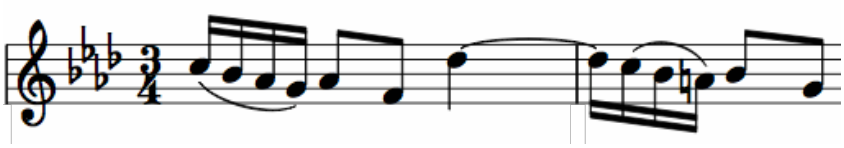
Mark Peng
Stephen Koo

Piano Fingering Generation with Linear CRFs

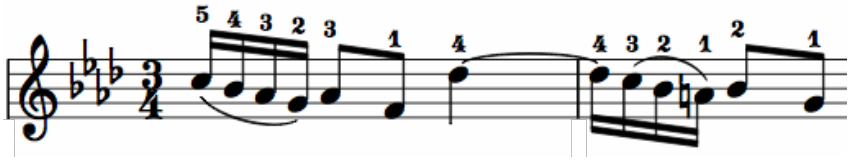
Introduction

Task Definition

A common challenge for beginning piano students is how to determine the proper fingerings for a given passage of music--that is, which finger does one use for each note in a sequence of notes in music. What would it take for a computer to learn how to assign reasonable fingerings? We aim to build such a system. More specifically, our system will take as input a sequence of unfingered notes:



and generate the fingerings for them as would be played on the piano with the right hand:



Constructing our system will involve three component tasks:

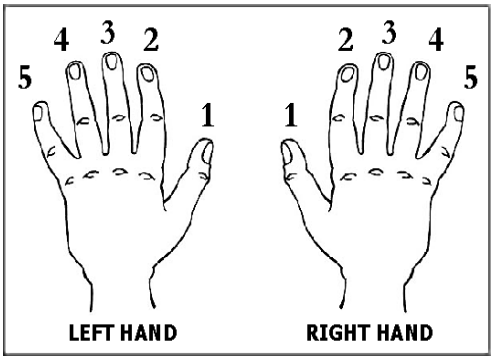
1. Parsing through sheet music that is represented by Lilypond notation in order to create our training and development sets.
2. Setting up the learning and inference machinery to learn and generate fingerings.
3. Design and test feature extractors that sufficiently capture the characteristics of the notes and fingerings.

Terminology and Definitions

For the rest of this paper, we adopt the following conventions for representing notes and

fingerings.

Fingerings



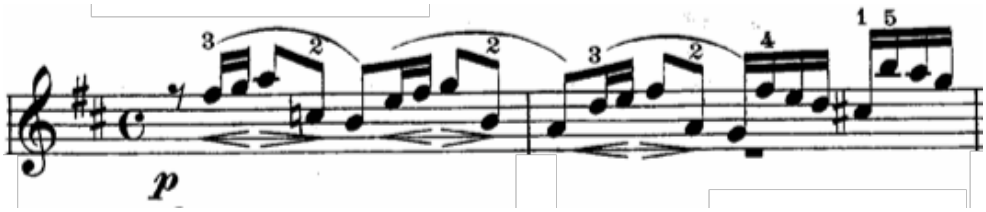
In standard piano pedagogy, each finger is notated with a natural number, as shown in the diagram on the right. As we will focus solely on the right hand within the scope of this project, there is a one-to-one correspondence between the integers in the range 1 through 5 and the fingers on the right hand.

Notes and Octaves

As we use LilyPond syntax as the format for our data, we will also adopt it as our notation for pitches. Most pitch classes are denoted by their standard Western names: c, d, e, f, g, a, b. To avoid using special characters, sharps and flats are denoted by the suffixes “is” and “es,” respectively, with double sharps and double flat additionally indicated by “isis” and “eses.” Representing octaves is slightly trickier, as we chose to use the LilyPond relative octave entry scheme in order to reduce the clutter in the notation. Relative octave entry implies that the octave of each note depends on the octave of the previous note. From the LilyPond documentation, “in relative mode, each note is assumed to be as close to the previous note as possible.”[1] Larger jumps in pitch are indicated by commas and single-quotes, with each comma indicating that the note should be an octave lower, and each single-quote indicating that the note should be an octave higher. The very first note in a sequence is represented in absolute octave mode, with each comma or single-quote indicating an octave offset from the third octave (C3 through B3 in scientific pitch notation).

As an example, here is an excerpt from J. S. Bach’s Sinfonia No. 3, BWV 789, translated into LilyPond syntax. Note that we omit indications of note duration and other details in the score, and that fingerings can be indicated by a dash followed by the finger number.

[1] http://lilypond.org/doc/v2.16/Documentation/notation/writing-pitches



```
fis''-3 g a c,-2 b e fis g b,-2 a d-3
e fis a,-2 g fis'-4 e d cis-1 b'-5 a g
```

Approach

Data Parsing and Representation

We began by attempting to parse large collections of professionally-produced LilyPond pianos scores for fingerings. However, the complexity of the documents, which involved macros and irrelevant structural information about the scores, turned out to be too difficult to navigate for the scope of the project. We resorted to manually cleaning the data, leaving only the note sequences of the right-hand parts.

The parsing itself then consists of extracting the pitch class and fingering (using regular expresions), and computing the octaves for the notes in the sequences. For each note, we transform pitch class and octave into an internal integer representation of the absolute pitch, which allows us to efficiently compute intervals between pitches through straightfoward substraction. Fingerings are also stored as integers, allowing for distances between fingers to be estimated as well in our model.

Baseline Method: Unigram Frequency Analysis

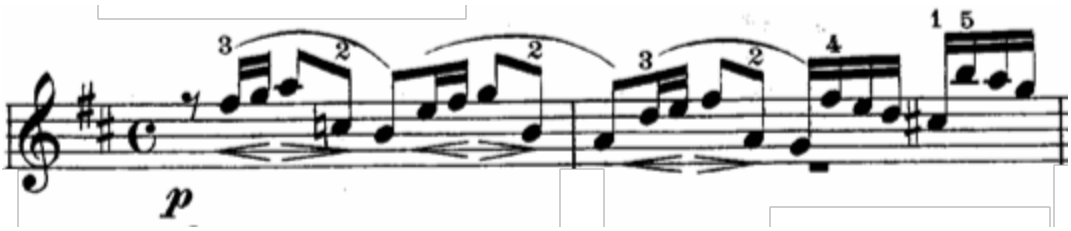
As a naïve baseline approach, we tested out a unigram frequency analysis model to learn and generate fingerings. In this model, we choose each fingering based on the most common fingering used for each pitch class.

$$y = \underset{f \in \{1,2,3,4,5\}}{\arg\max} \; |\{\text{note } x : x \text{ uses fingering } f\}|$$

We constructed a system that parsed LilyPond files for fingerings and then computed histograms of the fingerings for each pitch class. We parsed 329 LilyPond piano scores downloaded from the Mutopia Project, with the frequencies computed for the note C shown below as an example:

Finger	Count
1	63
2	29
3	40
4	30
5	22

generated a set of note-to-fingering mappings that were used to generate the fingerings for the passage from Bach's Sinfonia No. 3 below:



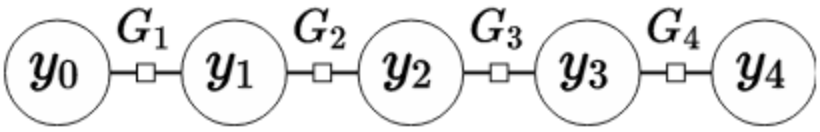
Note (L.P.)	fis''	g	a	c,	b	e	fis	g	b,	a	d	e	fis	a,	g	fis'	e	d	cis	b'	a	g
Pro Fingering	3			2					2		3			2		4			1	5		
B.L. Fingering	5	5	5	5	1	5	5	5	1	1	5	5	5	1	1	5	5	5	3	5	5	5

The notes from the passage are represented in Lilypond notation in the above table, and the fingerings that our baseline system generated are displayed in the bottom row of the table. As you can clearly see, the baseline system produced highly inaccurate fingerings that made it seem like we only have three fingers on our right hand and most commonly use our pinky to play the piano. It is also clear that these fingerings (marked in red) do not match up with the professional fingerings above the notes—with the exception of the 'b'. So, simply using the

highest frequency fingering for each note was evidently too naïve, and we needed to take a more sophisticated approach. On the positive side, our baseline system demonstrated that our data parsing worked.

Advanced Method: Linear Chain CRF

We decided to use a variable-based model, specifically linear-chain conditional random fields (CRFs), for our system. Conceptually, this makes sense since a linear chain of variables can be made to correspond to a sequence of unfingered notes. Each variable y_i represents the fingering for a ith note, and has the domain $\{1, 2, 3, 4, 5\}$. Binary factors G_i between consecutive variables in the factor graph represent soft constraints on the assignments to adjacent fingerings.



To choose the best assignment for a given sequence of notes $x_1, x_2, ..., x_n$, we perform inference on the chain, using either the Viterbi algorithm or Gibbs sampling, to maximize the weight of the assignment.

$$\text{Weight}(y) = \prod_{i=2}^{n} G_i(y_{i-1}, y_i)$$

$$y' = \arg\max_{y} \text{Weight}(y)$$

How do we define these factors? Rather than defining the factor functions directly using domain knowledge, we instead attempt to learn these factor functions. To do so, we define local feature vectors as a function of each pair of adjacent notes and their fingerings, with each factor G_i defined as the dot product of the local feature vector φ_i and a set of learned parameters θ. Letting x be the sequence of notes themselves, we have

$$G_i(y_{i-1}, y_i) = \theta \cdot \phi_i(y_{i-1}, y_i; x)$$

To learn the values of the parameters, we apply a stochastic gradient descent algorithm to maximize the likelihood of our training data.

$$\max_{\theta} \left[\log \mathbb{P}(y = y_{\text{train}}; \theta)\right]$$

Feature Extraction

With the standard learning and inference frameworks in place, the challenge for us was to design the features. In the spirit of named-entity recognition, we utilize mostly indicator feature templates on the adjacent notes and their fingerings. We avoided unary feature functions, since, as previous experiments demonstrated, frequency counts on the notes and fingerings give no helpful indication of the best fingerings. Instead we hoped the various atomic binary feature functions would sufficiently capture some of the ways the notes and fingerings interacted.

We also included a few more special features, based on a pianist's typical considerations for choosing fingerings. The `STRETCH` feature quantifies the amount by which the hand stretches over the two notes by taking the ratio between the pitch interval and the finger separation, with naïve Laplace smoothing to prevent divide-by-zero errors. In order to better enforce linearity of this value, the feature is templated and its label is dependent on whether or not one of the fingers used in the pair is the thumb (1), since the thumb allows a much wider reach than any other finger on the hand.

A couple additional indicator features provide flags for cases where fingers cross over without using the thumb—a situation almost always avoided by pianists.

Letting x'_i represent the pitch class of the note x_i, we have implemented the following features for our feature extractor. Variables and expressions in the feature label signify feature templates, with all features in the feature space defaulting to zero unless otherwise defined here.

Look-back Features

Feature Label	Value
(PAIR, y_{i-1}, y_i)	1
(PREV, y_i, x'_{i-1})	1

(GAP, y_i, $x_i - x_{i-1}$)	1
(GAP_, y_{i-1}, $x_i - x_{i-1}$)	1
(STRETCH, $[y_{i-1} = 1 \vee y_i = 1]$)	$\frac{\lvert x_i - x_{i-1}\rvert + 1}{\lvert y_i - y_{i-1}\rvert + 1}$
(MAP, x'_{i-1}, $x_i - x_{i-1}$, y_{i-1}, y_i)	1
(CROSSDOWN)	$\mathbb{I}[y_{i-1} \neq 1 \wedge y_i > y_{i-1} \wedge x_i < x_{i-1}]$
(CROSSUP)	$\mathbb{I}[y_i \neq 1 \wedge y_i < y_{i-1} \wedge x_i > x_{i-1}]$

Look-ahead Features

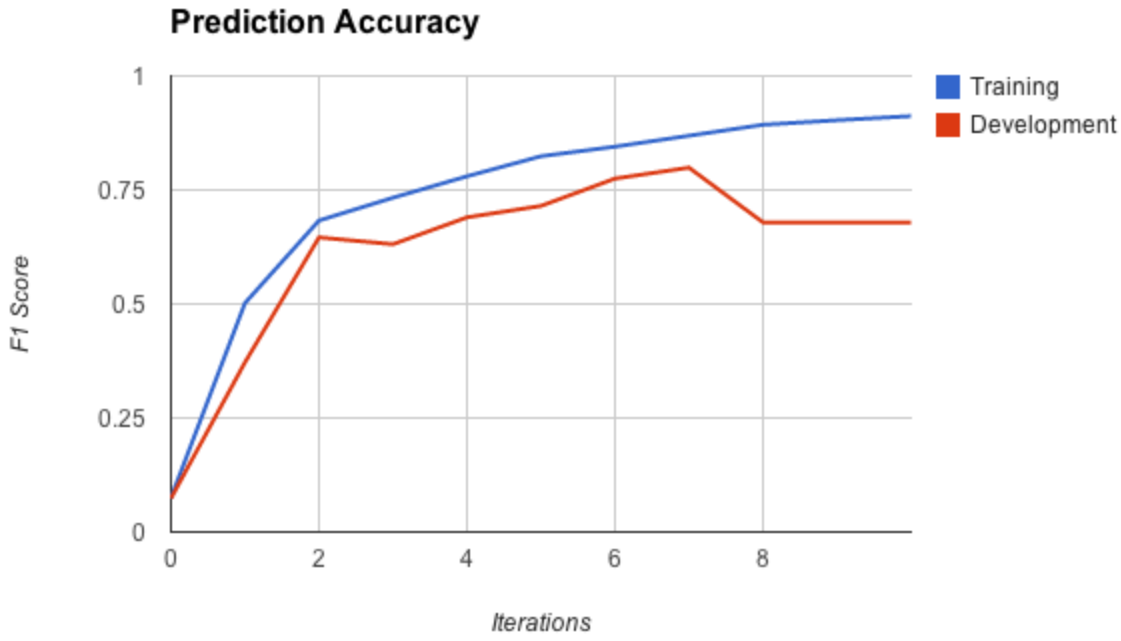
Feature Label	Value
(NEXT, y_i, x'_{i+1})	1
(GAP+, y_i, $x_{i+1} - x_i$)	1

Data and Experiments

One of the major challenges in building this system was finding data to feed the trainer. There does not currently exist a widely available dataset of piano music labeled with fingerings. A large number of professionally edited piano scores in the public domain can be found on the IMSLP Petrucci Music Library online, but the scores are invariably in PDF format and so cannot easily be parsed. One ray of hope came from the Mutopia Project, which offers a collection of classical musical scores typeset in LilyPond, a digital music engraving program analogous to LaTeX. The notes and fingerings are represented textually in the markup files, allowing for at least some amount of parsing. However, the fingerings provided in the Mutopia scores are sparse, as piano players typically only need occasional fingering suggestions, which means that the number of fully labeled consecutive sequences of notes comes out to be quite low. We needed a passages of music that were comprehensively labeled with fingerings, and thus we decided to manually create a dataset ourselves.

Our training dataset consists of the right-hand parts of the first four Two-Part Inventions by Johann Sebastian Bach, completely labeled with fingerings and notated in LilyPond syntax. We chose the Two-Part Inventions specifically because Bach employs a single line for each hand, which simplifies the task and allows us to disregard chords and focus on one note at a time. Each of the pieces are segmented into a number of continuous passages, each varying from several notes to dozens of notes, according to the phrasal structure of the music. Thus in total 37 passages (with a total of 1114 labeled notes) comprises our current training set. Our development set includes arbitrary partial samples from these passages, as well as other manually created examples such as scales.

We ran our training algorithm (employing Gibbs sampling with 10,000 samples) for 10 iterations, and produced the following training and development set F1 scores.



End-to-End Trials

Now we are ready to test out our system and the parameters learned.

Evaluation Criteria

How do we evaluate the quality of the fingerings generated by our program? Fingering decisions in general are quite subjective, and professional opinions often differ on the same music, sometimes greatly. Thus we establish two computationally verifiable grades of quality.

- **Baseline:** The produced fingerings must be physically possible. Validation is possible through simple rule check on finger crossovers: if the notes are ascending and fingering numbers decrease, the succeeding finger must be the thumb; if the notes are descending and the fingering numbers increase, the preceding finger must be the thumb.
- **Optimal**: The produced fingerings match perfectly with the professional labeled fingerings.

For example, consider the sequence of notes, [c, d, e, f, g], and the corresponding fingerings in the below table.

Professional/ Optimal	1	2	3	4	5
Legal/Baseline	1	2	3	1	2
Illegal	1	2	3	2	3

The first two rows of fingerings do not have any illegal finger crosses, whereas the last row has the illegal finger cross highlighted in red. Our system must generate fingerings of the first two varieties but not of the third in order to satisfy our evaluation criteria.

Results

We ran our development set (188 notes) through our program and computed the percentages of the resulting fingerings that fall into each of these grades.

Grade	Percentage of Development Set Satisfied
Baseline	100.0%
Optimal	55.8%

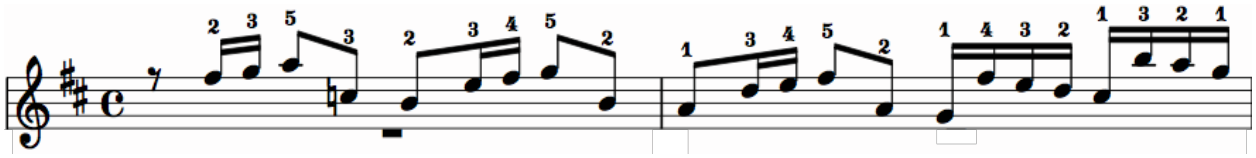
Analysis

Overfitting and Generalization

The pattern of training and development set accuracies over the learning iterations exhibit some interesting behavior. The training accuracy steadily converges to a rather high value of 0.912, which indicates that the feature extractor captures some characteristics of the training data well enough that the inference algorithms can make fairly accurate predictions within the training data. However, the development accuracy does not follow the trend as neatly, and in fact drops off after reaching a local maximum of 0.799 at the 7th iteration. This is perhaps a case of overfitting, which we can combat with a combination of reducing the number of training iterations and continuing to refine the features to generalize better to new examples. We may also simply need a larger and more varied training set: Bach Inventions are great, but the small collection we have so far is evidently still insufficient for deployment.

Accuracy

The results of our baseline evaluation, at 100% satisfaction, indicate that it is possible to learn and generate fingerings that are better than random, and physically playable at that. The percentage of the development predictions that met the optimal criteria, 55.8%, also indicates significantly better than random predictions, especially given that many possible valid fingerings exist for a given passage at the professional level. On closer inspection, however, there are many more challenges to tackle. We return to our example from Sinfonia No. 3, on which we now run the Viterbi inference algorithm to produce the following fingerings:



The fingerings are actually quite usable, with no unnavigable sequences. In fact the latter half is almost exactly how a professional piano player might approach the passage. There are however some issues: for example, the first three notes employs 2-3-5 for three consecutive adjacent notes, in which the jump from 3 to 5 is rather unnecessary. The use of 3 and 2 for the fourth and fifth notes, respectively, is also somewhat non-ideal in comparison to the possibility of using 2 and 1. We conjecture that the main underlying reason behind these mishaps is the inability of the linear chain CRF model to properly take into account long-range dependencies in fingerings: for example, the choice of fingerings on the fourth and fifth notes can affect the playability of the following three notes. However, the current model only considers constraints between adjacent notes and attempts to determine a global optimum from these myopic constraints. How might a new model properly weight the different holistic concerns for fingerings across an entire passage to determine the most elegant of choices? Perhaps it will remain a secret among the ranks of Artur Rubinstein and Vladimir Horowitz, but armed with our new tools and further research, we think not.