HMM-Based Voice Separation of MIDI Performance

Andrew McLeod
University of Edinburgh

Mark Steedman
University of Edinburgh

Abstract

Voice separation is an important component of Music Information Retrieval (MIR). In this paper, we present an HMM which can be used to separate music performance data in the form of MIDI into monophonic voices. It works on two basic principles: that consecutive notes within a single voice will tend to occur on similar pitches, and that there are short (if any) temporal gaps between them. We also present an incremental algorithm which can perform inference on the model efficiently. We show that our approach achieves a significant improvement over existing approaches, when run on a corpus of 78 J. S. Bach compositions, each of which has been separated into the gold standard voices suggested by the original score. We also show that it can be used to perform voice separation on live MIDI data without an appreciable loss in accuracy.

Keywords: voice, perception, music analysis, software

1 Introduction

Voice separation consists of the allocation of the notes of a given piece of music into melodic streams; however, it can be difficult to define precisely what constitutes a musical voice. (See Cambouropoulos (2008) for a full discussion on the different possible definitions). Cognitively, a voice simply refers to any set of notes which a listener may hear as a coherent melody, no matter whether that stream is monophonic (consisting of only non-overlapping notes) or polyphonic (containing any number of simultaneous notes). From a computational standpoint, however, it can be useful to restrict the definition of a voice to strictly monophonic streams of notes, especially when performing voice separation as a preprocessing step for other MIR tasks.

For example, existing work on rhythmic structure detection (van der Weij, 2012), pattern detection (Hsu, Liu, & Chen, 2001; de León & Inesta, 2007), query-by-tapping (Peters, Cukierman, Anthony, & Schwartz, 2006; Hanna & Robine, 2009), and query-by-humming (Birmingham, Dannenberg, & Pardo, 2006; Ryynanen & Klapuri, 2008) all require monophonic MIDI data as input. Furthermore, even among MIR techniques which can be run on polyphonic data, many still perform better when run on monophonic input. For example, Bruderer, McKinney, and Kohlrausch (2012) showed that song segmentation is consistently more accurate when run on monophonic input. Additionally, voice separation could be useful as the first step towards a general music model, and it was shown by Berg-Kirkpatrick, Andreas, and Klein (2014) that the use of such a model improves the accuracy of music transcription significantly.

However, acquiring appropriate monophonic MIDI data is not always trivial, especially when looking to evaluate a solution on real world data. There are some computer generated MIDI data online already separated into monophonic voices, some of which we have used to evaluate our model, but these are usually quantized and not from real performances. Therefore, the applicability of such evaluations to real world performance is not entirely clear.

In this work, we introduce a model which is able to separate polyphonic MIDI data, including real world and live MIDI performance data, into strictly monophonic voices which can be used by MIR methods such as those listed above. It can be run as a preprocessing step to those existing methods, or as a standalone voice separation model on the MIDI data itself. Our model allows for some limited overlap between consecutive notes within a single monophonic voice, and we show that using this feature, its accuracy remains high when separating live MIDI data into voices.

In Section 2, we will look briefly at some existing solutions to the voice separation problem, especially those which separate MIDI data into strictly monophonic voices, specifically noting when there are principles involved on which we also based our model. The model itself, along with an algorithm which can be used to perform inference on our model efficiently, is described in depth in Section 3, while Section 4 contains an evaluation of our model's performance. Finally, we draw some conclusions and propose future work in Section 5.

2 Related Work

Huron (2001) and Tymoczko (2008) investigate voice leading rules—rules which govern how voices evolve over time within a single piece—from a cognitive perspective, and offer valuable insights into possible voice separation rules, many of which we have applied to our model. Huron's *Common Tone Rule*, *Chordal Tone Rule*, and *Avoid Leaps Rule* all suggest that large melodic intervals between consecutive notes within a single voice should be avoided, a property which Tymoczko calls *efficient voice leading*. Likewise, Huron's *Part-Crossing Rule* and *Part Overlap Rule* suggest that the notes of two separate voices should not cross in pitch, nor should the notes of one voice become higher than the notes of another voice that was once higher than it, or vice versa, concepts referred to by Tymoczko as *voice crossings*. (The distinction between the two cases is subtle, but both are important: in the first case both voices have notes which cross each other, while in the second case, only one voice has notes, yet those notes cross the most recent note from the voice which has no concurrent notes.) A third principle suggested by Huron is that of temporal continuity—the idea that the stream of notes should be relatively constant within a single voice, and not have too many gaps of silence. Temperley (2008) applied many of these concepts into constructing a successful probabilistic model of melodic perception, a task closely related to voice separation.

There are already existing algorithms which perform voice separation, many of which are based on some of Huron and Tymoczyko's principles, though not all of them were designed to be used for the same purpose as our model. Kirlin and Utgoff (2005), for example, designed a system for voice separation; however, it uses features related to a song's time signature and tempo which requires it to be run on computer generated rather than live MIDI data. Karydis, Nanopoulos, Papadopoulos, Cambouropoulos, and Manolopoulos (2007) describe a more cognitive voice separation model which attempts to separate MIDI data into polyphonic voices, something we want to avoid. The approach first suggested by Kilian and Hoos (2002) and later expanded upon in a paper by Kilian (2004) has a large number of parameters which must be adjusted by the user at run time to produce the desired separation results, and therefore seems to be most useful as an aid to manual transcription of MIDI.

While the methods above cannot be applied directly to monophonic voice separation on live MIDI data, that is not to say that all of the concepts and techniques used by them are irrelevant to the problem. Kilian (2004), for example, use a Gaussian window function in their evaluation of voice continuity, something which we use as well in our model.

Chew and Wu (2005) propose a heuristic based solution to the monophonic voice separation problem. Quantized MIDI data are separated into chronological sections called "contigs," each of which represents a time period in a song during which there is a constant number of co-occurring notes. The notes within these contigs are then separated into monophonic sequences of notes called "fragments." Within a contig, no two fragments may cross in pitch. To join fragments from consecutive contigs into voices, a global optimization approach is used which minimizes the total pitch difference between consecutive notes in each resulting voice.

This approach performs well in general, but it has two weaknesses. First, since it performs a global optimization, it cannot be run in real time on live input. This may not seem like a problem when voice separation, a relatively computationally light task, is the only thing being computed on a song, but when it is only the first task in a series of other, more complicated MIR tasks, it could become more important. The second drawback is that it will always group a contig with n co-occurring notes into exactly n voices; however, there are cases where this is incorrect. For example, in Figure 1, an excerpt from Bach's 15th Invention (BWV 786), the first three notes will be grouped into a single monophonic contig, and then into a single voice, even though the proper separation would be to assign the first note to one voice, and group the second and third notes together in a different voice.

Madsen and Widmer (2006) propose a solution based on pitch proximity which uses a small lookahead and tests all possible combination of grouping notes into voices within that lookahead. However, the completeness values they report in evaluation are substantially lower than the average voice consistency results reported by Chew and Wu (2005) when run on the same MIDI data. Madsen and Widmer themselves note that the two metrics are comparable (2006).

Duane and Pardo (2009) proposed a heuristic solution where each note in an input MIDI file is treated as a node on graph, and edges are added to that graph grouping the corresponding notes into voices. Constraints are placed on which edges can be added to the graph so that the resulting voices are always monophonic. The program decides which edges to add based on a weight function dependent on each note pair's pitch and temporal difference. Nodes are first grouped into segments based on note onset times, and edges are added between nodes within each segment. Then, the segments themselves are tied together with edges based on another weighting function.

3 Proposed Solution

Our model, an HMM, is loosely based on Huron and Tymoczko's principles of pitch closeness and temporal continuity as described in Section 2 above. The model itself is presented in Section 3.1, and we present an algorithm which can be used to perform inference on our model incrementally in Section 3.2. An example of our model being run on some MIDI data is explained in Section 3.3 to make it more clear.

Our solution has a few advantages over some of the existing solutions. For one, it allows for notes within a single voice to overlap slightly, which eliminates the need to perform any preprocessing such as quantization on MIDI input as some other solutions require. Additionally, our model can be run incrementally since it does not set the number of voices in a song to some constant value before beginning separation (many existing algorithms set this to the greatest number of concurrent notes in the song). The fact that our model does not constrain voices to non-overlapping notes allows it to be run on MIDI data generated from live performances, while the incrementality of our algorithm allows it to be run in real time.

Before getting into any more details, it will be useful to define the relevant properties of a MIDI note n: number, onset time, and offset time. A note's number $\mathrm{Num}(n)$ is an integer on the range $[0-127]$ representing its pitch, where 72 corresponds to C4. A note's onset time $\mathrm{On}(n)$ represents the number of microseconds after the beginning of the song the note was played. Similarly, a note's offset time $\mathrm{Off}(n)$, represents the number of microseconds after the beginning of the song the note stopped being played. It is also useful to define the duration of a note as the difference of its offset and onset times as in Eqn (1).

$$\mathrm{Dur}(n) = \mathrm{Off}(n) - \mathrm{On}(n) \tag{1}$$

3.1 Model

Our model is an HMM where each state S represents a list of monophonic voices V_i. A voice V is a list of notes $n_{1 \to n}$ ordered by onset time, where $\text{On}(n_i) < \text{On}(n_{i+1})$. Within a single state, no two voices may contain the same note. Furthermore, since we will apply our model to live MIDI performance data, rather than just computer generated MIDI, we allow for some minimal overlap between consecutive notes within a voice. Specifically, we allow notes n_i and n_{i+1} overlap if and only if Eqns (2) and (3) are both satisfied. Eqn (2) ensures that the duration of the overlap comprises at most half of the duration of the first note involved, while Eqn (3) ensures that the overlap does not continue for the entirety of the second note.

$$\text{Off}(n_i) - \text{On}(n_{i+1}) \leq \frac{\text{Dur}(n_i)}{2} \tag{2}$$

$$\text{Off}(n_i) < \text{Off}(n_{i+1}) \tag{3}$$

Each voice V also has a pitch, calculated by the function given in Eqn (4), where l is a tunable constant. A voice's pitch is simply a weighted average of its l most recent notes.

$$\text{Pitch}(V) = \frac{\sum_{i=0}^{\min(l,|V|)} (2^i * \text{Num}(n_{|V|-i}))}{\sum_{i=0}^{\min(l,|V|)} 2^i} \tag{4}$$

Since the onset and offset times of each note is unbounded, and there are likewise an unbounded number of notes in a given MIDI file, the state-space of our model is of infinite size. Thus, instead of using discrete transition and emission probabilities, we use transition and emission functions.

The emission function for each state is entirely deterministic; each state has exactly one possible emission with probability 1, although multiple states do have identical emissions. Each state outputs a set N of MIDI notes, where each note within N has an equal onset time. That is, $\forall n, n' \in N, \text{On}(n) = \text{On}(n')$. Specifically, a state S outputs a set N of all MIDI notes n contained in any voice $V \in S$ whose onset time is the maximum value of that of any $n \in V \in S$. That is, N is the set of all $n \in V \in S$ which satisfy Eqn (5).

$$\text{On}(n) = \text{Max}(\text{On}(n)),\ \forall n \in V, \forall V \in S \tag{5}$$

Before we define our transition function, we must define precisely what transitions exist within our model. A state S has a transition to state S' if and only if the following two conditions are satisfied: (1) the set of notes contained by any voice in S' which are not contained by any voice in S must be exactly the set of notes determined by the emission function of S' as defined above; and (2) removing those notes from the voices of S', and then removing any voices from S' which become empty as a result, must produce exactly the voices of S, and these voices must appear in exactly the same order.

This transition from S to S' is represented by $T_{S,N,W}$, where S is the original state, N is a list of the notes from the emission function of S' ordered by increasing pitch, and W is a list of integers with exactly one element for each note $n_i \in N$, where w_i represents the voice $V \in S$ to which the corresponding note n_i should be added. No two w_i should be equal, and $\forall w_i, \text{abs}(w_i) \leq |S| + \text{Count}(w_i < 0)$. Each $T_{S,N,W}$ represents one individual note transition for each (n_i, w_i) pair. These note transitions are handled in order of increasing $\text{abs}(w_i)$, and the value of w_i represents the following: if $w_i < 0$, add n_i to a new voice V inserted at the w_ith index of S; if $w_i > 0$, add n_i to the existing voice $V_{w_i} \in S$.

We can now define the probability of a given transition $T_{S,N,W}$ as simply the product of the probabilities of each individual note transition within it, as shown in Eqn (6), times an order score which penalizes a note being added to a voice out of pitch order.

$$\text{P}(T_{S,N,W}) = \prod_{0 \leq i \leq |N|} \text{P}(S, n_i, w_i) * \text{order}(S, n_i, w_i) \tag{6}$$

The order function by default returns 1, but that value is divided by two for each of the following cases that applies:

1. $|w| > 1$ and $\text{Pitch}(V_{|w|-1}) > \text{Num}(n)$
2. $0 < w < |S|$ and $\text{Pitch}(V_{w+1}) < \text{Num}(n)$
3. $-|S| \leq w < 0$ and $\text{Pitch}(V_{|w|}) < \text{Num}(n)$

Case 1 is applicable when a note will be added to a voice and the preceding voice in the state (if one exists) has a greater pitch than the number of the note, while cases 2 and 3 are mutually exclusive based on the sign of w, but each applies when a note will be added to a voice and the succeeding voice in the state (if one exists) has a lower pitch than the number of the note.

The probability of each individual note transition is the product of its pitch score and its gap score, or a tunable constant if the note will be added to a new voice, as shown in Eqn (7).

$$\text{P}(S, n, w) = \begin{cases} \text{pitch}(S, n, w) * \text{gap}(S, n, w) & w > 0 \\ s_{new} & w < 0 \end{cases} \tag{7}$$

A note transition's pitch score is computed using the Gaussian window function as shown in Eqns (8) and (9), where σ_p is a tunable parameter. A note transition's gap score is computed using the max function shown in Eqn (10) on the result of the logarithmic function in Eqn (11), where σ_g and g_{min} are both tunable parameters.

$$\text{pitch}(S, n, w) = \text{Gauss}(\text{Num}(n) - \text{Pitch}(V_w), \sigma_p) \tag{8}$$

$$\text{Gauss}(\mu, \sigma) = e^{-\frac{1}{2}(\frac{\mu}{\sigma})^2} \tag{9}$$

$$\text{gap}(S, n, w) = \text{Max}\left(\text{g}(S, n, w), g_{min}\right) \tag{10}$$

$$\text{g}(S, n, w) = \ln\left(-\frac{\text{On}(n) - \text{Off}(\text{last}(V_w))}{\sigma_g} + 1\right) + 1 \tag{11}$$

Note that the offset time of the note within a note is note used in any of the above equations, so any two notes with equal pitch and onset time are treated as equally likely by our model, regardless of their offset times (given that each transition will create a valid resulting state). Note also that the transition probabilities out of a given state do not necessarily sum to 1, but this is not important, as the values can be normalized with a constant factor.

3.2 Inference

To find the most likely final state given our observed note sets we use a slightly modified Viterbi algorithm. (See Viterbi (1967) for an overview of the algorithm.) Our observed data are the notes found in a given MIDI file, ordered by onset time. If multiple notes have equal onset time, they are observed as a set.

We made two modifications to the Viterbi algorithm, each related to reducing the search space of the algorithm, since the size of the search space increases exponentially without any constraints. The first modification is applied when adding a note into a new voice. Specifically, we only check those indices w which have the order score for any valid index w. In practice, this assures that we only try to insert a new voice where it is roughly in pitch order with the surrounding voices. Secondly, we have decided to use beam search with a tunable beam size b. After each iteration of the algorithm, we save only the b most likely states given the observed data to that point.

3.3 Example

To help make our model more clear, we will now go through an example of the Viterbi algorithm being run on it, given the MIDI data shown in Figure 2 (and assuming some reasonable setting of the parameters). Here, the notes have been color-coded according to the voice to which each will be assigned by our model. For a diagram of this example, see Figure 3, where each note n is represented as $[\text{Num}(n), \text{On}(n)]$, and voices within a state hypothesis are grouped using braces. For simplicity, we will use a beam size of 2 in this example.

The initial state, S_0, is empty, as no notes have been observed yet. After seeing N_1, there is no decision to be made, as the only valid transition is to add the observed note into a new voice. Upon observing N_2, we again have only one valid transition to check. We cannot add this new note into the existing voice because it fails the overlap constraint in Eqn (2), and the new voice must be placed at index 2 of our state due to it being out of pitch order with the existing V_1, because of the optimization mentioned in Section 3.2.

Once we observe N_3, however, there are a few possible state transitions to check. Trivially, we could add the new note into a new voice at index 2 (the other indices would be out of pitch order again), but assuming that s_{new} has been set sufficiently low, this transition will not be saved due to our beam size of 2. The new note could also be added to the either of the existing voices. Both the pitch score from Eqn (8) and the gap score from Eqn (10) are slightly greater when adding the new note to V_2, so we assign that transition a higher probability.

Next, we observe N_4, and out of each current hypothesis state, besides adding any new voices, the only valid transition is that which adds the new note into the voice which does not currently contain the note $[71, 78]$, due to Eqn (2). The difference between the two remaining transition's probabilities is about a factor of 2, since performing the given transition on the current most likely hypothesis state introduces a pitch crossing, and therefore has an order score of $1/2$ in Eqn (6), due to case 1. Therefore, the two current hypothesis states each perform the transition and then switch in order, as indicated by the arrows in the diagram.

Finally, we observe N_5, and out of each current hypothesis state, besides adding any new voices, the only valid transition is that which adds the new note into the voice which does not currently contain the note $[70, 120]$, due to Eqn (3). So, we perform the valid transition on each hypothesis state, and the orders will not change due to the transition probabilities being relatively close to each other.

4 Evaluation

4.1 Corpora

We evaluated our model on six distinct corpora:

1. The 15 two-part inventions by J. S. Bach.[1]
2. The 15 three-part sinfonias by J. S. Bach.[1]
3. The 48 fugues from *The Well-Tempered Clavier, Book 1 (WTC I)* by J. S. Bach.[2]
4. The 24 fugues from *The Well-Tempered Clavier, Book 2 (WTC II)* by J. S. Bach.[2]
5. The 28 movements from *String Quartets*, Op. 1, by J. Haydn.[3]
6. The 19 live performances of J. S. Bach inventions (5), fugues (5), and preludes (9) (from *WTC I* and *WTC II*) from the CrestMusePEDB, introduced by Hashida, Matsui, and Katayose (2008).

The first five corpora all consist of computer generated MIDI data, which are separated into multiple tracks within each MIDI file. Each track corresponds to an individual fugal voice in the Bach compositions (as suggested by the original

[1]The inventions and the sinfonias were acquired from `www.imslp.org`.
[2]The fugues were acquired from `www.musedata.org`.
[3]The quartets were acquired from `www.kunstderfuge.com`

scores). The tracks were used as gold standard voices for these five corpora. In the case of the sixth corpus, we separated the MIDI data into tracks ourselves manually, each corresponding to a voice as suggested by the original scores. In the Haydn quartets, the tracks were separated by instruments, and we used these as the gold standard, as Duane and Pardo (2009) also did; however, these gold standard voices may not be entirely correct (as they also noted), since the melody often switches between instruments, especially between the two violin parts.

In rare cases, most often on the final note of a piece, a single voice in a may contain a chord. This is a problem for our gold standard since we want to separate the pieces into strictly monophonic voices. In such cases, we manually removed all but the lowest-pitched note in the chord. This makes musical sense, since it has been suggested by Dixon (2001) among others that a notes with lower pitches are more salient than those with higher pitches.

The inventions are the simplest of the compositions, each containing exactly two voices. The sinfonias are slightly more complicated, containing three voices each, and the fugues are the most complicated of the Bach compositions, sometimes even containing more than four voices. The quartets each consist of exactly four parts, but they can be more complicated than the fugues in that the different parts, especially the two violin parts, are liable to cross in pitch during a piece. The live performances are difficult in that the data is not clean. That is, note onsets which, in the score, occur immediately after another note's offset, may not occur precisely at that time. This adds some noise into the data and makes voice separation more difficult.

4.2 Metrics

We use two different evaluation metrics: Average Voice Consistency, as introduced by Chew and Wu (2005); and F-measure, as used by Duane and Pardo (2009). Specifically, we use Average Voice Consistency to compare against Chew and Wu's results, as we were unable to get their implementation. They evaluated on only the first four corpora of our data. We also report F-measure values for all of our corpora, and use it to compare against Duane and Pardo's results. They originally evaluated on the first five corpora we used, but we have their implementation and will also report their results on our sixth corpus.

Before we can explain Average Voice Consistency, a couple of definitions are needed. First, let voice(n) be the ground truth voice to which a note n belongs. Second, let voice(V) be the voice to which the majority of note $n \in V$ belong. Then the Voice Consistency of a voice VC(V) is given by Eqn (12).

$$\mathrm{VC}(V) = \frac{|\{n \in V : \mathrm{voice}(n) = \mathrm{voice}(V)\}|}{|V|} \tag{12}$$

The Average Voice Consistency of a given voice separation hypothesis state S is simply an average of the Voice Consistencies of every voice $V \in S$ as shown in Eqn (13).

$$\mathrm{AVC}(S) = 100 * \frac{\sum_{V \in S} \mathrm{VC}(V)}{|S|} \tag{13}$$

The F-measure we use is just the standard F-measure, where we treat the voice separation problem as one of binary classification where between each pair of notes, our model must decide whether the two notes occur consecutively within a single voice or not. Then, the F-measure is calculated by Eqn (14).

$$\textit{F-measure} = 2\frac{\textit{precision} * \textit{recall}}{\textit{precision} + \textit{recall}} \tag{14}$$

4.3 Training

To set the train the different parameters for our model, we used a grid search. For each of the parameters, we manually set a minimum and a maximum value, both inclusive. For some, we also set a minimum step size, limiting the number of values to check even if the grid size we picked was very small. See Table 1 for these values for each of our model's parameters. For all training, our beam size is restricted to 10. This is done to speed up training time, though during testing we use a beam size of 25.

We use different training set splits for each of our test sets to avoid overfitting. When evaluating on the inventions, we trained on the sinfonias and the fugues from both books one and two. When evaluating on the sinfonias, we trained on the inventions and the fugues from both books one and two. When evaluating on the fugues from books one and two, we trained on the inventions and the sinfonias. When evaluating on the Haydn quartets, we used leave-one-out cross validation between the six different quartets within the corpus. That is, for each of the six quartets, we trained on the other five when evaluating on the sixth. When evaluating on the live corpus from the CrestMusePEDB, we trained on the computer generated inventions, sinfonias, and fugues from both book one and two. The parameter setting used when evaluating each set of MIDI data are shown in Table 2.

4.4 Results

First, we will present our Average Voice Consistency results on Bach's inventions, sinfonias, and fugues from WTC I & II, and compare them to those reported by Chew and Wu (2005). They evaluated on the same pieces, though they handled chords by not separating the last few notes of each of their MIDI performances. Still, the results are be comparable, and they are shown in Table 3. It is important to note that the scores found in this table are averaged by piece. That is, a value of 99.29 on the Inventions means that the mean Average Voice Consistency over all 15 Inventions was 99.29. The overall score is also a per-piece average.

Our model sees a significant improvement over Chew and Wu's results, especially in the fugues. Both our improvement there and our overall improvement are statistically significant at the 0.05 level. We don't see much improvement on the inventions or the sinfonias, most likely due to those pieces being simpler than the fugues. With only two or three parts, there are fewer mistakes to be made, and therefore, there is much less room for improvement.

Here, we will look more closely at one simple case where our model outperforms Chew and Wu's, specifically the 15th invention (BWV 786), which was mentioned in Section 2 above. The first bar's MIDI notation is reproduced here in Figure 4. The white notes are all part of one voice, and the black notes are all part of a second voice. Both our model and Chew and Wu's program get this correct. However, while Chew and Wu's program incorrectly groups the gray note with the white notes, our model is able to correctly group it with the black notes. This is entirely due to Chew and Wu's constraint that portions of a song with the same number of simultaneous notes (in this case one), must be grouped into exactly that many voices.

Next, we will present our model's F-measure results, and compare them against those which we got by running Duane and Pardo's (2009) program on our data. We see definite improvement over their results among the Bach piano pieces, significantly for the sinfonias and the fugues, as shown in Table 4. On the Haydn quartets, however, our model performs slightly (though not significantly) worse than theirs, though we believe this to be due to the problem with the gold standard as mentioned in Section 4.1 above. On the live Bach performances, we again see significant improvement, this time on both the inventions and the fugues.

Diving more deeply into the results, it appears that most of our improvement of Duane and Pardo's work has come because our model more aggressively joins notes together into voices. That is, most of the errors that their program has which ours corrects are false negatives on their part. This occurs most often when a voice contains a rest, and is therefore absent from a piece for a beat or more. For example, in bars 18 and 19 of the first fugue from the Well-Tempered Clavier (BWV 846), one of the voices rests for two beats. This is shown in Figure 5, where the the bold notes are those which our model correctly joins, but Duane and Pardo's program fails to.

An example where our model separates voices incorrectly is in the 8th bar of the fugue from the 41st fugue in the Well-Tempered Clavier (BWV 886). A MIDI representation of that bar, in which the correct voices have been color-coded, is shown in Figure 6. The difficulty here is that the highest voice (white) ends at the exact same time that the lowest voice (black) begins. Rather than starting a new voice, our model incorrectly shifts each existing voice down one as shown by the arrows in the figure.

One thing which might help our model in separating the voices properly in cases such as this is looking for repeated patterns in the music. In the piece from this example, each time a new voice enters, it plays the same pattern of notes for two bars. A model which was able to detect such patterns would be able to recognize that pattern occurring in the lowest voice and infer that those notes likely belong to a new voice. Additionally, the second-to-lowest voice (dark gray) is still playing the tail end of that pattern, and thus should probably continue the pattern as the dark gray notes in the figure do.

Another example of when our model errs is when two voices cross. Such cases are difficult in general, given that the tendency of voices is not to cross; however, sometimes, there are enough rhythmic or harmonic clues that a model should be able to detect that the voices are crossing. For example, in the 73rd bar of the fourth fugue in the Well-Tempered Clavier (BWV 849), the two middle voices cross. (The two voices in question are reproduced, color-coded, in Figure 7.) The white voice contains only half notes, while the black voice contains only 8th notes. A model which is able to take such rhythmic information into account should be able to detect the correct voice separation in this case.

5 Conclusion

In this paper, we have presented a new model for separating polyphonic MIDI data into a set of monophonic voices, and argued that this sort of model can play a central role in other MIR tasks. We have shown that our model achieves a significant improvement over an existing solutions on a corpus of Inventions, Sinfonias, and Fugues by J.S. Bach, and shown that it still achieves good results when run on live performance data.

One advantage of our model is that it can be evaluated incrementally, and can therefore be run in real time,before the entire piece is available. In addition, it is more flexible than existing approaches in not defining the number of voices solely as the number of concurrent notes in a given song. Rather, it takes into account other factors which may align more closely with that song's true voice separation.

A shortcoming of the model is that it is easily confused by long range dependencies. For example, if there is a long rest in one or more voices, our model sometimes has trouble deciding where to assign the notes which follow the rest. It was shown by Granroth-Wilding and Steedman (2014), that long range harmonic dependencies do exist in music, and can be parsed successfully. One thing which might solve this and other mistakes which our model makes would again be to incorporate some knowledge of rhythmic and melodic patterns into it, as mentioned in relation to Figures 6 and 7 above.

We plan to update our model to use learned transition probability distributions rather than our somewhat naive Gaussian window and log score functions. We were unable to learn transition probabilities because of our small data set and large state space, but with a larger corpus of data, we will be able to apply machine learning to more closely approximate the true probability distributions of transitions with different pitch and gap differences, which could significantly improve our model's performance.

References

Berg-Kirkpatrick, T., Andreas, J., & Klein, D. (2014). Unsupervised Transcription of Piano Music. In *Advances in neural information processing systems* (pp. 1538–1546).

Birmingham, W., Dannenberg, R., & Pardo, B. (2006). Query by humming with the vocalsearch system. *Communications of the ACM*, *49*(8), 49–52.

Bruderer, M., McKinney, M., & Kohlrausch, A. (2012). Perceptual evaluation of musicological cues for automatic song segmentation. *Psychomusicology: Music, Mind and Brain*, *22*(1), 3.

Cambouropoulos, E. (2008, September). Voice And Stream: Perceptual And Computational Modeling Of Voice Separation. *Music Perception*, *26*(1), 75–94.

Chew, E., & Wu, X. (2005). Separating voices in polyphonic music: A contig mapping approach. In U. Wiil (Ed.), *Computer music modeling and retrieval* (Vol. 3310, pp. 1–20). Berlin, Heidelberg: Springer.

de León, P., & Inesta, J. (2007). Pattern recognition approach for music style identification using shallow statistical descriptors. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *37*(2), 248–257.

Dixon, S. (2001, March). Automatic Extraction of Tempo and Beat From Expressive Performances. *Journal of New Music Research*, *30*(1), 39–58.

Duane, B., & Pardo, B. (2009). Streaming from MIDI using constraint satisfaction optimization and sequence alignment. *Proceedings of the International Computer Music Conference*.

Granroth-Wilding, M., & Steedman, M. (2014, June). A Robust Parser-Interpreter for Jazz Chord Sequences. *Journal of New Music Research*, *43*(4), 355–374.

Hanna, P., & Robine, M. (2009). Query by tapping system based on alignment algorithm. In *Acoustics, speech and signal processing, 2009. icassp 2009. ieee international conference on* (pp. 1881–1884).

Hashida, M., Matsui, T., & Katayose, H. (2008). A New Music Database Describing Deviation Information of Performance Expressions. *International Conference of Music Information Retrieval (ISMIR)*, 489–494.

Hsu, J., Liu, C., & Chen, A. (2001). Discovering nontrivial repeating patterns in music data. *Multimedia, IEEE Transactions on*, *3*(3), 311–325.

Huron, D. (2001, September). Tone and voice: A derivation of the rules of voice-leading from perceptual principles. *Music Perception*, *19*(1), 1–64.

Karydis, I., Nanopoulos, A., Papadopoulos, A., Cambouropoulos, E., & Manolopoulos, Y. (2007). Horizontal and vertical integration/segregation in auditory streaming: a voice separation algorithm for symbolic musical data. In *Proceedings 4th sound and music computing conference (smc2007).*

Kilian, J. (2004). *Inferring Score Level Musical Information From Low-Level Musical Data* (Unpublished doctoral dissertation). TU Darmstadt.

Kilian, J., & Hoos, H. (2002). Voice separation-a local optimization approach. In *Ismir.*

Kirlin, P., & Utgoff, P. (2005). VOISE: Learning to Segregate Voices in Explicit and Implicit Polyphony. *Proceedings of the Sixth International Conference* *on Music Information Retrieval*.

Madsen, S. T., & Widmer, G. (2006). Separating voices in MIDI. *ISMIR*, 57–60.

Peters, G., Cukierman, D., Anthony, C., & Schwartz, M. (2006). Online music search by tapping. In *Ambient intelligence in everyday life* (pp. 178–197). Springer.

Ryynanen, M., & Klapuri, A. (2008). Query by humming of midi and audio using locality sensitive hashing. In *Acoustics, speech and signal processing, 2008. icassp 2008. ieee international conference on* (pp. 2249–2252).

Temperley, D. (2008, February). A Probabilistic Model of Melody Perception. *Cognitive Science*, *32*(2), 418–444.

Tymoczko, D. (2008, March). Scale Theory, Serial Theory and Voice Leading. *Music Analysis*, *27*(1), 1–49.

van der Weij, B. (2012). *Subdivision-Based Parsing of Expressively Performed Rhythms* (Unpublished master's thesis). University of Edinburgh.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, *13*(2), 260–269.