

# VISA: THE VOICE INTEGRATION/SEGREGATION ALGORITHM

Ioannis Karydis

Alexandros  
Nanopoulos

Apostolos  
Papadopoulos

Emilios  
Cambouropoulos

Department of Computer Science  
Aristotle University of Thessaloniki  
{karydis, ananopou, apostol}@csd.auth.gr

Dept. of Music Studies  
Aristotle Univ. of Thessaloniki  
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. This is paramount, among others, 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.

## 1. INTRODUCTION

Recently, there have been a number of attempts [3, 5, 8, 9, 10, 11, 12] to model computationally the segregation of polyphonic music into separate voices. Much of this research is influenced by empirical studies in music perception [1, 6, 7] as well as by musicological concepts such as melody, counterpoint, voice-leading and so on.

It appears that the term ‘voice’ has different meanings for different research fields (traditional musicology, music cognition, computational musicology). A detailed discussion is presented in [1]. A single musical example is given (Figure 1) that presents three different meanings of the term voice.



**Figure 1** Number of voices: in terms of literal voices (human or monophonic) 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).

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,<sup>1</sup> then, it becomes clear that the standard meaning is not sufficient. It is possible that a single monophonic sequence may be perceived as more than one voices/streams (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).

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 Figure 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 (Figure 1b) and three voices in the third example (Figure 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 eight different musical works are presented.

## 2. RELATED WORK

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 assist 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

<sup>1</sup> 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.

different parts (perceptually implausible) neither in voices that are not perceptually independent (e.g. inner parts in an accompaniment) but within voices that are heard as having a life of their own.

Voice separation algorithms such as [3, 5, 8, 9, 10, 11, 12] assume that ‘voice’ is a monophonic sequence of successive non-overlapping musical tones (exception is the model by Kilian and Hoos [8] which is discussed below). The underlying perceptual principles that organise tones in voices are the principles of temporal and pitch proximity (cf. Huron’s [7] 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 (maximum number of voices is equal to the 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).

Kilian and Hoos’s [8] model is pioneering in the sense that multi-note sonorities within single voices are allowed. The pragmatic goal of the algorithm is the derivation of reasonable score notation - not perceptually meaningful voices (see [8], p.39). The algorithm is based on perceptual principles in terms of a cost function that sums penalty values for features promoting segregation, such as large pitch intervals, rests/gaps, and note overlap between successive notes, and large pitch intervals and onset asynchrony within chords; penalty values are user-defined. The results, however, 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 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 mentioned above.

Kilian and Hoos’s model allows multiple synchronous or overlapping tones in a single voice 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 (see next section). For instance, their 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, overlapping tones with different onsets and durations do not tend to 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 2<sup>nd</sup> apart are less likely to be fused (see next section). The

perceptual factor of tonal fusion is not taken into account by the model.

### 3. 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 [7] 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 four of these principles as the starting-point of our exploration.

#### 3.1. Vertical Integration

Bregman [1] 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 [7], 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 that start, evolve and finish together tend to be merged perceptually. The proposed principle (below) relates to Huron’s Onset Synchrony Principle but it differs in a number of ways as discussed by Cambouropoulos [2].

*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:

*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 ([7], 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 [8].

Finally, according to the Pitch Co-modulation Principle: ‘The perceptual union of concurrent tones is encouraged when pitch motions are positively correlated.’ ([7], p.31) The strongest manifestation of this principle is when notes move in parallel intervals (especially in octaves). This principle has not yet been incorporated in the current version of the algorithm.

#### 3.2. Horizontal Integration

The horizontal integration of musical elements (such as notes or chords) relies primarily on two fundamental principles: Temporal and Pitch Proximity. These principles are described by Huron [7] 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.’ (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 computational models.

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 (most algorithms ignore the vertical component). This is an optimization process wherein the various perceptual factors compete with each other in order to produce a ‘simple’ interpretation of the music in terms of a minimal number of streams.

#### 4. VISA: THE VOICE INTEGRATION/SEGREGATION ALGORITHM

In this section we describe the proposed voice separation algorithm, referred to as Voice Integration/Segregation Algorithm (VISA).

##### 4.1. 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 centred on  $S$ . Thus,  $W$  is defined as follows:

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

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.<sup>1</sup>

#### 4.2. 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 Figure 2.

```
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.

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 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.

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.

<sup>1</sup> The denominator of  $r$  is the number of pairs with same onset times, as they represent the potential concurrent notes within  $W$ .

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  $v_i$ . 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  $v_i$ .

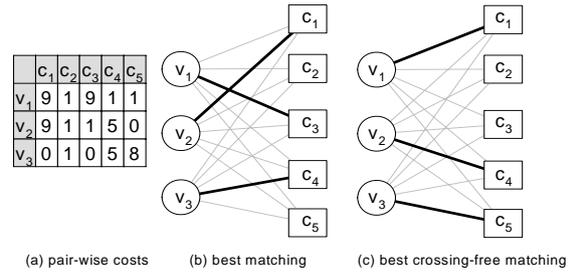
Having determined the cost on every edge, we can solve the assignment problem by finding the *matching* with the lowest cost in the bipartite graph. Two cases are possible: (i) If  $|V| < |C|$ , i.e., the number of currently detected voices is smaller than the number of clusters in the SLS, then we match 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 minimized. 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  $|V| \geq |C|$ , 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 minimized. 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 [11], 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 fulfil 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.

### 4.3. The Matching Process

For convenience, we convert the minimisation problem to an equivalent maximisation 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  $c_l$ ).



**Figure 3** Maximum matching examples

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 Figure 3, where an instance is illustrated with three voices and five clusters. Figure 3(a) gives the pair-wise costs for assigning voices to clusters. A maximum weighted matching is given in Figure 3(b), with a total cost of  $9+9+5 = 23$ . Evidently, the maximum weighted matching in this graph does not necessarily avoid voice crossing. A crossing-free maximum weighted matching with cost  $9+5+8 = 22$  is depicted in Figure 3(c).

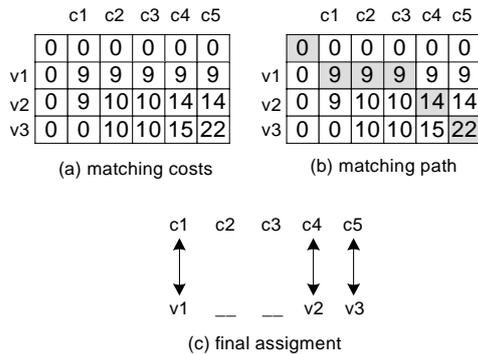
Although the number of voices and clusters is usually small (i.e., 3, 4, 5) we propose an efficient solution which can handle larger numbers of voices and clusters. The naïve approach to determine the best crossing-free matching is to perform an exhaustive search. This technique does not scale well for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



**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 Figure 4(c), whereas the matching path is given in Figure 4(b), and is represented by the shaded cells.

According to the previous discussion, the running time of the algorithm is  $O(n*m)$  ( $n \geq 2, m \geq 2$ ) 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.

## 5. EXPERIMENTS AND RESULTS

The proposed algorithm has been tested on a small set of musical works for piano. Eight pieces with clearly defined streams/voices are used as groundtruth for testing the performance of the algorithm. The first four pieces are two fugues and two inventions by J.S.Bach; these polyphonic works consist of independent monophonic voices. Two mazurkas and a waltz by F.Chopin consist 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 excerpts in Figures 5, 6, 7.

In this pilot study, the aim was to examine if 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 eight pieces; the number of streams/voices is determined automatically (not set manually). It should be noted that for the pieces by Chopin and Joplin all other voice separation algorithms would determine automatically 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.<sup>1</sup> At present, the algorithm has been applied to quantised musical data (symbolic scores converted to MIDI) - expressively performed musical data can be quantised (e.g., [3]) before being fed into the algorithm.



**Figure 5** Four independent streams/voices are present in this excerpt from the Fugue No.1 in C major, WTCl, BWV846 by J.S.Bach. The algorithm performs voice separation correctly except for the last five notes of the upper voice which are assigned to the second voice rather than the first voice (as these are closer by a semitone to the last note of the second voice).



**Figure 6** 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).

<sup>1</sup> These independent ‘voices’ correspond to separate spines in the kern format; all four test pieces have been obtained from KernScores <<http://kern.humdrum.org>>.



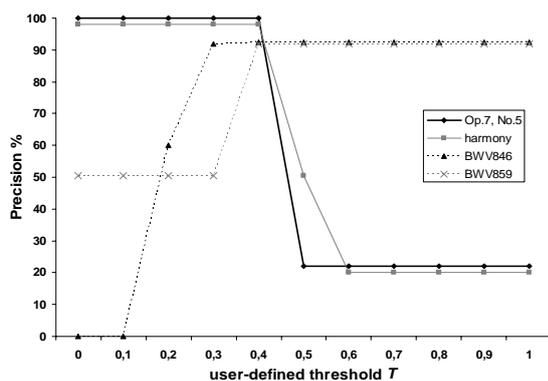
**Figure 7** Two independent chordal streams/voices are correctly determined by the algorithm in this excerpt from the “Harmony Club Waltz” by S.Joplin; the only mistake is indicated by the circled note which is placed ‘erroneously’ in the upper stream (because of pitch proximity).

The evaluation metrics 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.

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%
J.S.Bach, Invention No.1 in C Major, BWV 772	99.34%
J.S.Bach, Invention No.13 in A Min, BWV 784	96.45%
F. Chopin, Mazurka, Op.7, No.5	100%
F. Chopin, Mazurka in A Minor, Op. 67, No.4	85.00%
F. Chopin, Waltz in B Minor, Op. 69, No. 2	90.31%
S.Joplin, “Harmony Club Waltz”	98.12%

**Table 1** Results in terms of precision for the dataset.

Additionally, we have experimented with the impact of the user-defined threshold value  $T$  on the precision of the proposed algorithm (Figure 8). 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  $T$ .

The results were examined in detail (qualitative analysis) in order to understand the kinds of mistakes produced by the algorithm. The majority of wrong results were given in cases where the number of voices change and erroneous connections are introduced primarily due to pitch proximity (for instance, see last upper five notes in Figure 5). Kilian and Hoos [8] address this same problem claiming that, in essence, it is unsolvable at the note level. 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 the sonority may be misplaced (see, for instance, circled note in Figure 7).

## 6. CONCLUSIONS

In this paper the notions of voice and auditory stream have been examined. 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’.

The proposed voice separation algorithm, *VISA*, 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 (see, for instance, Figure 6). We have shown in the above experiments that a single algorithm, with the same parameters, can achieve good performance in diverse musical textures in terms of identifying perceptually pertinent voices/streams. Ongoing research involves testing the algorithm on a much larger database; future work will incorporate additional principles such as the Pitch Co-modulation Principle.

## 7. 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 9<sup>th</sup> 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 AAI Workshop on Artificial Intelligence and Music*, 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, Cambridge (Ma).
- [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, Univ. of London (pp. 552-557).
- [10] Madsen, S. T. and Widmer, G. (2006) Separating Voices in MIDI. In *Proceedings of the 9<sup>th</sup> International Conference in Music Perception and Cognition (ICMPC2006)*, 22-26 August 2006, Bologna, Italy.
- [11] Temperley, D. (2001) *The Cognition of Basic Musical Structures*. The MIT Press, Cambridge (Ma).
- [12] 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)*.