Separating voices in MIDI

Søren Tjagvad Madsen
Austrian Research Institute for Artificial Intelligence
Freyung 6/6
A-1010 Vienna, Austria
soren.madsen@ofai.at

Gerhard Widmer
Department of Computational Perception
Johannes Kepler University, Altenbergerstraße 69
A-4040 Linz, Austria
gerhard.widmer@jku.at

Abstract

This paper presents an algorithm for converting midi events into logical voices. The algorithm is fundamentally based on the pitch proximity principle. New heuristics are introduced and evaluated in order to handle unsolved situations. The algorithm is tested on ground truth data: inventions and fugues by J. S. Bach. Due to its left to right processing it also runs on real time input.

Keywords: Voice separation, Stream separation

1. Introduction

Voice separation or stream separation is the task of dividing a set of notes into most likely voices or auditory streams as it was coined by Bregman [1]. Adding voice information to notes is essentially adding structure to complex data. An obvious application that can benefit from a voice separation algorithm is music transcription, but a reliable stream separation algorithm could also be useful in music analysis systems. For MIR systems based on monophonic techniques – such MIR applications could be query by humming or theme finding – voice information is a prerequisite. This particular separation algorithm is intended to be used for online analysis of performed music.

2. Related work

A handful of existing approaches for separating voices has been published.

Cambouropoulos [2] and Kilian and Hoos [3] propose algorithms aiming at transcription applications. The latter method uses a stochastic local search based on a parametric cost function. The main feature of this approach is the ability to assign chords to a single voice.

Other algorithms aim at being able to reproduce the voices as they are perceived or exactly as they were written in the score. Kirlin and Utgoff [4] propose a data driven approach where a same-voice predicate is learned and later used to separate new music.

Chew and Wu [5] attack the problem by splitting the piece into 'contigs' where the number of voices is constant. Contigs are then combined using pitch proximity.

Temperley [6] lists well-formedness rules and preference rules (GTTM style [7]) and uses dynamic programming to minimize the cost function balancing the preferences. Our method is highly inspired by this approach as we shall see in Section 4.

The algorithms differ in the way they determine the number of voices they will produce. The methods described by Cambouropoulos, and Chew and Wu use as many voices as there are notes in the largest chord in the input file. Kilian and Hoos require an input value of desired maximum number of voices voices, whereas Kirlin and Utgoff and Temperley decide this dynamically.

In order to be able to do real time voice separation, we will pursue the approach of 'left to right' processing, as well as dynamically opening and closing voices as they are needed.

2.1. Gestalt principle of proximity

All methods are in some way or another dependent on the principle of pitch proximity when grouping notes into sequences.

If two note sequences are played simultaneously but separated somewhat in pitch, we tend to perceive them as two separate voices – the notes sequentially group together in two melodies. This is even true when the notes are played in alternation (see Figure 1). Pseudo-polyphony is the term for this phenomenon when one sequence of notes gives the impression of two melodies. The impression weakens when the sequence is played slowly (longer time between note onsets). The impression increases when the pitch separation is greater. Huron reports limit values for when events in a sequence will separate or fuse [8].

3. Quantizing the MIDI events

A MIDI file contains events with information about note onsets and offsets – we need to construct notes from that information. The algorithm processes events in groups of notes approximately beginning at the same time. MIDI events are processed in their time stamp order. Notes having onsets separated by no more than 35 ms are assumed to onset at the same time and are treated as a chord (onset group).

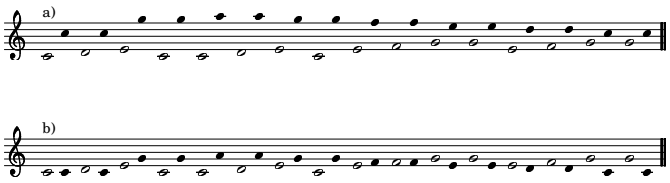

Figure 1. When two pitch sequences are interleaved, the two sequences will seem to fuse into a single stream if they are close together in pitch (b); otherwise they will seem to form two independent streams (a) [6].

The algorithm processes the groups of onsetting notes in time order. Note that only notes *onsetting* (approximately) simultaneously are member of the same group. Notes sustained from past onset groups are not included. Such notes are said to be busy at group time t_g (t_g is the start time of the first onsetting note in the group) when they have not ended before 80 ms after t_g. We shall use this later to determine if a voice is occupied by a note at any given group time.

When the input is a file, all notes are constructed at the beginning and afterwards the groups are determined and the separation begins. From real time input notes and groups are created on the fly.

4. The algorithm

Inspired by Temperley's well-formedness rules we will state some requirements the algorithm must assure:

- Each note must be assigned to exactly one voice
- A voice must not contain overlapping notes (must consist of temporally contiguous notes)

Voices are not allowed to contain notes sounding at the same time – we want the resulting voices to be entirely monophonic melodies. Furthermore the following will be preferred:

- Minimize leaps between notes in all voices
- Minimize the number of voices
- Minimize the number of rests within a voice (end a voice instead of introducing long rests)

In contrast to most of the existing algorithms voice crossing is not prohibited. Making voices cross from one onset group to the next is always more costly than making them continue the shortest path to the next notes (when the notes in each group have distinct pitches) and will thus not be a preferred choice. We shall later return to the issue of handling groups containing equal pitched notes.

The algorithm is implemented with a *voice configuration* unit generating valid (well formed) solutions and a *note assignment* unit calculating the preferred-ness of a solution using a parameterized cost function. There is a cost related to starting and ending a voice, inserting a rest (not assigning a note to an open voice) and there is a cost function of the leap size (for example the number of semitones). The hope is that the cheapest solution – given appropriate parameters – now corresponds to the correct one.

The optimal voice configuration and note assignment of a group g_i is highly dependent on the surrounding groups. We cannot consider the entire search space at once. Instead the separation happens iteratively with a small lookahead. When determining the best configuration and assignment (ca) for g_i we consider for example all possible configurations and assignments for g_i, g_{i+1}, and g_{i+2}. Each ca has a cost so the total cost of the lookahead is $ca(g_i)+ca(g_{i+1})+ca(g_{i+2})$. The ca for g_i leading to the cheapest solution when including the lookahead is finally applied.

In a given iteration, the well-formedness part of the algorithm makes sure that enough voices will be open (at least as many voices as there are notes in the group have to be open). Different configurations of incrementally opening more voices (up to the size of the group) and closing voices that are not needed are all evaluated.

The notes in g then have to be assigned to the open voices. Rests are added to open voices that have not been assigned any note (in the case there are more voices than notes to be assigned). All possibilities of assigning notes to the voices are evaluated. The cost of assigning a note to a voice is the pitch difference (or a function of the pitch difference) between that note and the previous note in the voice. The cost of assigning a rest is proportional to the duration of the group; the duration of a group is the time span to the following group.

The search algorithm uses a 'branch and bound' heuristic to prune the search space. When a solution is found, its cost is stored. Other possible solutions are not evaluated to the end if their cost-so-far already exceed the best solution's cost.

5. Evaluation methods

We have tested the algorithm mainly on the 15 two and 15 three part inventions by J. S. Bach (BWV 772-801) as well as the 48 fugues from the Well Tempered Clavier (BWV 846-893) from the same composer – the 'chewBach' data set also used by Chew and Wu in [5]. The inventions were obtained from *www.bachcentral.com* and the fugues from *www.musedata.org*. Voices are stored on different tracks in the midi file, providing the ground truth.

The separation algorithm assigns notes to voices. The question is now how well the voices correlate to the actual voices in the input data. Two evaluation procedures will be used: *soundness* and *completeness* as suggested by Kirlin [4].

Soundness is calculated by running through the notes in the ground truth streams, and for each adjacent pair determine if the notes were really assigned to the same voice.

Table 1. Evaluation in soundness and completeness

Exp.	Soundness (errors)	Completeness
1	94.25 % (4077)	66.70 %
2	94.34 % (4013)	69.47 %
3	94.79 % (3692)	73.76 %
4	97.21 % (1979)	67.83 %
5	97.44 % (1812)	71.31 %
6	97.58 % (1717)	71.58 %

A percentage will be returned. Soundness thus counts the percentage of 'right transitions' between notes.

Completeness on the other hand indicates how well the algorithm actually assigns notes from the same voice to the same stream. To give an example: consider two ground truth melodies consisting of 50 notes each. They do not cross. If a separation makes the streams cross exactly after the 25th note in each stream (let's assume that's possible), and otherwise makes no errors, soundness will be 97.96 % (96/98) and completeness 50%. Completeness is comparable to Chew and Wu's average voice consistency measure [5].

As in [5] we also end the separation where a ground truth voice becomes non-monophonic.

6. Experiments and results

To reduce calculation time for the experiments in this section, we adopted the policy of only opening up to one more stream than needed (instead of op to the number of the notes in the group) in the voice configuration unit. Thus if two streams are present, and two notes are to be assigned, we check the possibilities of opening none and opening one additional stream – not two more, which in theory would also be possible, since any note can start a stream. When more streams are present than notes to be assigned, we do not try opening new streams (but only closing).

To run the algorithm we need to specify cost values for opening and closing voices together with a cost value for rests (the value is per second). In the first experiment we set c_{open}, c_{close}, and c_{rest} to 50 and lookahead to 3 groups. The leap cost is simply the number of semitones. The result of Experiment 1 is shown in Table 1. A total of 70874 note transitions were examined in the soundness evaluation. The algorithm seems to perform better when evaluated for soundness than completeness. One factor influencing this is that the number of voices found necessary to separate the files (from 2 to 24) are higher than the actual number of voices in the inventions and fugues (2-5).

To compensate for this we introduce a small post processing step, trying to recombine ended voices with voices beginning later on. The assumption that closed streams will later reappear seems to be somewhat plausible in the chew-Bach data set, but it might not hold in all types of music. The concatenation of streams has a greater impact on completeness than on soundness, see Experiment 2 in the table. This approach is used in all experiments below.

A side effect of allowing voices to cross is the following: when two voices jump in the same direction, and the lower voice ends above the previous pitch of the upper voice, the sum of the leaps will be the same whether the voices cross or not. Adding a small non-linear term will make the algorithm prefer two intermediate-sized leaps over a large and a small, and thus prefer the shortest path for both voices. Experiment 3 was done with the leap cost function: $\text{cost}(d) = d + 0.05d^2$. This function will be used in the remaining experiments. Again the improvement primarily concerns completeness.

Other cost parameters might be a better choice. In Experiment 4 we have chosen the following parameters: $c_{open} = 100$, $c_{close} = 20$, and $c_{rest} = 50$. An improvement in soundness is introduced, but completeness drops. These values will be used in the remaining experiments.

Notes sharing both pitch and duration (group) are unseparable. We can only hope that they will be assigned to correct voices. There are 78 occurrences of this phenomenon in the data. We can expect our separation to handle these situations correctly in half of the occurrences. In 273 situations notes from different voices begin on the same pitch, and in 135 situations voices end on the same pitch – in both cases the durations are not equal. Voices coming together in unison are furthermore likely to cross, which result in errors primarily influencing the completeness measured. A way to avoid this could be by preferring a voice to continue close to the previous pitch it possessed. So when two voices jump into unison, they will try to part again without crossing. By adding 1/20 of the leap cost between this note and the second last note, we notice an improvement in completeness – see Experiment 5 in Table 1. This strategy is also used in the following experiment.

In Experiment 6 a new strategy is introduced: pattern matching. The idea is that previously heard sequences will be perceived in that way again. When a solution is considered, the solution is preferred, if the intervals between the last 5 notes in the voice(s) are already present somewhere in existing voices. We simply perform a search in all existing voices at the given time to see if the sequence previously occurred. Exact matching of pitches is used and no duration information is taken into account in this simple approach. On some files the effect is more positive than on other. By letting pattern matching compete with or overrule the pitch proximity rule in this way, new errors are naturally introduced. Overall only a small improvement is obtained in both soundness and completeness.

Changing the cost parameters in Experiment 4 caused a large impact on both soundness and completeness. With the danger of overfitting, we plan to do a stochastic search, optimizing soundness and completeness respectively in order to see how far we can go on this data set by using the

'right' parameters. Many pieces can be quite nicely separated with the tools at hand, but by using file specific parameters. Preliminary experiments show that when minimizing the total number of errors (optimizing soundness) using an evolutionary algorithm, we are not likely to get much further. Using the settings from Experiment 5, but with the values (c_{open}, c_{close}, c_{rest}) = (96.4, 9.2, 95.2) we get a soundness of 97.60% and a completeness of 71.13%. However, when optimizing completeness, it is possible to reach at least 79.68% by using the values (64.0, 98.5, 21.3) (soundness is then 94.62%).

We did a final attempt to improve the performance with respect to completeness. It can be noted that voices that cross tend to cross back again after a (short) while. We have run a few experiments preferring voices that cross each other an even number of times. Experiments showed it possible to gain some completeness (81.70%) while losing some soundness (94.73%).

6.1. Error analysis

The principle of pitch proximity is not enough to solve all problems in voice separation completely. However some of the mistakes committed by the algorithms making use of the principle are not clear to the perceiver. A common situation is when a voice ends, and one of the other voices present immediately jumps to a pitch close to the one the ending voice was playing. This situation makes the algorithm prefer to end the leaping voice and continue the ending. Pattern matching was expected to be able to handle some of these situations, by preferring that reoccurring material should not be divided into different streams. The effect of pattern matching on voice separation deserves a study on its own – we expect to do more research along these lines.

In many cases the principle of 'good continuation' could be a helping hand: prefer voices to have a logical continuation of what they recently were 'doing': prefer to continue in a way that reinforces any kind of musical 'idea' in pitch and/or rhythm. Since the percept of parallelism in music can be created by many compositional means, this heuristic will be hard to implement. However in cases where a musical idea is transferred from voice to voice, this approach would fail.

A systematic approach of locating errors and categorizing them could be useful but has not yet been pursued. This could give statistical evidence of which problems are most fruitful to solve.

7. Conclusion

A stream separation algorithm has been presented. The algorithm performs well on highly polyphonic music by Bach – most convincing when measuring the performance in soundness. Regarding completeness Chew and Wu's algorithm is clearly ahead (they report 88.98%, but provide no evaluation directly comparable to our measure of soundness).

However our method has other advantages: the real time processing ability including the dynamic opening and closing of streams. When connecting a MIDI instrument the input can almost immediately be viewed as separated voices. The border between single melody and pseudo-polyphony (the fusion of alternating notes) can be demonstrated.

In performed music, onset and offset of notes are most often not as strictly proportional to the notated values in the scores, as is the case with our Bach files rendered from scores. More intelligent means of quantizing the data are required – for example beat tracking could be useful when determining the onset groups (see [9]).

We are planning to use the separation algorithm for online analysis of performed music.

8. Acknowledgments

This research was supported by the Viennese Science and Technology Fund (WWTF, project CI010). The Austrian Research Institute for AI acknowledges basic financial support from the Austrian Federal Ministries of Education, Science and Culture and of Transport, Innovation and Technology.

References

[1] Albert S. Bregman. *Auditory Scene Analysis. The Perceptual Organization of Sound*. The MIT Press, Cambridge, Massachusetts, 1990.

[2] Emilios Cambouropoulos. From midi to traditional musical notation. In *In Proceedings of the AAAI'2000 Workshop on Articial Intelligence and Music*, Austin, TX, 2000. AAAI Press.

[3] Jürgen Kilian and Holger H. Hoos. Voice separation: A local optimisation approach. In Michael Fingerhut, editor, *Proceedings of the Third International Conference on Music Information Retrieval: ISMIR 2002*, pages 39–46, 2002.

[4] Phillip B. Kirlin and Paul E. Utgoff. VoiSe: Learning to segregate voices in explicit and implicit polyphony. In Joshua D. Reiss and Geraint A. Wiggins, editors, *Proceeding of the Sixth International Conference on Music Information Retrieval: ISMIR'05*, pages 552–557, 2005.

[5] Elaine Chew and Xiaodan Wu. Separating voices in polyphonic music: A contig mapping approach. In *Proceedings of the Second International Symposium on Computer Music Modeling and Retrieval (CMMR'04))*, volume 3310 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.

[6] David Temperley. *The Cognition of Basic Musical Structures*. MIT Press, Cambridge, MA, 2001.

[7] Fred Lerdahl and Ray Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, 1983.

[8] David B. Huron. Tone and voice: A derivation of the rules of voice-leading from perceptual principles. *Music Perception*, 19(1):1–64, 2001.

[9] Fabien Gouyon and Simon Dixon. A review of automatic rhythm description systems. *Computer Music Journal*, 29(1):34–54, 2005.