Horizontal and Vertical Integration/Segregation in Auditory Streaming: A Voice Separation Algorithm for Symbolic Musical Data

Ioannis Karydis[*], Alexandros Nanopoulos[*], Apostolos Papadopoulos[*], Emilios Cambouropoulos[†] and Yannis Manolopoulos[*]

[*] Department of Computer Science, Aristotle University of Thessaloniki, Greece, {karydis,ananopou,apapadop,manolopo}@csd.auth.gr

[†] Department of Music Studies, Aristotle University of Thessaloniki, Greece, emilios@mus.auth.gr

Abstract — **Listeners are thought to be capable of perceiving multiple voices in music. Adopting a perceptual view of musical 'voice' that corresponds to the notion of auditory stream, a computational model is developed that splits a musical score (symbolic musical data) into different voices. A single 'voice' may consist of more than one synchronous notes that are perceived as belonging to the same auditory stream; in this sense, the proposed algorithm, may separate a given musical work into fewer voices than the maximum number of notes in the greatest chord (e.g. a piece consisting of four or more concurrent notes may be separated simply into melody and accompaniment). This is paramount, not only in the study of auditory streaming per se, but also for developing MIR systems that enable pattern recognition and extraction within musically pertinent 'voices' (e.g. melodic lines). The algorithm is tested qualitatively and quantitatively against a small dataset that acts as groundtruth.**

Keywords: **Voice separation, auditory streaming, melodic extraction.**

I. INTRODUCTION

It appears that the term 'voice' has different meanings for different research fields (traditional musicology, music cognition, computational musicology). Recently, there have been a number of attempts (e.g. Temperley, 2001; Cambouropoulos 2000; Kilian & Hoos 2002; Szeto and Wong, 2003; Chew & Wu 2004; Kirlin & Utgoff 2005; Madsen and Widmer 2006) to model computationally the segregation of polyphonic music into separate voices. Much of this research is influenced by empirical studies in music perception (e.g. Bregman, 1990; Deutsch, 1999; Huron 2001), as well as by more traditional musicological concepts such as melody, counterpoint, voice-leading and so on.

Before presenting various aspects of voice separation in terms of perception and computational modelling, it is important to clarify what is meant by the term 'voice'. A detailed discussion in presented by Cambouropoulos (2006). In this paper, a single musical example will be given that presents three different meanings of the term voice.

In Fig. 1 three musical examples are given that share the same harmonic structure. In these examples, we can see three different ways in which 'voice' may be understood: literally 'voice' (human voice or monophonic instrument)[1], harmonic 'voice' relating to harmonic content and evolution, and perceptual 'voice' relating to auditory streaming. In terms of literal voice, the first instance (Fig.1a) is purely monophonic (one voice), the second (Fig.1b) consists of two voices, and the third (Fig.1c) is made up of three voices. In terms of harmonic voice, all instances can be analysed as comprising of three voices that relate to the implied harmony (progression of triadic chords I-IV-V-I). In terms of perceptual voice, each instance is perceived as a single auditory stream (e.g., in Fig.1c a listener does not perceive three independent voices but a single unified progression of chords).

This paper is partially funded by a national GSRT PAVET project

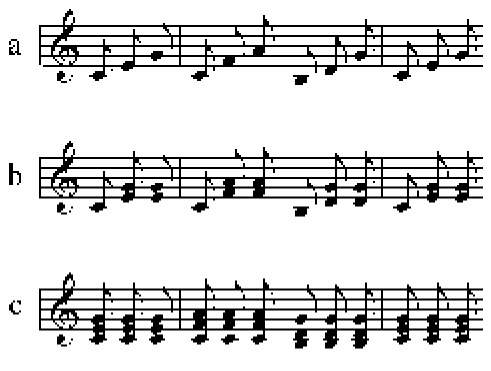


Figure 1 Number of voices: in terms of literal voices we have in the three examples one, two and three voices respectively; in terms of harmonic voices all examples can be understood as comprising of three voices (triadic harmony); in terms of perceptual voices/streams each example is perceived as a single auditory stream (harmonic accompaniment).

The standard understanding of the term voice is that voice is a *mono*phonic sequence of successive non-overlapping musical tones; a single voice is thought not to contain multi-tone sonorities. However, if 'voice' is seen in the light of auditory streaming then it becomes clear that the standard meaning is not sufficient. It is possible that a single monophonic sequence may be perceived as

[1] The literal meaning of the term 'voice' may be broadened, making it applicable to music produced by a single 'polyphonic' instrument (such as the piano, celesta, guitar etc.) in cases where the music consists of a relatively fixed number of individual musical lines (e.g. 3- or 4-part fugues or other 4-part works for keyboard instruments). Terms that are synonymous to 'voice' in this sense are 'part' and 'line' – in the case of polyphonic music the term 'contrapuntal voice' is often used.

more than one voice/stream (e.g., pseudopolyphony or implied polyphony) or that a passage containing concurrent notes may be perceived as a single perceptual entity (e.g., homophonic passages as in Fig.1c). Auditory stream integration/segregation (in music) determines how successions of musical events are perceived as belonging to coherent sequences and, at the same time, segregated from other independent musical sequences. A number of general perceptual principles govern the way musical events are grouped together in musical streams (see section 3).

The perceptual view of voice adopted in this study, that allows multi-tone simultaneities in a single 'voice', is the most significant difference of the proposed model to the other existing models. In the examples of Fig. 1, all existing algorithms (see exception regarding Kilian and Hoos's algorithm in the next section), that are based on purely monophonic definitions of voice, would find two voices in the second example (Fig.1b) and three voices in the third example (Fig.1c). It is clear that such voices are not independent voices and do not have a life of their own; it makes more musical sense to consider the notes in each example as a single coherent whole (a unified harmonic sequence). The algorithm proposed in this paper determines that in all three examples we have a single 'voice'/stream.

In this paper, initially, a number of recent voice separation algorithms are briefly described and their main differences to the current proposal are highlighted. Then, the fundamental auditory streaming principles that form the basis of the proposed model are presented. The description of the algorithm follows. Finally, the way the algorithm has been evaluated and the results of the application of the algorithm on four different musical works are presented.

II. COMPUTATIONAL MODELS OF VOICE SEPARATION

'Voice' separation algorithms are very useful in computational implementations as they allow pre-processing of musical data opening thus the way for more efficient and higher quality analytic results. In domains such as music information retrieval or automated musical analysis, having sophisticated models that can identify multiple melodic voices and/or 'voices' consisting of multi-note sonorities can enable more sophisticated processing within the voices (rather than across voices). For instance, if one wants to identify musical works that contain a certain melodic pattern, this pattern should be found not spread across different parts (perceptually implausible) neither in voices that are not perceptually independent (e.g. internal parts in a homophonic work) but within voices that are heard as having a life of their own.

Recently, there have been a number of attempts to model computationally the segregation of polyphonic music into separate 'voices' (e.g. Marsden, 1992; Temperley, 2001; Cambouropoulos 2000; Kilian & Hoos 2002; Szeto and Wong, 2003; Chew & Wu 2004; Kirlin & Utgoff 2005; Madsen and Widmer 2006). These models differ in many ways but share two fundamental assumptions:

Firstly, 'voice' is taken to mean a monophonic sequence of successive non-overlapping musical tones (exception is the model by Kilian and Hoos which will be discussed further below)

Secondly, the underlying perceptual principles that organise tones in voices are the principles of temporal and pitch proximity (cf. Huron's Temporal Continuity and Pitch Proximity principles).

In essence, these models attempt to determine a minimal number of lines/voices such that each line consists of successions of tones that are maximally proximal in the temporal and pitch dimensions. A distance metric (primarily in regards to pitch and time proximity) is established between each pair of tones within a certain time window, and then an optimisation process attempts to find a solution that minimises the distances within each voice keeping the number of voices to a minimum (usually equal to the maximum number of notes in the largest chord). These models assume that a voice is a succession of individual non-overlapping tones (sharing of tones between voices or crossing of voices is forbidden or discouraged).

For instance, Temperley (2001) proposes a number of preference rules that suggest large leaps (Pitch Proximity Rule) and rests (White Square Rule) should be avoided in streams, the number of streams should be minimised (New Stream Rule), common tones shared between voices should be avoided (Collision Rule) and the top voice should be minimally fragmented (Top Voice Rule) – the maximum number of voices and weight of each rule is user-defined. Cambouropoulos (2000) assumes that tones within streams should be maximally proximal in terms of pitch and time, that the number of voices should be kept to a minimum and that voices should not cross – the maximum number of streams is equal to the number of notes in the largest chord. Chew and Wu (2004) base their algorithm on the assumption that tones in the same voice should be contiguous and proximal in pitch, and that voice-crossing should be avoided – the maximum number of voices is equal to the number of notes in the largest chord. Szeto and Wong (2003) model stream segregation as a clustering problem based on the assumption that a stream is essentially a cluster since it is a group of events sharing similar pitch and time attributes (i.e. proximal in the temporal and pitch dimensions) – the algorithm determines automatically the number of streams/clusters. All of these voice separation algorithms assume that a voice is a monophonic succession of tones.

The voice separation model by Kilian and Hoos (2002) differs from the above models in that it allows entire chords to be assigned to a single voice, i.e. more than one synchronous notes may be considered as belonging to one stream. The model is based on a dynamic programming approach. It partitions a piece into slices; contiguous slices contain at least two non-overlapping notes. A cost function is calculated by summing penalty values for features that promote segregation such as large pitch intervals, rests/gaps, and note overlap between successive notes, and large pitch intervals and onset asynchrony within chords. Within each slice the notes are separated into streams by minimising this cost function. The user can adjust the penalty values in order to give different prominence values to the various segregation features leading thus to a different separation of voices. The maximum number of voices is user-defined or defined automatically by the number of notes in the largest chord.

The aim of the algorithm is to find ‘a range of voice separations that can be seen as reasonable solutions in the context of different types of score notation’ (Kilian and Hoos, 2002, p.39). The pragmatic goal of the algorithm is the derivation of reasonable score notation - not perceptually meaningful voices. The algorithm is based on perceptual principles, but the results are not necessarily perceptually valid (e.g., a 4-part homophonic piece may be ‘forced’ to split into two musical staves that do not correspond to perceptually pertinent streams). The algorithm does not discover automatically the number of independent musical ‘voices’ in a given excerpt; if the user has not manually defined the maximum number of voices, the algorithm automatically sets the maximum number equal to the maximum number of co-sounding notes – in this case the algorithm becomes similar to all other algorithms presented above.

Kilian and Hoos’s model allows multiple synchronous or overlapping tones in a single stream based on pitch and temporal proximity. However, there are two problems with the way this idea is integrated in the model. Firstly, simple pitch and temporal proximity are not sufficient for perceptually pertinent ‘vertical’ integration. For instance, Kilian and Hoos’s model can separate a 4-part fugue into two ‘streams’ based on temporal and pitch proximity, but these two ‘streams’ are not perceptual streams but rather a convenient way to divide notes into two staves. In perceptual terms, tones merge when they have ‘same’ onsets and durations (see next section); overlapping tones with different onsets and durations do not merge (there exist, however, special cases where this happens – not discussed in this paper). Secondly, synchronous notes that are separated by a small pitch interval are not in general more likely to be fused than tones further apart. For instance, tones an octave apart are strongly fused whereas tones a 2nd apart are less likely to be fused (see next section). The perceptual factor of tonal fusion is not taken into account by the model.

Kilian and Hoos’s model is pioneering in the sense that multi-note sonorities within single voices are allowed; their model, however, is apt to give results that are erroneous in terms of auditory stream segregation as this is not the goal of the algorithm.

III. PERCEPTUAL PRINCIPLES FOR VOICE SEPARATION

In this section, fundamental principles of perceptual organisation of musical sounds into streams will be presented that form the basis of the current computational model (next section). Huron (2001) provides an excellent survey of relevant research and presents a set of 10 principles that cover all major aspects of stream integration/segregation; we will use a number of these principles as the starting-point of our exploration. This work by Huron has formed the theoretical and empirical starting point for many of the computational models mentioned above.

Since we have assumed that a voice may contain multi-tone sonorities, it is important to distinguish between principles that are primarily responsible for vertical integration and ones for horizontal integration. Below we will look into the way tones are organised ‘vertically’ and ‘horizontally’ into coherent ‘wholes’

A. Vertical Integration

Bregman (1990) explores in depth processes relating to the perceptual integration/segregation of simultaneous auditory components. In this paper, we will focus only on two aspects of such processes that relate to two principles presented by Huron (2001), namely the principles of Onset Synchrony and Tonal Fusion.

Sounds that are coordinated and evolve synchronously in time tend to be perceived as components of a single auditory event. ‘Concurrent tones are much more apt to be interpreted by the auditory system as constituents of a single complex sound event when the tones are temporally aligned.’ (Huron, 2001, p.39). Concurrent tones that start, evolve and finish together tend to be grouped together into a single sonority.

In practical terms, we could state that notes that start concurrently and have same duration tend to be merged vertically into a single sonority.[2] This principle relates to Huron’s Onset Synchrony Principle[3] but it differs in a number of ways as discussed in Cambouropoulos 2006.

> *Synchronous Note Principle:* Notes with synchronous onsets and same inter-onset intervals IOIs (durations) tend to be merged into a single sonority.

A second important factor for vertical integration of tones, relates to the Principle of Tonal Fusion: The fusion between synchronous notes is strongest when notes are in unison, very strong when separated by an octave, strong when separated by a perfect fifth and progressively weaker when separated by other intervals.

> *Principle of Tonal Fusion:* The perceptual independence of concurrent tones is weakened when they are separated by intervals (in decreasing order: unisons, octaves, perfect fifths…) that promote tonal fusion (Huron, 2001, p.19).

This principle suggests that concurrent pitches are integrated depending on the degree of tonal fusion implied by interval type rather than mere pitch proximity; this principle appears to be (at least partially) in conflict with the pitch proximity principle that has been adopted for vertical integration in the computational model by Kilian and Hoos (2002).

Finally, according to the Pitch Co-modulation Principle: ‘The perceptual union of concurrent tones is encouraged when pitch motions are positively correlated.’ (Huron, 2001, p.31) The strongest manifestation of this principle is when notes move in parallel intervals (especially in octaves). The Pitch Co-modulation Principle can be seen as a special case of the Synchronous Note Principle in the sense that the integration of synchronised note progressions is reinforced when pitch progressions are positively correlated (e.g. moving in parallel octaves, fifths etc.). This principle has not yet been incorporated in the current version of the algorithm.

[2] For simplicity, in this study we consider notes as internally static events that are characterised by onset, pitch and duration (as represented in piano-roll notation).

[3] *Onset Synchrony Principle:* ‘If a composer intends to write music in which the parts have a high degree of independence, then synchronous note onsets ought to be avoided. Onsets of nominally distinct sounds should be separated by 100ms or more.’ (p.40)

B. Horizontal Integration

The horizontal integration of musical elements (such as notes or chords) relies primarily on two fundamental principles: Temporal and Pitch Proximity. This means that notes close together in terms of time and pitch tend to be integrated perceptually in an auditory stream. These principles are described succinctly by Huron (2001) as follows:

> *Principle of Temporal Continuity:* 'In order to evoke strong auditory streams, use continuous or recurring rather than brief or intermittent sound sources. Intermittent sounds should be separated by no more than roughly 800ms of silence in order to ensure the perception of continuity.' (Huron, 2001, p.12).
>
> *Pitch Proximity Principle:* 'The coherence of an auditory stream is maintained by close pitch proximity in successive tones within the stream. …' (p.24)

Most existing voice separation research takes these two principles as the basis for the development of computational models.

C. Vertical vs. Horizontal Integration

The horizontal integration of tones affects the way tones in vertical sonorities are integrated (and the reverse). Bregman (1990) talks of 'capturing' a tonal component out of a 'mixture'. One of the strongest factors that weakens the vertical links between tones is the appearance of a tone that is proximal to one of the tones of the mixture in terms of both pitch and time. In a sense, there is a competition between the vertical and horizontal principles of auditory grouping. It is exactly this competition that makes it difficult to describe systematically processes of auditory streaming.

In this paper, it is suggested that vertical integration is, in some respect, prior to horizontal sequencing of tones. The idea of capturing a component out of a mixture suggests that the formation of a mixture is anterior to the process of capturing one of its tones into a horizontal stream. This view is in contrast to most models of 'voice' separation that start off with horizontal organisation of streams and then proceed (or at least suggest that one should proceed) with vertical integration of streams into higher-level streams that may contain multiple simultaneous tones.

It is suggested, that a voice separation algorithm should start by identifying synchronous notes that tend to be merged into single sonorities and then use the horizontal streaming principles to break them down into separate streams. This is an optimisation process wherein the various perceptual factors compete with each other in order to produce a 'simple' (as much as this is possible) interpretation of the music in terms of a minimal number of streams (ambiguity, however, should be accommodated).

IV. *VISA:* THE VOICE INTEGRATION/SEGREGATION ALGORITHM

In this section we describe the proposed voice separation algorithm, referred to as Voice Integration/Segregation Algorithm (VISA). We first describe how concurrent notes are merged into single sonorities. Next we detail the proposed algorithm and, finally, we describe the procedures to match notes to voices.

A. Merging Notes into Single Sonorities

During vertical integration, according to the synchronous note principle (described in Section 3.1), we have to determine when to merge concurrent notes, i.e., notes with synchronous onsets and same IOIs. Since it is possible that synchronous notes may belong to different voices, we need a way to decide if such merging should be applied.

Given a set of concurrent notes, the algorithm examines a certain musical excerpt (window) around them. If inside the window, most co-sounding notes have different onsets/offsets, then it is most likely that we have independent monophonic voices so occasional synchronous notes should not be merged.

In particular, let the entire musical piece be represented as a list L of notes that are sorted according to their onset times. For any note n $\in$ L, O(n) denotes its onset time. Moreover, for two notes n_1 and $n_2 \in$ L, with $O(n_1) \leq O(n_2)$, inter(n_1,n_2) denotes the number of all intermediate notes, i.e., all $n_k \in$ L with $O(n_1) \leq O(n_k) \leq O(n_2)$. For a given set S of concurrent notes and a window size w, we consider the set W that contains the notes in a window with length w and centered on S. Thus, W is defined as follows:

$$W=\{n_i \in L\text{-}S \mid \forall\ n \in S\ inter(n_i, n) \leq w/2 \vee inter(n, n_i) \leq w/2\}$$

Next, we examine if concurrency is frequent within *W*. We define the ratio *r* as follows (where IOI(*n*) denotes the inter-onset interval of note *n*):

$$r = \frac{|\{(n_i,n_j) \mid n_i \in W, n_j \in W, O(n_i) = O(n_j) \wedge IOI(n_i) = IOI(n_j)\}|}{|\{(n_i,n_j) \mid n_i \in W,\ n_j \in W, O(n_i) = O(n_j)\}|}$$

Thus, by having a user-defined threshold *T* (in the range [0,1]) that signifies frequency, if *r* > *T*, we merge the notes of *S* as a single sonority.[4]

B. The Algorithm

The Voice Integration/Segregation Algorithm (VISA), receives as input the musical piece in the form of a list *L* of notes that are sorted according to their onset times, a window size *w*, and the threshold *T*. The output is a set of lists *V* (initially empty). After termination, each list contains the notes of each detected voice, each sorted according to onset times. Notice that VISA does not demand a-priori the number of voices. The proposed algorithm is illustrated in Fig. 2.

In VISA, a sweep line, starting from the beginning of L, proceeds in a step-wise fashion (procedure getNextSweepLineSet) to the next onset time in L. The set of notes that have onsets equal to the position of the sweep line is denoted as sweep line set (SLS). Notice that an SLS may contain one or more notes. Next, every SLS is divided into clusters using a procedure called ClusterVertically, which partitions the notes in the SLS

[4] The denominator of *r* is the number of pairs with same onset times, as they represent the potential concurrent notes within *W*.

into a set of clusters C. The ClusterVertically procedure, according to Section 4.1, has to detect contextual information, accepting, thus, as parameters w and T. If, based on context, we decide to merge concurrent notes, each cluster contains all notes with the same IOI (recall that all notes in SLS have identical onset time). Otherwise, if merging is not decided, each cluster contains a single note. The reason for this kind of clustering is two-fold: (i) Concurrent notes are highly probable to belong to the same voice, thus if merging is decided, they are initially placed in the same cluster (as will be explained later on a cluster may split). (ii) Overlapping (in time) but not concurrent notes, are placed into different clusters, because we do not desire any overlapping between notes of the same voice.

```
VISA(NoteList L, Set NoteList V, int w, float T)
begin
    V ← ∅;
    while ((SLS ← getNextSweepLineSet(L)) ≠∅)
    begin-while
        C ←ClusterVertically(SLS, w, T);
        if (|V | < |C |)
        begin-if
            MatchVoicesToClusters(V, C);
        else
            MatchClustersToVoices(C, V);
        end-if
    end-while
end
```

Figure 2 The VISA algorithm.

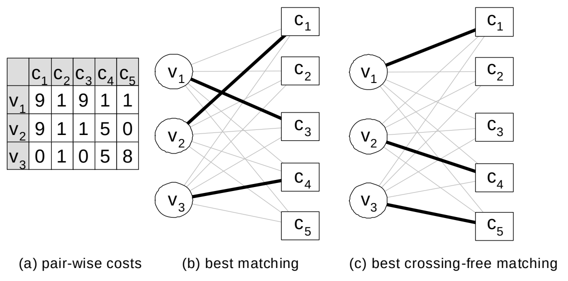
Given the set of clusters, C, we have to assign them to voices. We can form a bipartite graph, where the one set of vertices corresponds to the currently detected voices and the other set of vertices corresponds to the clusters in C. Between every pair of vertices in the bipartite graph, i.e., between every detected voice v_i and cluster c_j, we draw an edge to which we assign a cost. This cost is compound and is determined by the two principles described in Section 3.2. Thus, according to the Principle of Temporal Continuity, we add to the edge cost an amount equal to the difference between the onset time of the notes in c_j and the onset time of the most recent note (or most recent cluster of notes) in v_i.[5] If the notes in c_j and the last note (or last cluster of notes) in v_i overlap in time, then we set the edge cost equal to infinity, in order to forbid the assignment of c_j to v_i. Moreover, according to the Pitch Proximity Principle, we add to the cost the minimum absolute difference between the pitches of notes in c_j and the pitch of the last note (or last cluster of notes) in v_i. Thus, the total cost on the edge represents the temporal and pitch proximity between each pair c_j and v_i.

Having determined the cost on every edge, we can solve the assignment problem by finding the matching with the lowest cost in the bipartite graph. Two cases are possible: (i) If |V| < |C|, i.e., the number of currently detected voices is smaller than the number of clusters in the SLS, then we match voices to clusters (procedure MatchVoicesToClusters). This is done by assigning to each of the currently detected voices a cluster, in a way that the total cost is minimised. The remaining clusters that have not been assigned to a voice, constitute new voices that are added to V. This case is handled inside procedure MatchVoicesToClusters. (ii) Conversely, if |V| ≥ |C|, we match clusters to voices (procedure MatchClustersToVoices), i.e., each cluster is assigned to one of the currently detected voices, in a way that the total cost is minimised. Even though in the latter (ii) case the clusters are fewer than the voices, due to possible overlapping between notes in them, a matching may not be feasible (the total cost equals infinity). We handle this case (inside procedure MatchClustersToVoices) by creating new voices that enable a matching.

Finally, we introduce two extra constraints to the problem of a matching. The first one is that voice crossing should be avoided, thus a sub-optimal solution in terms of cost may be required that avoids voice crossing. The second one is that, according to the Top Voice Rule (Section 2), the matching has to take into account that the top voice should be minimally fragmented. This is handled by adding a penalty *P* to the cost of a matching that does not fulfill this rule. To find the matching with the minimal cost, a cluster may be split into sub-clusters, so that one can be assigned to the top voice. This may hold particularly in the special case where *C* contains a single cluster and there are more than one voices. More details about the inclusion of the two constraints in the matching procedures, are given in Section 4.3.

C. The Matching Process in Detail

For convenience, we convert the minimisation problem to an equivalent mazimization one. Therefore, we are interested in maximising the total matching instead of minimising it. For this reason, the assignment of the cost $w(e_{ij})$ between a voice v_i and a cluster c_j is converted to $\max\{e_{kl}\} - w(e_{ij})$, where $\max\{e_{kl}\}$ is the maximum edge cost determined for the specific instance of the matching problem (and this cost is due to the edge connecting voice v_k and cluster c_l) .



	c_1	c_2	c_3	c_4	c_5
v_1	9	1	9	1	1
v_2	9	1	1	5	0
v_3	0	1	0	5	8

Figure 3 Maximum matching examples

The traditional bipartite matching algorithms do not preserve the order of the matching. In our case, order preservation is important because voice crossing should be avoided. By enforcing the rule that each matched pair should not 'intersect' another matched pair, a new problem is formulated that can not be directly tackled by bipartite matching algorithms. This issue is depicted in Fig. 3, where an instance is illustrated with three voices and five clusters. Fig. 3a gives the pair-wise costs for assigning voices to clusters. A maximum weighted matching is given in Fig. 3b, with a total cost of 9+9+5 = 23. Evidently, the maximum weighted matching in this graph does not necessarily avoid voice crossing. A

[5] In our implementation we assume that onset times are given in milliseconds.

crossing-free maximum weighted matching with cost 9+5+8 = 22 is depicted in Fig. 3c.

Although the number of voices and clusters is usually small (i.e., 3, 4, 5) we propose en efficient solution which can handle larger numbers of voices and clusters. The naïve approach to determine the best crossing-free matching is to perform an exhaustive search. This technique does not scale well for larger number of voices and clusters.

The proposed matching algorithm is based on dynamic programming (Cormen et. al. 2001). Let *VSEQ* denote the sequence of notes, and *CSEQ* be the sequence of clusters. Without loss of generality, we assume that we have less voices than clusters, i.e. $|VSEQ| < |CSEQ|$. Equality is trivially solved by matching the first voice to the first cluster, the second voice to the second cluster and so forth. The case where $|VSEQ| > |CSEQ|$ is handled similarly. Let M_{ij} denote the current total matching cost after voice v_i and cluster c_i have been matched. The recurrence formula used by the dynamic programming technique in our case is the following:

$$M_{ij} = \max\{M_{i-1,j-1} + w(i,j), M_{i-1,j}\}$$

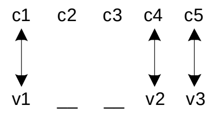
This formula states that either voice v_i will be matched with cluster c_j, or a gap will be placed in the voice sequence, meaning that we postpone the matching of v_i. We illustrate the matching process by using the example instance given in Fig. 3. Therefore, $VSEQ = v_1, v_2, v_3$ and $CSEQ = c_1, c_2, c_3, c_4, c_5$. The matching process for is depicted in Fig. 4a, where each cell of the matrix M represents the total matching cost. The matrix has 3+1=4 rows and 5+1=6 columns (an additional row and column have been placed). The matrix is filled according to the previous recurrence equation, by starting at the upper-left cell and ending at the lower-right one. Initially we place a zero in the first row and first column of the matrix. The cost of the maximum matching is shown in the lower-right cell, which contains the value 22.

		c1	c2	c3	c4	c5
	0	0	0	0	0	0
v1	0	9	9	9	9	9
v2	0	9	10	10	14	14
v3	0	0	10	10	15	22

(a) matching costs

		c1	c2	c3	c4	c5
	0	0	0	0	0	0
v1	0	**9**	**9**	**9**	9	9
v2	0	9	10	10	**14**	14
v3	0	0	10	10	15	**22**

(b) matching path



(c) final assigment

Figure 4 The matching process

The best matching cost by itself does not give the assignment of voices to clusters. To achieve this, the matching path needs to be determined. We perform a trace-back process starting at the cell which contains the best matching value (i.e. cell with the value 22 in the matrix). Based on the fact that the matching cost of v_3 and c_5 is 8, the only possible predecessors are the top and diagonal cells which contain the value 14. However, in the trace-back process we never choose a vertical cell, since no gaps are allowed to be placed on the cluster sequence, meaning that all voices must be matched. The only alternative left is to choose the diagonal cell. This means that voice v_3 will be matched to cluster c_5. The current cell is now (2,4) which contains the value 14. Again, since the matching cost of v_2 and c_4 is 5, the only valid predecessor is cell (1,3) which contains the value 9 (recall that we never move vertically). Since we have chosen to move diagonally, we match voice v_2 to cluster c_4. The next two steps involve a horizontal movement, meaning that we place two gaps in the voice sequence. These steps take us to cell (1,1) from where we can only move diagonally to cell (0,0). This diagonal movement implies an assignment of voice v_1 to cluster c_1. The final assignment is given in Fig. 4c, whereas the matching path is given in Fig. 4b, and is represented by the shaded cells.

According to the previous discussion, the run time of the algorithm is O(n*m) (n>=2, m>=2) where n is the number of voices and m the number of clusters. Evidently, we need O(n*m) time to calculate all elements of the matrix M, and O(n+m) time to reconstruct the matching path. On the other hand, if an exhaustive algorithm is used the number of all available crossing-free matchings that can be produced are C(m,n), which are all possible combinations of selecting n out of m items. For some values of m and n, the exhaustive algorithm performs better than dynamic programming. Therefore, according to the current values of m and n we can decide whether to use the exhaustive method or switch to dynamic programming. However, even if in several cases the exhaustive method performs better, counting all possible matchings and selecting the best one may become tedious, since we have to keep track of which matchings have been examined and which have not. On the other hand, dynamic programming offers a more clear framework and determines the best matching in a more manageable and systematic way.

V. EXPERIMENTS AND RESULTS

A. Test Data

The proposed algorithm has been tested on a small set of musical works for piano. Four pieces with clearly defined streams/voices are used as groundtruth for testing the performance of the algorithm. The first two pieces are two fugues from the first book of the Well-Tempered Clavier by J.S.Bach (Fugue No.1 in C major, BWV846, and Fugue No.14 in F# major, BWV859); these polyphonic works consist of four independent voices. A mazurka by F.Chopin (Mazurka, Op.7, No.5) consists of a melody (upper staff) and accompanying harmony (lower staff). Finally, the “Harmony Club Waltz” by S.Joplin has two parallel homophonic streams (chordal ‘voices’) that correspond to the two piano staves. See musical excerpts in Figs 5, 6, and 7.

In this pilot study, the aim was to examine whether a single algorithm can be applied to two very different types of music (i.e. pure polyphonic music and music containing clear homophonic textures). All the parameters of the algorithm are the same for all four pieces; the number of streams/voices is determined automatically (not set manually). It should be noted that for the two pieces by Chopin and Joplin all other voice separation algorithms would automatically determine at least four different voices (up to eight voices) that do not have perceptual validity (and musicologically are problematic).

Annotated datasets for musical streams/voices - as voices are defined in this paper - do not exist. A small dataset was therefore selected for which it is assumed that musicologists/musical analysts would unreservedly agree on the number of independent musical streams in each piece.[6] Working on a small dataset has enabled both quantitative and qualitative evaluation of the results (all the results have been analysed note-by-note and mistakes have been categorised in types of problems – see section 5.2). A larger dataset, however, is currently assembled in order to run larger scale tests.

At present, the algorithm has been applied to quantised musical data (symbolic scores converted to MIDI). Expressively performed musical data (e.g. expressive MIDI) can be quantised (e.g., see Cambouropoulos 2000) before being fed into the algorithm.

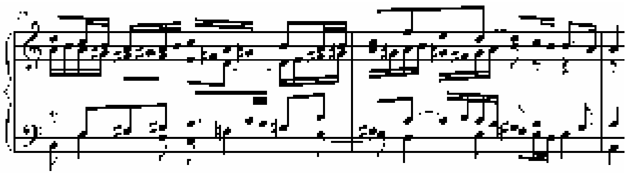


Figure 5 The algorithm performs voice separation correctly in this excerpt from the Fugue No.1 in C major, WTCI, BWV846 by J.S.Bach except for the last five notes of the upper voice which are assigned to the second voice (see text).

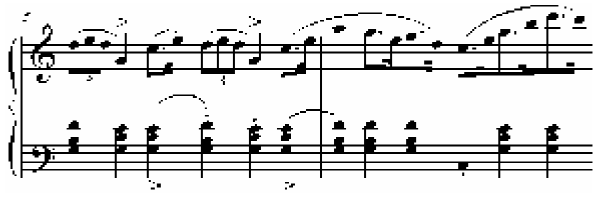


Figure 6 Two independent streams/voices (melody and accompaniment) are correctly determined by the algorithm in this excerpt from the Mazurka, Op.7, No.5 by F.Chopin.

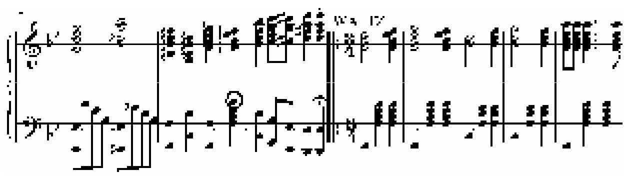


Figure 7 Two independent chordal streams/voices are correctly determined by the algorithm in this excerpt from the "Harmony Club Waltz" by S.Joplin - mistake is indicated by the circled note (see text).

B. Quantitative Results

The evaluation metric used is the precision of the obtained result. For the previously described musical dataset, Table 1 shows the results. The effectiveness of the proposed methodology is evident by the high precision rates achieved for all eight pieces.

TABLE I.
RESULTS IN TERMS OF PRECISION FOR THE FOUR PIECES

Musical Work	*Precision*
J.S.Bach, Fugue No.1 in C major, BWV846	92,38%
J.S.Bach, Fugue No.14 in F# major, BWV859	95,56%
F. Chopin, Mazurka, Op.7, No.5	100%
S. Joplin, "Harmony Club Waltz"	98.12%

Additionally, we have experimented with the impact of the user-defined threshold value *T* on the precision of the proposed algorithm. The experiment (Fig. 8) refers to the examination of the attained precision with respect to the user-defined threshold *T*. The mazurka and waltz datasets naturally exhibit increased synchronicity of notes, thus lower values of *T* give high precision, while on the contrary, the two fugues do not include notes that can be merged into single sonorities and thus require higher values of *T*.

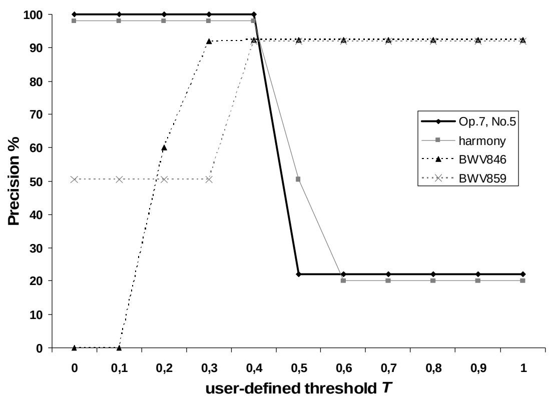


Figure 8 Precision vs. user-defined threshold *T*.

C. Qualitative Analysis

The aforementioned results were examined in detail in order to understand the kinds of mistakes produced by the algorithm. Most of these problems are, in a sense, expected and cannot be solved merely when taking into account only pitch and temporal distances between notes.

The majority of wrong results were given in cases where the number of voices change and erroneous connections are introduced due primarily to pitch proximity (for instance, in Fig. 5 the algorithm erroneously 'gives' the last five notes of the upper voice to the second voice simply because the first of these notes is closer by a semitone to the last note of the second voice). Kilian and Hoos (2002) address this same problem claiming that, in essence, it is unsolvable at the note level ('It seems that when only considering the notes … there is no reason why another separation should be preferred.' p.45).

A second kind of problem involves voice crossing. Since voice crossing is disallowed by the algorithm notes at points where voices cross (in the Bach fugues) are assigned to wrong voices.

A third type of mistake relates to the breaking of vertically merged notes into sub-sonorities and allocating these to different voices; in this case the breaking point in

[6] These independent 'voices' correspond to separate spines in the kern format; all test pieces have been obtained from KernScores <http://kern.humdrum.org>.

the sonority may be misplaced (see, for instance, circled note in Fig. 7).

The success rate (see previous section) of the algorithm on this small but diverse dataset is remarkable. It should be emphasised that the algorithm is capable of detecting different number of voices in the same piece automatically, not only in cases where a 'monophonic' voice may disappear for a while and reappear later on (e.g., fugues), but also in cases where polyphonic/homophonic textures change resulting in a different number of multi-note voices (e.g., example in Fig. 9).

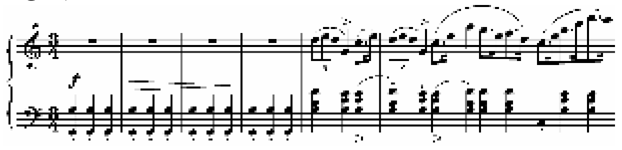


Figure 9 In the opening of the Mazurka, Op.7, No.5 by F.Chopin, the algorithm detects correctly one voice (low octaves) and, then, switches automatically to two voices (melody and accompaniment).

VI. CONCLUSIONS

In this paper the notions of voice and auditory stream have been examined, and an attempt has been made to clarify the various meanings. It is suggested that if 'voice' is understood as a musicological parallel to the concept of auditory stream, then multi-note sonorities should be allowed within individual 'voices'.

It is proposed that a first step in voice separation is identifying synchronous note sonorities and then breaking these into sub-sonorities incorporated in horizontal streams or 'voices'. This proposal is in direct contrast with most computational systems that start by finding first horizontal 'voices' and then merging these into higher level 'voices' (actually, the latter step has not been implemented by any of the aforementioned computational models).

The proposed voice separation algorithm incorporates the two principles of temporal and pitch proximity, and additionally, the Synchronous Note Principle and the Tonal Fusion Principle. Allowing both horizontal and vertical integration enables the algorithm to perform well not only in polyphonic music that has a fixed number of 'monophonic' lines but in the general case where both polyphonic and homophonic elements are mixed together. We have shown in the above experiments that a single algorithm, with the same parameters, can achieve very good performance in diverse musical textures in terms identifying perceptually pertinent voices/streams.

The pilot study reported in this paper gives promising results in the domain of voice separation. Future work involves testing the algorithm on a much larger database and, also, incorporating additional principles such as the Pitch Co-modulation Principle (notes that move in parallel intervals, especially in octaves, are strongly integrated) .

REFERENCES

[1] Bregman, A (1990) *Auditory Scene Analysis: The Perceptual Organisation of Sound*. The MIT Press, Cambridge (Ma).

[2] Cambouropoulos, E. (2006) 'Voice' Separation: theoretical, perceptual and computational perspectives. In *Proceedings of the 9th International Conference in Music Perception and Cognition (ICMPC2006)*, 22-23 August, Bologna, Italy.

[3] Cambouropoulos, E. (2000) From MIDI to Traditional Musical Notation. In *Proceedings of the AAAI Workshop on Artificial Intelligence and Music: Towards Formal Models of Composition, Performance and Analysis*, July 3 - Aug. 3, Austin Texas.

[4] Cormen, T., Leiserson, C.E., Rivest, R.L. and Stein, C (2001). *Introduction to Algorithms*, The MIT Press.

[5] Chew, E. and Wu, X. (2004) Separating voices in polyphonic music: A contig mapping approach. In *Computer Music Modeling and Retrieval: Second International Symposium* (CMMR 2004), pp. 1-20.

[6] Deutsch, D. (1999) Grouping Mechanisms in Music. In D. Deutsch (ed.), *The Psychology of Music* (revised version). Academic Press, San Diego.

[7] Huron, D. (2001) Tone and Voice: A Derivation of the Rules of Voice-Leading from Perceptual Principles. *Music Perception*, 19(1):1-64.

[8] Kilian j. and Hoos H. (2002) Voice Separation: A Local Optimisation Approach. In *Proceedings of the Third International Conference on Music Information Retrieval* (ISMIR 2002), pp.39-46.

[9] Kirlin, P.B. and Utgoff, P.E. (2005) VoiSe: Learning to Segregate Voices in Explicit and Implicit Polyphony. In *Proceedings of the Sixth International Conference on Music Information Retrieval* (ISMIR 2005), Queen Mary, University of London (pp. 552-557).

[10] Madsen, S. T. and Widmer, G. (2006) Separating Voices in MIDI. In *Proceedings of the 9th International Conference in Music Perception and Cognition (ICMPC2006)*, 22-26 August 2006, Bologna, Italy.

[11] Marsden, A. (1992) Modeling the Perception of Musical Voices: a Case Study in Rule-based Systems. In *Computer Representations and Models in Music*, Marsden, A. and Pople, A. (eds), Academic Press, London.

[12] Temperley, D. (2001) *The Cognition of Basic Musical Structures*. The MIT Press, Cambridge (Ma).

[13] Szeto, W.M. and Wong, M.H. (2003) A Stream Segregation Algorithm for Polyphonic Music Databases. In *Proceedings of the Seventh International Database Engineering and Applications Symposium* (IDEAS'03).