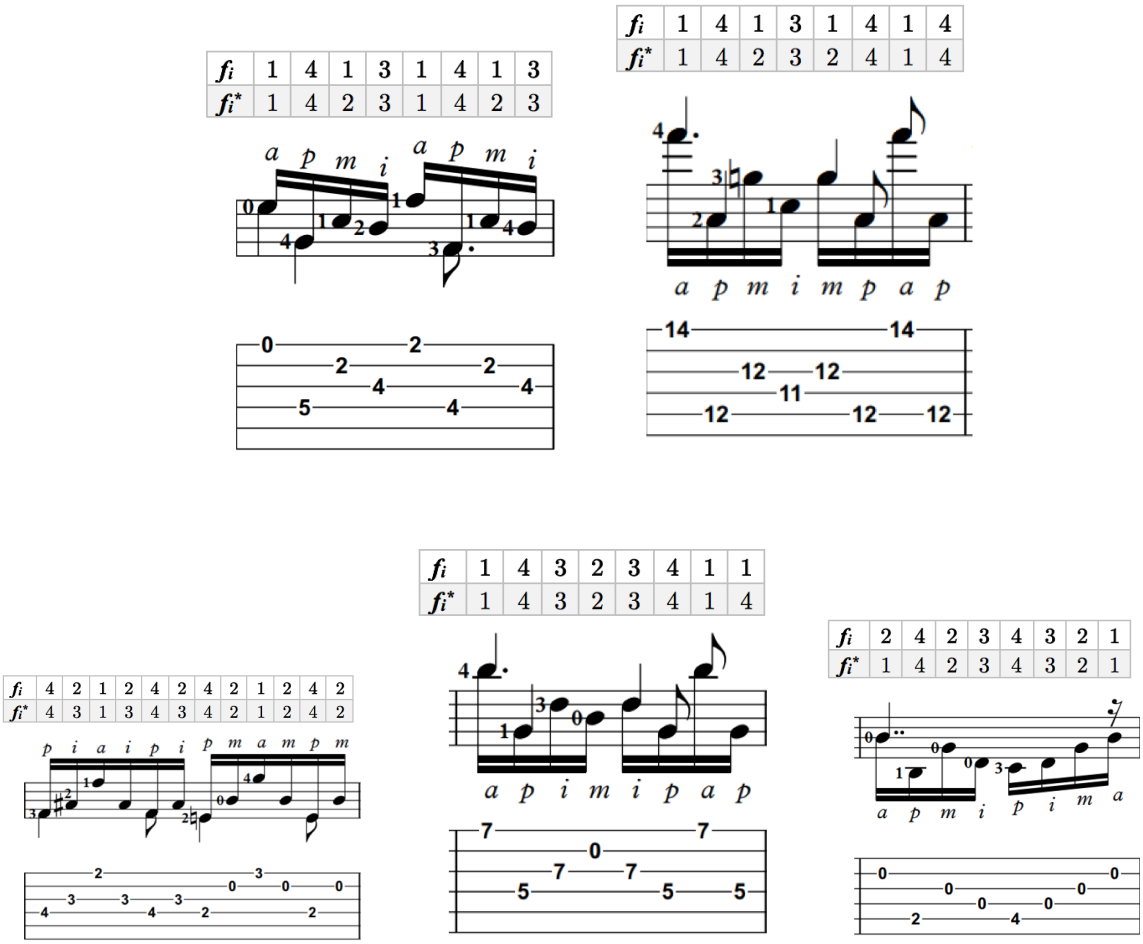


Figure 3: *Results of implementing our model in Gurobi compared with optimal fingerings according to professional classical guitarist Daniel F. Savarese for eight selected measures in Barrios' "La Catedral".*

The eight measures sum to a total of 68 individual notes against which we tested our model. Our model correctly identified the fingering for 53 of those notes, corresponding to a 78% accuracy rate. Additionally, several of the fingerings that our model incorrectly identified are individual preferences that are made by the musician (e.g. 3-2-3 vs. 2-1-2). Thus, the fingerings produced by our model are largely acceptable and would only need a small amount of human revision.

Conclusions

We have shown a technique for producing optimal right hand guitar fingerings for scale-based fingerstyle European classical music. Our integer program takes a sequence of strings to be played as input and produces the optimal right hand fingerings for each note as output. This has the potential to significantly decrease the amount of time classical guitarists spend on right hand fingerings, which is often a trivial, yet arduous task.

Further work on this topic would expand our research to include chord-based music as well as music with varying note durations.

References

[1] M. Hart, R. Bosch, and E. Tsai. Finding optimal piano fingerings. The UMAP (Undergraduate Mathematics and Its Applications) Journal, 21:167177, 2000.

[2] A. Radisavljevic, P. Driessen. Path Difference Learning for Guitar Fingering Problem. Proceedings of the International Computer Music Conference. Miami, USA, 2004.

[3] B. Tahon. Fingers to frets - A Mathematical Approach Optimizing Finger Assignment for Guitar Tablature. Master's Thesis, KU Leuvin. 2017.

[4] C. Mowbray. Ida Presti as a Solo Performer and Composer of Works for Solo Guitar. Doctor of Musical Art's Thesis, Shenandoah Conservatory. 2012.

[5] A. Kozinn. Alexandre Lagoya Dies at 70; Innovative Classical Guitarist. The New York Times. 1999.

[6] D. F. Savarese. La Catedral by Agustin Barrios. `https://www.savarese.org/music/LaCatedral.html`. Last accessed: December 20, 2018.

Appendix

Here we include the Python code used to generate the Gurobi LP file. The s dictionary at the beginning of the code is where one would input the sequence of strings to be played.

```
import sys
sys.stdout = open("coins.lp", "w")
s = {
    1: 2,
    2: 5,
    3: 3,
    4: 4,
    5: 5,
    6: 4,
    7: 3,
    8: 2
}
b = {}
for i in range(1,len(s)+1):
    if s[i]<=3:
        b[i] = 1
    else:
        b[i] = 0
print("Minimize")
for i in range(1,len(s)+1): # objective function
    print("\t0 F" + str(i) + " +")
for i in range(1,len(s)+1): # objective function
    if i != len(s):
        print("\tPBC"+str(i)+" + PT"+str(i)+" + .5 PF"+str(i)+
        " + PB" + str(i) + " +")
```

```
    else:
        print("\tPBC"+str(i)+" + PT"+str(i)+" + .5 PF"+str(i)+
        " + PB" + str(i))
print("Subject To")
for i in range(1,len(s)+1):
    if b[i] == 1:
        print("\tF"+str(i)+" + PT"+str(i)+" <= 3")
for i in range(1,len(s)+1):
    print("\tF"+str(i)+" - 100 T" + str(i) + " - PF" + str(i) +
    " <= " + str(s[i]))
for i in range(1,len(s)+1):
    print("\t-F"+str(i) + " - 100 T" + str(i) + " - PF" +
    str(i) + " <= "+str(-s[i]))
for i in range(1,len(s)+1):
    print("\tF"+str(i)+" - T" + str(i)+ " <= 3")
for i in range(1,len(s)+1):
    print("\t4 T"+str(i)+" - F"+str(i)+" <= 0")
for i in range(1,len(s)):
    d = s[i] - s[i+1]
    print("\t"+str(-d) + " F" + str(i) + " + " + str(d) +
    " F" + str(i+1) + " - 100 PBC" + str(i) + " <= 0")
for i in range(1,len(s)):
    print("\tF" + str(i) + " - F" + str(i+1) +
    " - 100 Y" + str(i) + " <= -.5")
for i in range(1,len(s)):
    print("\tF" + str(i) + " - F" + str(i+1) +
    " - 100 Y" + str(i) + " >= -99.5")
for i in range(1,len(s)-2):
    print("\tF" + str(i) + " + F" + str(i+1) + " + F"
    + str(i+2) + " >= 5")
for i in range(1,len(s)+1):
    if s[i] == 6:
        print("\tF"+str(i)+" - PB" + str(i) + " >= 4")
for i in range(1,len(s)+1):
    if s[i] == 5:
        print("\tF"+str(i)+" - PB" + str(i) + " >= 4")
for i in range(3,len(s)-1):
    if s[i] == 4:
        if s[i+1] <= 4 and s[i+2] <= 4 and s[i-1] <=4
        and s[i-2] <= 4:
            print("\tF"+str(i)+" - PB" + str(i) +
            " >= 4")
print("Bounds")
for i in range(1,len(s)+1):
    print("\t1 <= F" + str(i) + " <= 4")
```

```
for i in range(1,len(s)+1):
    print("\t0 <= T" + str(i) + " <= 1")
for i in range(1,len(s)+1):
    print("\t0 <= PBC" + str(i) + " <= 1")
for i in range(1,len(s)+1):
    print("\t0 <= PT" + str(i) + " <= 1")
for i in range(1,len(s)+1):
    print("\t0 <= PB" + str(i) + " <= 1")
for i in range(1,len(s)+1):
    print("\t0 <= PF" + str(i) + " <= 5")
for i in range(1,len(s)+1):
    print("\t0 <= Y" + str(i) + " <= 1")
print("Integers")
for i in range(1,len(s)+1):
    print("\tF" + str(i))
for i in range(1,len(s)+1):
    print("\tT" + str(i))
for i in range(1,len(s)+1):
    print("\tPBC" + str(i))
for i in range(1,len(s)+1):
    print("\tPB" + str(i))
for i in range(1,len(s)+1):
    print("\tPT" + str(i))
for i in range(1,len(s)+1):
    print("\tPF" + str(i))
for i in range(1,len(s)+1):
    print("\tY" + str(i))
print("End")
```