Voice Separation — A Local Optimisation Approach

Jürgen Kilian
Department of Computer Science
Darmstadt University of Technology
Wilhelminenstr. 7
64283 Darmstadt, Germany
+49 6151 166184
kilian@salieri.org

Holger H. Hoos*
Department of Computer Science
University of British Columbia
2366 Main Mall
Vancouver, BC, V6T 1Z4, Canada
+1 604 822 1964
hoos@cs.ubc.ca

ABSTRACT

Voice separation, along with tempo detection and quantisation, is one of the basic problems of computer-based transcription of music. An adequate separation of notes into different voices is crucial for obtaining readable and usable scores from performances of polyphonic music recorded on keyboard (or other polyphonic) instruments; for improving quantisation results within a transcription system; and in the context of music retrieval systems that primarily support monophonic queries. In this paper we propose a new voice separation algorithm based on a stochastic local search method. Different from many previous approaches, our algorithm allows chords in the individual voices; its behaviour is controlled by a small number of intuitive and musically motivated parameters; and it is fast enough to allow interactive optimisation of the result by adjusting the parameters in real-time. We demonstrate that compared to existing approaches, our new algorithm generates better solutions for a number of typical voice separation problems. We also show how by changing its parameters it is possible to create score output suitable for different needs, *e.g.*, piano-style *vs.* orchestral scores.

1. INTRODUCTION

For the transcription of low-level musical data into score notation, three major tasks need to be performed: tempo-detection, quantisation, and, in the case of non-monophonic input, a separation of the notes into different voices which can possibly contain chords. Voice separation is also essential for any MIR system based on monophonic techniques (see, *e.g.*, [5]). Finally, voice separation is of interest in the contexts of musical analysis and music cognition [2, 9].

In this work, our focus is on the creation of voice separations for various needs, particularly as arising in score generation and notation tasks as well as in the context of music information retrieval. Hence, our goal is not to find 'the correct' voice separation, which could hardly be defined without making restrictive assumptions about musical style, and would be difficult to capture accurately even in the presence of such assumptions. Rather, we pursue the more pragmatic goal of creating an adequate algorithm that is capable of finding a range of voice separations that can be seen as reasonable solutions in the context of different types of score notation (*e.g.*, only monophonic voices, only one voice including chords, multiple voices and chords; see Figure 1). The underlying idea is to allow a user by controlling a small number of intuitive parameters of the algorithm in real-time to interactively find a voice separation for a given piece that suits her specific but not necessarily explicitly known needs, and requires only minimal manual modifications in order to obtain a satisfying result.

*Also affiliated with the Department of Computer Science, Darmstadt University of Technology, Wilhelminenstr. 7, 64283 Darmstadt, Germany.

Our approach splits a complete piece into small slices of overlapping notes. These slices are processed iteratively, assigning each note of a slice to a voice. During this process, chords can be created by assigning multiple notes from a slice to the same voice. A randomised local search algorithm is used for finding assignments that minimise a parametric cost function which is used to assess the quality of partial voice separations. This cost function includes components that reflect the relationship between the notes within a slice (possible chord groupings) as well as between a slice and the partial voice assignment of previously processed slices (voice leading).

The task of assigning the notes of a slice to a number of existing voices becomes challenging if the number of notes differs from the number of voices available at that point. If there are more notes within the slice than voices, chords have to be introduced in one or more voices, or new voices need to be created. Our approach works with a maximum number of voices that can either be specified by the user or derived from the maximum number of notes overlapping at any point in time. (It may be noted that we preprocess the input data prior to performing voice separation in order to deal with certain types of inaccuracies and noise; this will be described in more detail in Section 5.2.) However, voices are only used when required and – depending on the parameter setting for the cost function – the result of the voice separation process will not always utilise the maximum number of voices.

An empirical evaluation of our algorithm shows that by using different (user accessible) settings for the parameters of the cost functions, a range of reasonable voice separations can be obtained for a given input. In many cases that are known to be problematic for other voice separation methods, such as static split-point separation or the approach described by Temperley [9], our algorithm achieves good results. In other cases, including randomly generated music, pieces with a high degree of pitch overlaps between voices, or pieces without any voice structure, our method encounters similar problems as other approaches.

The remainder of this paper is structured as follows. In the next section, we briefly describe revious work on voice separation and known algorithmic approaches. We then introduce some formal concepts that are required for a precise description of the voice separation problem addressed in this work and our algorithmic solution to this problem. The basic voice separation algorithm is presented in Section 4, and Section 5 describes important aspects of our current implementation. Finally, we present and discuss a range of sample results obtained from our new voice separation algorithm. We end with some conclusions and directions for future work.

2. EXISTING APPROACHES

Various approaches for finding good voice separations of a given input piece have been proposed in the literature and/or are used in practice. Most commercial sequencer software products implement the extremely simple split point separation technique, while more complex approaches appear to be only implemented in academic systems, such as Temperley and Sleator's Melisma Music Analyzer [10].

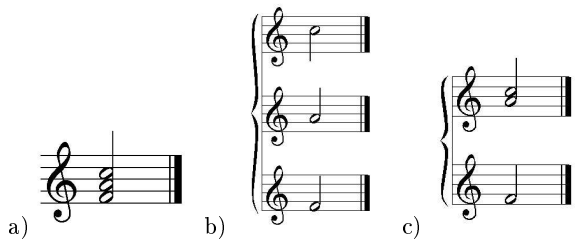

Figure 1: Different separations of three notes with equal onset times and durations: a) single chord, b) three voices, c) two voices with chord.

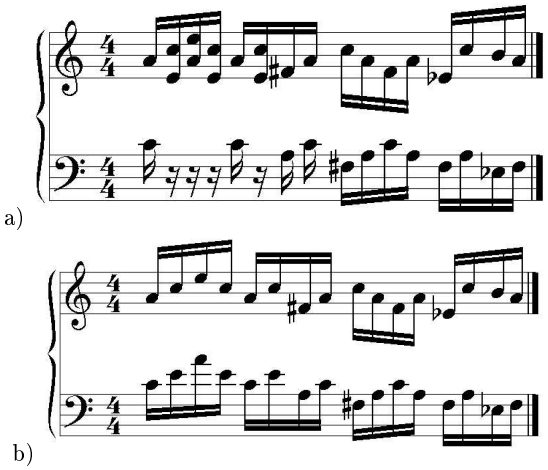

Figure 2: Voice separation for a piano style piece: a) with a fixed split point approach and b) with our new approach.

Split Point Separation

One of the simplest methods for voice separation is to split the range of all possible pitches into a number of disjoint intervals and to assign notes to voices depending on which pitch range they fall into. For two voices, this is achieved by fixing one pitch as a split point and by assigning all notes with equal or higher pitch to the first voice, while all notes with a lower pitch are assigned to the second voice. In general, for separating a piece into k voices, $k-1$ split points are used to define the k respective pitch ranges.

This approach is easy to implement and is used in most commercial sequencer software packages; however, it works correctly only for pieces in there are no overlaps in the ranges of different voices and will produce errors if this condition is not satisfied (see Figure 2a).

Rule Based Approaches

Because composers/arrangers are typically using voice-leading rules when composing or arranging musical pieces, it is possible to use the same rules for the inverse task of separating notes into separate voices. Studies of human perception of music show that if melodies are following such rules they are easier the perceive by human listeners [2]. Examples of voice leading rules include the following:

(1) prefer small intervals between succeeding notes

(2) keep range of a voice small

(3) use a small number of voices

(4) avoid crossings of voices

Because there are many such rules, including 'weak' rules which depend on the musical context, this approach will lead to many errors in pieces with more than two voices or with a changing number of voices. An implementation which mostly uses the 'nearest path rule' (1) is described by Cambouropoulos [1]. The nearest path rule implementation in FERMATA [6] showed that the results are mostly correct if the input data can be cleanly separated into a fixed number of continuous voices throughout the piece. If, however, the number of active voices changes or the ranges of two voices show major overlaps, this approach often produces erroneous separation. Improved performance can be achieved by segmenting the given piece into fragments during each of which the number of active voices remains constant, but unfortunately, finding such segmentations can be difficult.

As the number of rules increases, finding optimal voice separations with respect to a given rule system becomes a non-trivial task. Temperley and Sleator solved this problem by using a dynamic programming approach [9, 10]. Their Melisma Music Analyzer includes a contrapuntal analysis module which is based on a preference rule system. Different from our approach presented here, their focus is more on a correct analysis than on creating reasonable and flexible score-notation. One major difference to our approach is the fact that their system does not detect chords.

3. PRELIMINARIES

For the purposes of this work, we assume that the piece of music for which voice separation is to be performed is given in the form of a list of notes sorted by the time positions of their respective onset times. The ith note in this list is represented by a feature vector $m_i = (a_i, d_i, p_i)$, where a_i,d_i, and p_i denote the onset time, duration, and pitch of m_i. We also use $\mathrm{onset}(m_i)$, $\mathrm{duration}(m_i)$, and $\mathrm{pitch}(m_i)$ to refer to a_i, d_i, and p_i for a given note m_i. Likewise, we associate two integer values v_i and c_i with each note m_i to represent the voice and chord m_i is currently associated with; we also refer to these values as $\mathrm{voice}(m_i)$ and $\mathrm{chord}(m_i)$, respectively. Furthermore, we define $\mathrm{offset}(m_i) = a_i + d_i$, which effectively indicates the endpoint of note i.

Next, we define some relations between notes that are needed in the following:

$$\begin{aligned} m_i \leq m_j \quad &:\Leftrightarrow \quad \mathrm{onset}(m_i) \leq \mathrm{onset}(m_j) \\ m_i = m_j \quad &:\Leftrightarrow \quad \mathrm{onset}(m_i) = \mathrm{onset}(m_j) \end{aligned}$$

Furthermore, the function $\mathrm{overlap}(m_i, m_j)$ indicates whether notes m_i and m_j overlap in time: $\mathrm{overlap}(m_i, m_j)$ is true if and only if $m_i \leq m_j$ and $\mathrm{offset}(m_i) > \mathrm{onset}(m_j)$ or if $m_i > m_j$ and $\mathrm{offset}(m_j) > \mathrm{onset}(m_i)$.

Using these definitions, the input of our algorithm can be formally written as $M = (m_1, \ldots, m_l)$ such that $\forall i \in \{1, \ldots, l-1\} : m_i \leq m_{i+1}$, *i.e.*, as a list of notes sorted in ascending order of their respective onset times.

In the output of our voice separation algorithm, we allow two or more notes with the same onset time only to be assigned to the same chord if they form a chord:

$$\begin{aligned} m_i = m_j \quad \Rightarrow \quad & \mathrm{voice}(m_i) \neq \mathrm{voice}(m_j) \vee \\ & \mathrm{chord}(m_i) = \mathrm{chord}(m_j) \end{aligned}$$

For quantised input data, we restrict chords to consist only of notes with equal onset times. In the case of unquantised input data, onset times may be imprecise and hence we have to allow combining overlapping notes with different onset times into the same chord. In our implementation, we recognise and eliminate small overlaps and other inaccuracies during a preprocessing phase; this will be explained in more detail in Section 5. As a result of these constraints, each voice generated by our algorithm is a sequence of non-overlapping notes and chords.

Before assigning the notes of the input piece M into voices, we

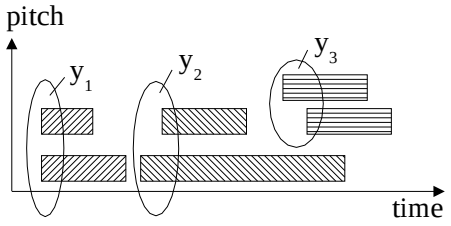

Figure 3: Example for partitioning a simple piece into slices.

will partition M into slices y_i of consecutive overlapping notes $(m_k, \ldots, m_{k+p})$ such that there is an overlap between any pair of notes within each slice and that between any two consecutive slices, y_i and y_{i+1} there are at least two notes that do not overlap. This implies that all notes from y_i except for the one with the smallest offset time may overlap with notes in y_{i+1} (see Figure 3 for an example).

Mathematically, the partitioning of M into slices can be modelled as follows. First, we define the set B of all indices of notes in M that become the first notes of the slices $y_1, \ldots, y_n$:

$$\begin{aligned} B \;:=\; & \{\, b_1, \ldots, b_n \mid \forall i \in \{1, \ldots, n\} : b_i \in \mathbb{N} \wedge \\ & b_1 = 1 \wedge b_n \leq l \wedge \\ & (\forall q, r \in \{b_i, \ldots, b_{i+1} - 1\} : \\ & \quad \mathrm{overlap}(m_q, m_r)) \wedge \\ & (\neg \exists s \geq b_{i+1} : \forall q \in \{b_i, \ldots, b_{i+1} - 1\} : \\ & \quad \mathrm{overlap}(m_q, m_s)\ \} \end{aligned}$$

Based on B, we can now define the set of all slices y_i:

$$Y \;:=\; \{y_1 \ldots y_n \mid y_i = (m_{b_i}, \ldots, m_{b_{i+1}-1}\}$$

The induces the following partitioning of M into slices y_i:

$$M = (\underbrace{m_1, \ldots, m_{b_2-1}}_{y_1}, \underbrace{m_{b_2}, \ldots, m_{b_3-1}}_{y_2}, \ldots, \underbrace{m_{b_n}, \ldots, m_l}_{y_n})$$

We denote a voice separation for a slice $y_i = (m_{b_i}, \ldots, m_{b_{i+1}-1})$ by $S(y_i) = (s_{i1}, \ldots, s_{ip})$ where the $s_{iq} = (\mathrm{voice}(m_{b_i+q-1}), \mathrm{chord}(m_{b_i+q-1}))$ represent the voice and chord that the qth note of slice y_i is assigned to under separation $S(y_i)$. As an abbreviation, will use $S_i := S(y_i)$. Furthermore, we use S_i^* to denote the set of all possible voice separations $S(y_i)$ for slice y_i. Any voice separation S for the complete input piece correspond to the combinations of voice separations for all slices y_i, *i.e.* $S = (S_1, \ldots, S_n)$, and the set of all possible voice separations of M is denoted S^*.

The number of different voice separations for a single slice y_i. *i.e.*, the size of the set S_i^*, depends on the number of notes in y_i, $|y_i|$, and on the maximum number of voices in the desired output of our separation algorithm, $nVoices$. More precisely, in the worst case, in which any subset of notes in y_i can be combined into a chord, there are at least $nVoices^{|y_i|}$ possible voice separations of slice y_i. Hence, in the worst case, the number of possible voice separations of a given input piece M is exponential in the number of notes in M.

This suggests that in order to find voice separations that are optimal with respect to a given set of criteria (which will be more precisely defined in the next section), the naive method of enumerating all possible separations and selecting the best of these can be prohibitively expensive, especially when the goal is to allow the user to interactively tune the parameters voice separation process in order to obtain a desired output. Consequently, our voice separation algorithm uses a substantially more efficient, heuristically guided process for iteratively constructing voice separations using a stochastic local search procedure for optimising the separation of individual slices.

```
procedure voiceSeparation(M, k)
  input:
    sorted list of notes M
    maximal number of voices k
  output:
    voice separation S
  segment M into slices y_1, ..., y_n
  S := ()
  for i := 1 to n
    S_i := separateSlice(y_i, S)
    S := S + S_i
    eliminate overlaps within voices of S
    and regularise chords where required
  end
end
```

Figure 4: Outline of our voice separation algorithm; for unquantised input data, this procedure is called after removing inaccuracies and noise in the input piece M.

4. OUR HEURISTIC ALGORITHM

The main idea underlying our algorithm is to construct a voice separation for the given input piece M from locally optimised separations for the slices of M. This local optimisation is based on a parametric cost function C that assesses the quality of the voice separation of a given slice, S_i, given the separations of all previous slices, *i.e.*, $(S_1, \ldots, S_{i-1})$. The definition of this cost function will be given below.

Given an input piece M and a maximal number of voices $nVoices$, our voice separation algorithm works as follows: After segmenting M into slices $y_1, \ldots, y_n$ (as described above), a cost-optimised voice separation for the first slice, y_1, is computed. Then, this voice separation is iteratively extended by cost-optimised separations for $y_2, \ldots, y_n$, resulting in a complete voice separation for M. However, particularly in the case of unquantified input data, this voice separation might contain chords with notes that slightly differ in their onset times or durations as well as overlapping notes within the same voice. Therefore, every time after the voice separation is extended by a slice separation, these situations are resolved by adjusting the durations or onset times. This guarantees that in the final result there are no overlaps between notes within any of the voices and all notes within any chord have the same onset time and duration.

In the following, we describe the cost function and the cost-optimising voice separation of slices in more detail.

4.1 The Cost Function

The cost function C used for assessing and optimising the quality of a voice separation S_i of a slice y_i, given separations $S_1, \ldots, S_{i-1}$ for all previous slices, is a weighted sum of terms that penalise individual, undesirable features:

$$\begin{aligned} C(S_i, S) \;=\; & p_{pitch} C_{pitch}(S_i, S) + p_{gap} C_{gap}(S_i, S) + \\ & p_{chord} C_{chord}(S_i) + p_{ovl} C_{ovl}(S_i, S) \end{aligned}$$

Here, S denotes the partial voice separation $(S_1, \ldots, S_{i-1})$. Intuitively, C_{pitch} and C_{gap} penalise large pitch intervals and gaps (rests) between successive notes in a voice, respectively; C_{chord} penalises chords with a large pitch interval between the highest and the lowest note, as well as irregular chords containing notes with different onset times or durations; and C_{ovl} penalises overlaps between successive notes in the same voice. By adjusting the weights of these terms, different trade-offs between these features can be achieved, leading to qualitatively different voice separations (chordal, single voices, *etc.*; see Figures 1 and 5). In the following, we describe in detail how the four penalty terms are calculated.

Pitch Distance Penalty C_{pitch}

The segregation of multiple melodic lines by human listeners de-

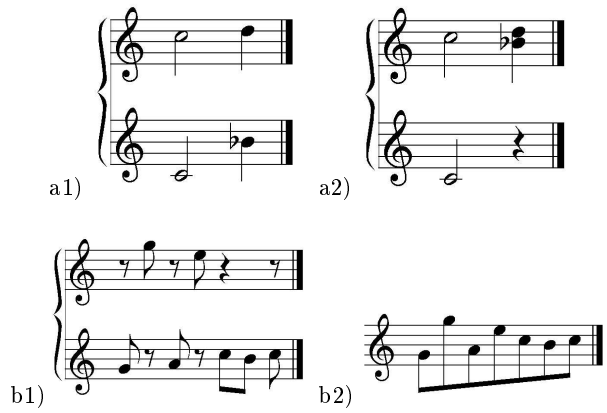

Figure 5: Different separations of non overlapping notes: a1) pChord>>pPitch, a2) pPitch>>pChord, b1) pPitch>>pGap, b2) pGap>>pPitch; Example b) is also discussed by R. O. Gjerdingen [3], p. 350, Figure 16.

pends very strongly on the frequency distribution and separation of the melodies [2]. Consequently, it makes sense to use similar features for in the context of automatic voice separation. The pitch distance penalty increases with the interval size between two succeeding notes in a voice. For the first note of a voice, a fixed penalty is imposed for starting a new voice. In some cases (melodies including short sequences of large intervals), a "lookback mechanism" can be advantageous by which the pitch at the end of an existing voice is calculated from the average pitch of the last n notes in the respective voice. This mechanism behaves somewhat similar to the approach of Gjerdingen [3] in which the motion-tracking system moves with some delay from a current pitch to the pitch of an incoming note. Only if the incoming note is long enough, the motion tracker reaches the exact pitch level of that note and stays there.

If a note m_j is assigned to a chord, we define the pitch-distance (interval size) between note m_j and a pitch p_l as the pitch-distance to the note of the chord which is closest in pitch to p_l. Therefore we define a function $cPitch(m_j, p_l)$ which returns $pitch(m_j)$ if m_j is a non-chord note and returns the pitch of the chord note which is closest to p_l otherwise. If no pitch lookback is used, the pitch of an existing voice v – when comparing to pitch p_l – is defined as $\mathrm{cPitch}(v, p_l)$ = $\mathrm{cPitch}(\mathrm{lOn}(v), p_l)$, where $\mathrm{lOn}(v)$ denotes the latest onset time in a voice v, *i.e.*, $\mathrm{lOn}(v)$ =onset(m_j) where j is the largest value such that $voice(m_j) = v$ in S. If pitch lookback is used, the pitch of voice v is calculated as shown in Figure 6; $\mathrm{prev}(m_j, p_l)$ denotes the note directly preceeding note m_j in the same voice as m_j and not assigned to the same chord as m_j. If the directly preceeding note is assigned to a chord, the chord note which is closest to pitch p_l returned instead *i.e.*, $prev(m_j, p_l) := m_k$ where k is the largest value such that $m_k < m_j$ with $\mathrm{voice}(m_k) = \mathrm{voice}(m_j)$ and $\mathrm{chord}(m_k) \neq \mathrm{chord}(m_j)$. The weighting of 0.8 for the current pitch and 0.2 for the previous pitch that is used in this calculatiopn was found empirically to give good results for the tested input data.

The pitch distance penalty C_{pitch} for a single voice can then be calculated as shown in Figure 7. Based on the C_{pitch} values for each voice v, the overall pitch distance penalty for a complete slice separation S_i can be calculated as shown in Figure 8.

Gap Distance Penalty C_{gap}

Studies have shown that melodies with few and short rests are perceived more easily as a coherent melodic line by a human listener than melodies with many long rests [2]. The structure of many melodic lines in Western music is consistent with this observation. Therefore, we impose a gap penalty if adding a note from the current

```
function cPitch(v, l, p_j)
    input:
        voice index v, lookback size l
        pitch p_j of note j
    output:
        average pitch p of voice v
        for comparison to p_j
    prevNote := prev(lOn(v), p_j))
    p := cPitch(lOn(v), p_j)
    i := 1
    while i ≤ l
        p := 0.8 · p + 0.2 · cPitch(prevNote, p_j)
        prevNote := prev(prevNote)
        i := i + 1
    end
    return(p)
end
```

Figure 6: Pitch calculation for voice v with $pitchlookback > 0$; pitch is measured in semitones.

```
function C_pitch(S_i, S, v)
    input:
        slice separation S_i
        separation S for previous slices
        voice index v
    output:
        pitch distance pvD
    prevNote := prev(v, pitch(m_j))
    pvD := 0;
    for all m_j ∈ S_i with voice(m_j) = v do
        pDist := |cPitch(prevNote, pitch(m_j))−
                  pitch(m_j))|/128
        pvD := pvD + (1 − pvD) · pDist
        if chord(prevNote) ≠ chord(m_j) then
            prevNote := m_j
        end
    end
    return(pvD)
end
```

Figure 7: Calculation of C_{pitch} for a single voice v in S_i; the division by 128 is based on the fact that when using MIDI data, 128 is the maximum pitch difference.

```
function C_pitch(S_i, S)
    input:
        slice separation S_i
        voice separation S for previous slices
    output:
        pitch distance penalty pD
    pD := 0;
    for all v used in S_i do
        pD := pD + (1 − pD) · C_pitch(S_i, S, v)
    end
    return(pd)
end
```

Figure 8: Calculation of pitch distance penalty C_{pitch} for slice separation S_i given separation S for previous slices.

```
function C_gap(S_i, S)
    input:
        slice separation S_i
        voice separation S for previous slices
    output:
        gap distance penalty gD
    gD := 0
    cNotes := 0
    for all voices v used in S_i
        m := earliest note in S_i with voice(m_j) = v;
        gD := gD + C_gap(m, v)
        cnotes := cnotes + 1
    end
    gD := gD/cnotes
    return(gD)
end
```

Figure 9: Calculation of gap distance penalty C_{gap} for slice separation S_i given separation S for previous slices.

slice to a voice introduces a rest; furthermore, the penalty increases with the duration of the rest. If the added note m is the first of the respective voice, the time difference between time position zero, *i.e.*, the onset time of the first note in M, and the onset time of m is penalised. Because all notes in a slice y_i are overlapping each other, gaps between notes within y_i cannot occur.

The gap distance penalty C_{gap} for a single note m_j and a voice v is defined as the length of the gap introduced in voice v by adding m_j divided by the maximal gap length introduced by adding m_j to any voice in S; this results in gap penalty values that are always between zero and one. Based on this measure, the overall gap distance penalty for S_i and S can be calculated as shown in Figure 9.

Chord Distance Penalty C_{chord}

Both, the limitations of human physiology in playing chords with very large ranges, *i.e.*, pitch differences between the highest and lowest note, as well as compositional practice suggest that when combining notes into chords, chords with small ranges should be preferred over chords with large ranges. Furthermore, in most cases, we would expect all notes belonging to the same chord to have identical or very similar onset times and durations. (Note that we allow the grouping of notes of with different onset times into the same chord only for unquantised input data.) Hence, we use a chord distance penalty that increases with the range of a chord, with the differences in durations of its notes, and with the distance between the respective onset times (in the case of unquantised data).

Based on these considerations, the following penalty terms are used as components of the overall chord distance penalty for a given slice separation S_i:

The range penalty $pRange$ for a chord c is defined as $pRange(c) = \min\{\text{range}(c)/24, 1\}$, where $\text{range}(c)$ is the pitch difference between the lowest and the highest note in c (measured in semitones). Note that according to this definition, the range penalty is always a value between zero and one, and all chords with a range of two octaves or more receive the same maximum penalty value of one.

Analogous to the range penalty, the duration penalty $pDuration$ for a chord c depends on the relation between shortest and longest note in c; it is defined as $pDuration(c) = 1 - sdur(c)/ldur(c)$, where $sdur(c)$ and $ldur(c)$ are the durations of the shortest and longest note in c, respectively. Note that according to this definition, the duration penalty is always a value between zero and one, and a range penalty of zero is obtained if and only if all notes of chord c have equal duration.

Finally, the onset time penalty pOn of a given chord c is defined as $pOn(c) = (lOn(c) - eOn(c))/lDur(c)$, where $lOn(c)$ and $eOn(c)$ are the onset times of the latest and the earliest note in c (with respect to their respective onset times), while $lDur(c)$ is the duration of the longest note in c. Note that the onset time penalty for chords in which all notes have equal onset times is zero; furthermore, pOn values can never be larger than one, because non-overlapping notes cannot be part of the same slice and hence will never be combined into the same chord.

```
function C_chord(S_i)
    input:
        slice separation S_i
    output:
        chord distance penalty cD
    cD := 0;
    for all chords c in S_i do
        p := pDuration(c)+
             (1 − pDuration(c)) · pRange(c)
        p := p + (1 − p) · pOn(c)
        cD := cD + (1 − cD) · p
    end
    return(cD)
end
```

Figure 10: Calculation of chord distance penalty C_{chord} for slice separation S_i.

Based on these three penalty terms for individual chords, the overall chord distance penalty for a complete slice separation S_i is calculated as shown in Figure 10. This particular way of combining the penalty terms is chosen to ensure that if one of the terms is large, the overall chord penalty is large as well; note that by using a (weighted) arithmetic average of the three penalty terms, this property cannot be guaranteed.

Overlap Distance Penalty C_{ovl}

Although notes within a voice generally should not overlap, depending on the instrument and style of music, there are cases in which the notes of a single melodic line are played with substantial overlaps that cannot be removed reliably by preprocessing, as illustrated in the of fingered pedal example shown in Figure 11. Therefore, we allow overlapping notes to be assigned to the same voice without combining them into a chord, but impose a penalty that increases with the amount of overlap. (Note that such overlaps are ultimately eliminated in our algorithm by shortening the duration of the earlier note.)

Consequently, we define the overlap distance penalty for two successive notes m_j and m_k within the same voice as $C_{ovl}(m_j, m_k) = 1 - (\text{onset}(m_k) - \text{onset}(m_j))/\text{duration}(m_j)$ if m_j and m_k overlap, *i.e.*, if $\text{overlap}(m_j, m_k)$, and $C_{ovl}(m_j, m_k) = 0$ otherwise. According to this definition, the overlap distance penalty between notes $C_{ovl}(m_j, m_k)$ is always a value between zero and one. The overlap distance penalty for a single voice v used in a slice separation S_i and the overall overlap distance for a slice separation S_i are then calculated as shown in Figures 12 and 13.

4.2 Cost-Optimised Slice Separation

Based on the cost function C defined above and given a separation S of slices $y_1, \ldots, y_{i-1}$, we use a stochastic local search approach for finding a cost-optimised voice separation S_i for slice y_i: Starting with an initial separation $S_i := S_i^0$, a series of randomised iterative improvement steps is performed during each of which one note is reassigned to a different voice. Whenever such step results in an assignment with lower cost than the best assignment seen so far, this assignment and its cost are memorised. This search process is terminated when no such improvement has been achieved for a fixed number θ of steps. In the current implementation, we use $\theta = 3 \cdot |y_i| \cdot nVoices$. Figure 14 gives a pseudo-code specification of this randomised iterative improvement procedure.

An intial separation S_i^0 for the given slice S_i is obtained by assigning all notes of y_i to the first voice. During this process, notes with equal

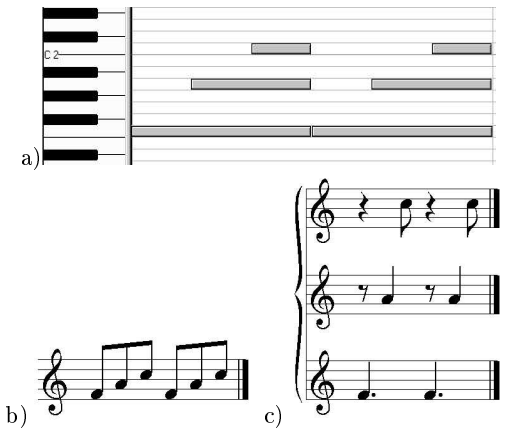

Figure 11: Different separations of three overlapping notes: a) input data b) all overlaps removed to form single voice, c) split into three voices.

function $C_{ovl}(S_i, S, v)$
  input:
    slice separation S_i
    voice separation S for previous slices
    voice-ID v
  output:
    overlap distance penalty ovD
  $prevNote :=$ lOn(v)
  $ovD := 0$
  for all m_j in y_i with voice$(m_j) = v$ **do**
    $oDist := C_{ovl}(prevNote, m_j)$
    $ovD := ovD + (1 - ovD) \cdot oDist$
    if chord$(m_j) \neq$ chord$(prevNote)$ **then**
      $prevNote := m_j$
    end
  end
  return(ovD)
end

Figure 12: Calculation of overlap distance penalty for single voice.

function $C_{ovl}(S_i, S)$
  input:
    slice separation S_i
    voice separation S for previous slices
  output:
    overlap distance penalty oD
  $oD := 0$
  for all v used in S_i
    $oDist := C_{ovl}(S_i, S, v)$
    $oD := oD + (1 - oD) \cdot oDist$
  end
  return(oD)
end

Figure 13: Calculation of overlap distance penalty C_{ovl} for slice separation S_i given separation S for previous slices.

function separateSlice(y_i, S)
  input:
    slice y_i
    voice separation S for previous slices
  output:
    optimised selection S_i^{opt}
  obtain S_i by setting all notes of y_i to voice 0
  and combining all notes with equal onset times
  into chords
  $S_i^{opt} := S_i$
  $noImpr := 0$
  while $noImpr < |y_i| * nVoices * 3$
    with probability 0.8 **do**
      $S_i :=$ neighbour S_i' of S_i with
        minimal cost $C(S_i', S)$
    otherwise
      $S_i :=$ randomly selected neighbour of S_i
    end
    if $C(S_i, S) < C(S_i^{opt}, S)$ **then**
      $S_i^{opt} := S_i$
      $noImpr := 0$
    else
      $noImpr := noImpr + 1$
    end
  end
  return(S_i^{opt})
end

Figure 14: Randomised iterative improvement algorithm for finding a cost-optimised separation for a single slice y_i.

onset times are combined into chords. Another natural choice for the initial separation is to distribute all notes in y_i into voices such that overlaps and chords are avoided as far as possible. Empirical tests (not reported here) suggested that the former initialisation method leads to better results than the latter approach.

Subsequently, in each local search step we move from the current separation of y_i to a neighbouring separation; two separations S_i and S_i' are neighbours if and only if S_i and S_i' are both valid separations of y_i that differ in the voice and/or chord assignment of exactly one note in y_i. A separation is valid if and only if any notes with identical onset times that are assigned to the same voice are also combined into a chord (see Equation 3).

The selection of the actual search step to be performed is based on a randomised greedy choice: With a certain probability (in our implementation, we used a value 0.8 for which we obtained good empirical results), the neighbouring separation with minimal cost is selected, otherwise, a neighbour of the current assignment is selected uniformly at random. The randomisation prevents the search process from getting stuck in local minima of the cost function C.

While there is no theoretical guarantee that this randomised iterative improvement algorithm will find the globally optimal separation for the given slice, it finds optimal or close-to-optimal separations very efficiently in practice. Similar stochastic local search strategies have been very successfully applied to many prominent combinatorial problems (see [4]).

5. IMPLEMENTATION

Our voice separation algorithm is implemented in the current version of midi2gmn, a larger programme for converting MIDI files into GUIDO Music Notation. All parameters for the voice separation can be specified in an initialisation file (fermata.ini). The code is written in ANSI C++ and has been successfully compiled and tested under Windows, Linux and Mac OS.

The input data can be quantised or unquantised low-level musical data. Unquantised data (where the tempo might also be unknown) requires preprocessing to remove inaccuracies and noise that could

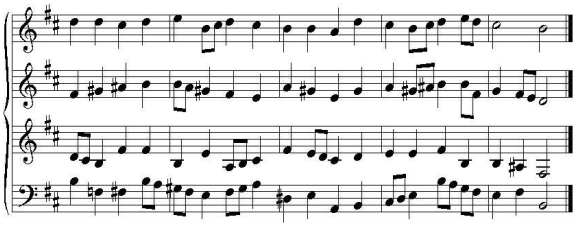

Figure 15: Ending of the choral 'Mitten wir im Leben sind' by J.S. Bach, BWV 383, separated as a four voice score.

have detrimental effects on the performance of the voice separation algorithm.

Our preprocessing removes small overlaps between notes as may arise, *e.g.*, when playing legato passages on a keyboard instrument. We consider an overlap small, if the time difference between the offset of the earlier note and the onset time of the later note is small compared to the durations of the two notes. Such overlaps are eliminated by shortening the duration of the earlier note

Furthermore, minor differences in the onset times of two notes that result from imprecise playing can be resolved by replacing both onset times with their average if the durational overlap of the two notes is large compared to the onset time distance and if the onset time distance is very small.* Notes with a longer duration or greater intensity can have a 'higher influence' on the calculation of the resulting average onset time. This onset time correction forces our voice separation algorithm to combine the respective notes into chords if they get assigned to the same voice and hence facilitates the recognition of chords.

Our implementation uses MIDI files as input data; these can be obtained from notation software, by recording a human performance using a MIDI sequencer, from a wave to MIDI converter, or as the result of converting musical data from other formats.

The four penalty parameters and the pitch lookback parameter can be defined by the user in an initialisation file. Parameters for which no value is specified are set to default values predefined in our implementation. The maximum number of voices to be used in the final separation of the given piece can also be specified in the initialisation file. If this parameter is not specified by the user, the maximum number of voices is set to the maximum number of overlapping notes at any time position of the input piece.

6. RESULTS AND DISCUSSION

We tested our approach on different types of music: most of the inventions by J.S. Bach, some chorals by the same composer, a waltz by F. Chopin, parts of 'Mikrokosmos' by B. Bartok, and several other pieces. As input data we used quantised MIDI files with all events merged into a single track.

With the correct parameter settings, the tested Bach chorals and inventions were separated almost entirely correctly (see Figures 15 and 16). Only in a few cases where the voices nearly meet at the same pitch, sometimes localised errors occurred. With different parameter settings, it was possible to separate the chorals into single voices, into a two staff piano score, or to collect them as chords in one voice.

Figure 17 shows a correct separation of a part of a waltz by Chopin obtained from our voice separation algorithm. When using higher chord penalties and lower overlap penalties, however, the same input data is (incorrectly) separated as shown in Figure 18.

*It has been shown experimentally that onset times with a distance of less than approx. 40ms are perceived as simultaneous by human listeners (see, *e.g.*, [2]).

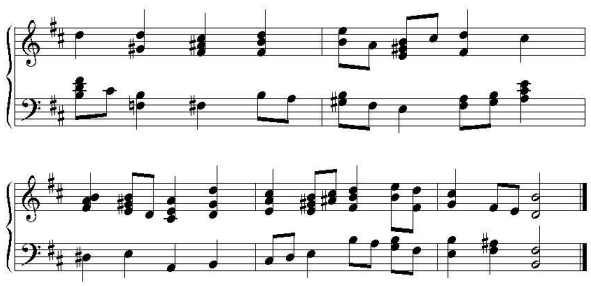

Figure 16: Ending of the choral 'Mitten wir im Leben sind' by J.S. Bach, BWV 383, separated as a piano-style score.

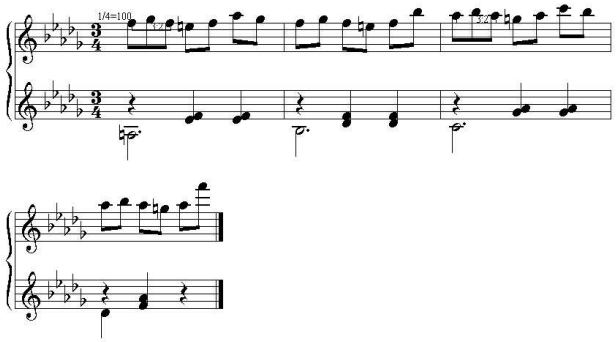

Figure 17: Chopin, Valse, Opus 64 Nr. 1, mm. 101-104.

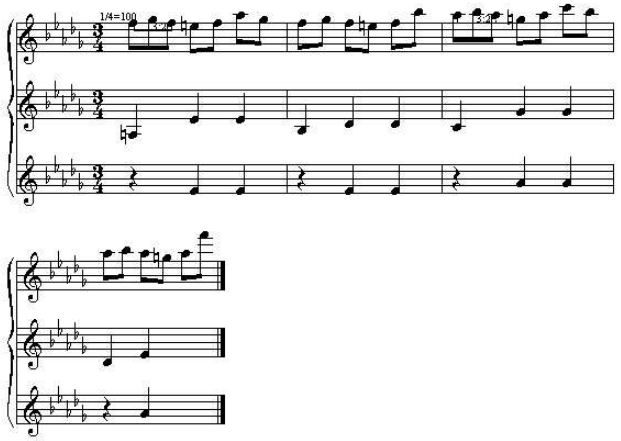

Figure 18: Chopin, Valse, Opus 64 Nr. 1, mm. 101-104, incorrect separation.

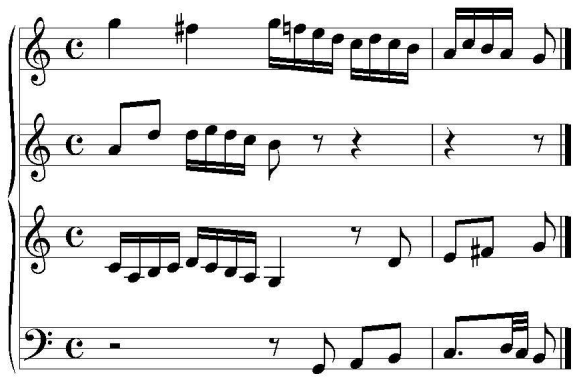

Figure 19: J.S. Bach, Well-Tempered Clavier, Book I, Fugue No. 1 in C Major, mm. 10-11.

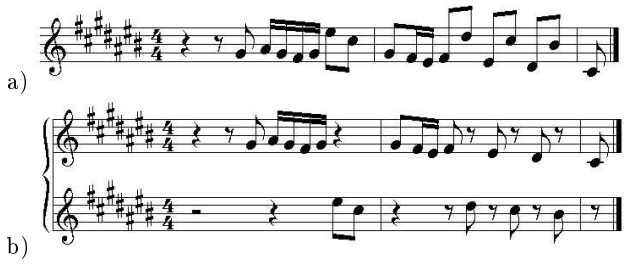

Figure 20: J.S. Bach, Well-Tempered Clavier, Book I, Fugue No. 3 in C# Major, mm. 1-3. a) Correct separation with $pGap > pPitch$. b) Incorrect separation with $pPitch > pGap$.

Situations in which a voice continues with a large interval step after a rest can lead to incorrect voice separations. In the example of Figure 19, the alto and tenor voices pause in the middle of measure 10. In the original score, the alto voice continues at the end of measure 10 and the tenor voice has still tacet. Because the first note of the continuing motive (d1) has a smaller pitch distance and a smaller gap distance to the tenor voice (g0) than to the alto voice (b1) and because there are only three notes to separate into four voices, the algorithm assigns this fragment incorrectly to the tenor voice.

The same example is also discussed by Temperley [9], whose approach encountered the same problems. We could not improve this result by changing parameter settings. It seems that when only considering the notes, without any knowledge about the composer and style of the piece, there is no reason why another separation should be prefered. This case shows that there exist scores or compositions where at some points the intention of the composer differs from the results obtained by applying standard voice-leading rules or cost functions. In more complex pieces (*e.g.*, Mikrokosmos 153 by Bartok), the same effect occurs at some positions. In another example (Figure 20), we could avoid the problems which Temperley discusses in his book by using our algorithm with appropriately chosen parameter settings.

7. CONCLUSIONS AND FUTURE WORK

We presented an approach for voice separation on low-level musical data using stochastic local search for calculating an optimal voice structure. Different from most other existing approaches which are restricted to separating a piece into multiple monophonic lines, our system can detect chords. It is also capable of producing different types of voice separations, based on the settings of a small set of parameters that are accessible to the user. These parameters control the relative influence of various criteria for good voice separations in an intuitive way.

Our algorithm can be applied to quantised input data as well as to unquantised input data with our without tempo information. For unquantised input data, we apply preprocessing in order to eliminate inaccurracies and noise prior to running our voice separation algorithm. It may be noted that correctly separating the voices within unquantised input data can increase the quality of a subsequent quantisation process.

We believe that the quality of the results obtained from our algorithm can be further improved by fine-tuning the cost function. For example, our latest tests indicate that using an exponential function for penalising the chord range may give better voice separations.

We are currently investigating possibilities for connecting our current implementation (midi2gmn) to the online GUIDO NoteServer [8] to provide an online MIDI-to-Score service. In this context, we are studying further optimisations of our implementation with respect to its run-time.

In the future, we plan to develop a graphical interface for our voice separation programme that allows the user to change the penalty parameters and see the corresponding results in real-time. Furthermore, in some cases it would be beneficial to support the use of different parameter settings for different parts of a given piece. This could be accomodated by allowing the user to select individual fragments of a given piece and to perform voice separation only for the selected fragments.

8. REFERENCES

[1] E. Cambouropoulos. From MIDI to Traditional Musical Notation; In *Proc. of the AAAI Workshop on Artificial Intelligence and Music*: Towards Formal Models for Composition, Performance and Analysis. Austin (TX), USA, 2000.

[2] A.S. Bregman. Auditory Scene Analysis. The MIT Press, Cambridge (MA), USA, 1990.

[3] R.O. Gjerdingen. Apparent motion in music? Music Perception, Vol. 11, pp. 335–370, 1994

[4] H.H. Hoos. Stochastic Local Search – Methods, Models, Applications. Infix Verlag, Sankt Augustin, Germany, 1999.

[5] H.H. Hoos, K. Renz, and M. Görg. GUIDO/MIR — an Experimental Musical Information Retrieval System based on GUIDO Music Notation. In *Proc. of the 2nd International Symposium on Music Information Retrieval (ISMIR 2001)*, Indiana University, Bloomington (IN), USA, 2001.

[6] J. Kilian. FERMATA – Flexible Quantisierung von Musikstücken im MIDI-Dateiformat. M.Sc. thesis (in German), Darmstadt University of Technology, Germany, 1996.

[7] S. L. McCabe and M. J. Denham. A model of auditory streaming. Journal of the Acoustical Society of America, Vol. 101(3), pp. 1611–21, 1997.

[8] K. Renz and H. H. Hoos. A Web-based Approach to Music Notation Using GUIDO. In *Proc. of the International Computer Music Conference 1998*, pp. 455–458, ICMA, San Francisco (CA), USA, 1998. See also: http://www.noteserver.org.

[9] D. Temperley. The Cognition of Basic Musical Structures. The MIT Press, Cambridge (MA), USA, 2001.

[10] D. Temperley and Daniel Sleator. The Melisma Music Analyzer. http://www.links.cs.cmu.edu/music-analysis.