TUNE CLASSIFICATION USING MULTILEVEL RECURSIVE LOCAL ALIGNMENT ALGORITHMS

Chris Walshaw
Department of Computing & Information Systems,
University of Greenwich, London SE10 9LS, UK
c.walshaw@gre.ac.uk

ABSTRACT

This paper investigates several enhancements to two well-established local alignment algorithms in the context of their use for melodic similarity. It uses the annotated dataset from the well-known Meertens Tune Collection to provide a ground truth and the research aim to answer the question, to what extent do these enhancements improve the quality of the algorithms? In the results, recursive application of the alignment algorithms, applied to a multilevel representation of the melodies, is shown to be very effective for improving the accuracy of the classification of the tunes into families. However, the ideas should be equally applicable to music search and melodic matching.

1. INTRODUCTION

1.1 Background

In the field of music information retrieval an important topic, impacting on music search, melodic matching / clustering and tune classification, is the calculation of melodic similarity. This paper investigates several enhancements to two well-established local alignment algorithms in the context of their use for melodic similarity.

It builds on results established by Janssen *et al.*, [1], [2] and van Kranenburg *et al.*, [3], which suggest that alignment-based similarity measures provide some of the best results for matching melodic segments, or indeed whole melodies, in a corpus of folk songs.

It uses an annotated dataset from the Meertens Tune Collection, [4], to provide a ground truth with which to evaluate the quality of the enhancements and for that reason deals with classification of melodies into tune families. However, the algorithms should be equally applicable to music search, e.g. [5], and melodic matching, e.g. [6], where some of the ideas were originally presented.

1.2 Organisation

The paper is organised as follows: section 2 presents the baseline algorithms and four enhancements. Section 3 discusses the evaluation and section 4 presents the results. Conclusions and further work are discussed in section 5.

2. ALGORITHMIC VARIANTS

This section discusses the alignment-based algorithms tested for the classification problem. Initially the two baseline algorithms, local alignment and longest common substring, are outlined (section 2.1). Subsequently a number of enhancements are discussed, including length normalisation (section 2.2) and the rhythmic representation of the melody (section 2.3), as well as globalising and multilevel enhancements (sections 2.4 & 2.5).

2.1 Baseline Similarity Measures

2.1.1 Local alignment (LA)

Local alignment is a well-known technique originating from molecular biology. Given two strings it finds the optimal alignment for two sub-sequences of the originals. The algorithm does not require the aligned sub-sequences to match exactly and makes allowances for gaps and substitutions. For example the strings `***abcde**` and `*acfe****` (where the asterisks represent non-matching entries) could potentially be aligned between `a` and `e` with a gap at the `b` and the substitution of `d` for `f`. Gaps (otherwise known as insertions and deletions) and substitutions are penalised with weights.

The algorithm is known as local alignment (LA) because, unlike the global alignment algorithms which preceded it, mismatching sub-strings from either side of the alignment are not penalised (i.e. in the example, the string of non-matching entries, indicated by asterisks, could be arbitrarily long without changing the alignment score).

To compute the optimal local alignment for two strings of length m & n, an (m+1) x (n+1) score matrix A is constructed with the top row and left hand column initialised to zero. The remainder of the matrix is then filled using

$$A(i,j) = \max \begin{cases} A(i-1,j-1) + s(X_i, Y_j) \\ A(i,j-1) + W_{\text{gap}} \\ A(i-1,j) + W_{\text{gap}} \\ 0 \end{cases}$$

$$\text{where } s(X_i, Y_j) = \begin{cases} W_{\text{match}} & \text{if } X_i = Y_j \\ W_{\text{substitution}} & \text{if } X_i \neq Y_j \end{cases}$$

and where W_{match}, $W_{\text{substitution}}$ and W_{gap} represent the weights for a matching or substituted entry or a gap in the aligned sequences. The implementation discussed here follows Janssen *et al.*, [1], [2], and uses $W_{\text{match}} = 1$, $W_{\text{substitution}} = -1$ and $W_{\text{gap}} = -0.5$.

This algorithm was introduced by Smith & Waterman, [7]. In fact their original scheme is a little more computationally involved but the scheme above is widely used and is the variant tested by Janssen *et al.*

To calculate the alignment score, and hence the qualitative similarity, the above scheme suffices. However to determine the aligned sub-sequences (needed for recursion, section 2.4) a trace-back procedure is required. The trace-back is implemented by recording a matrix of DIAG, UP or LEFT pointers for every entry of the score matrix indicating where the maximum value originated. If the maximum value is zero an END pointer is stored.

The trace-back starts at the pointer matrix entry corresponding to the maximum score found and then tracks back through the pointers, terminating when it reaches an END. Diagonal moves indicate contiguous values in the two aligned sub-sequences whilst left or up moves indicate a gap in one of them.

2.1.2 Longest Common SubString (LCSS)

The longest common substring algorithm also finds matched sub-sequences from two strings but requires the sub-sequences to match exactly with no gaps or substitutions. It operates in a very similar fashion to local alignment filling in an (m+1) x (n+1) matrix of alignment values. However, because there is no need to allow for gaps, no trace-back is required: the position of the maximum score in the matrix indicates the end of the longest common substring and the value of this entry gives its length.

2.1.3 Sub-sequence alignment

In fact it is easy to see that, if the local alignment weights $W_{substitution}$ and W_{gap} are sufficiently large so that gaps and substitutions can never occur, then the LCSS algorithm is just a special case of local alignment.

From here on, therefore, both algorithms, LA and LCSS, will be referred to collectively as sub-sequence alignment, the main distinction between the two being that LCSS produces exact matching aligned substrings, is faster to compute and requires less memory (there is no need to use a full matrix and a memory efficient version exists which just repeatedly swaps a pair of arrays, one containing the row under calculation and one containing the previous row). Conversely, LA is more computationally complex and more memory intensive, but will generally match longer strings.

Both algorithms can be used for melodic similarity by representing each melody as a sequence of pitches or intervals: here intervals are used (see section 3.2). Then, if using $W_{match} = 1$ for LA, the similarity measure , S_{XY}, that either algorithm calculates between a pair of melodies, X and Y, represents the length (the number of notes) of the sub-sequences aligned. However, in the case of LA there may also be penalty weights for gaps or substitutions (for example, the matching of `abcde` with `acfe` has a score of 1 – ½ + 1 – 1 + 1 = 1½).

2.2 Normalisation

The first simple algorithmic variant is just the way that the raw similarity measure is normalised. In their papers, [1], [2], Janssen *et al.* normalise the similarity S_{XY} by dividing by the length of the query. In the context of their use of short melodic phrases to query a database of melodies, and since the longest query is normally much shorter than the shortest melody, this means that the similarity measure is effectively normalised by min(length(X), length(Y)), In addition, since the maximum value possible of S_{XY} is also min(length(X), length(Y)), i.e. the length of the shortest of the two sequences being compared, then S_{XY} gives a value between 0.0 and 1.0 (with 1.0 being returned when an exact match of the query is found within the melody being queried).

For phrase-based classification studied in [1], [2], this makes perfect sense; there is no expectation that the query will match the entire tune. However, for the tune-based classification discussed here, that is no longer true and so using the minimum length may no longer be appropriate. For example, consider matching the sequence `abc` with two other sequences, `abc` and `abcdef`. In both cases the raw similarity is 3 and using minimum length (also 3) to normalise, the normalised similarity is 3/3 = 1.0. However, it does seem unreasonable that the similarity is the same in both cases (particularly since the match with the first sequence is exact, whereas the sequence `abcdef` could be arbitrarily long without changing the result).

Alternatives are to normalise with the maximum length or the average length. For example consider `abcxyz` matching with `abcdef` and `abcdefghi`. Using minimum length the normalised similarity is 0.5 for both matches (3/6) which doesn't seem unreasonable. However, using either maximum or average length, a sequence with identical raw similarity (in this case 3) to two other sequences will be normalised as closer to the one which is of a similar length, and arguably this is more appropriate.

Because it is not immediately clear which normalisation to use, the experimentation tests all three empirically.

2.3 Representation – bar indicators

A second algorithmic variant tries to take account of the position of notes within the bar. This is particularly relevant for folk music (although perhaps less so for jazz) since the position often determines which are the stressed (more important) notes.

One way to achieve this is to adapt the similarity measure to add weight to stressed notes, e.g. [8]. However, that relies on what is arguably a subjective assessment of which are the stressed notes. Instead, as discussed in [5], it is possible to use bar indicators or even bar numbers in the sequences of intervals to be compared. For example a major scale can be indicated by the intervals `2212221`. Including bar indicators, and assuming 4 notes to the bar this could be represented as `|221|2221|` where the `|` symbols represent bar lines (note that an interval between the last note in a bar and the first note in the next bar, could be shown before or after the bar indicator; in the experiments here it is always included afterwards). This means that any matched common substrings must respect bar lines (unless they are shorter than the length of a bar).

Furthermore, if the bar symbols are numbered, e.g. $|_1221|_22221|_3$, where each $|_i$ represents a numbered bar, then matched common substrings also need to respect the position in the tune. (If matching of subsections of the tune is important then the numbering could be restarted at natural breaks such as double bar lines and repeat marks; however, that is not tested here.)

In terms of implementation, the “strings” of intervals are represented as an array of short integers so that bar markers (or numbers) can easily be included with large integer values outside the possible range of intervals.

This inclusion of bar indicators is more of a representational variant than an algorithmic one and increases the computational complexity of the matching slightly (as the strings to be compared by the similarity measure are longer). However, even though some melodies in the dataset under investigation are in free meter and have no bar lines, it has a significant effect on the results and is an important enhancement.

2.4 Recursive sub-sequence alignment

2.4.1 Recursive alignment (= global alignment)

A problem with using LCSS, and to a lesser extent LA, is that they are local. For example, using LCSS, `ab**ba` has exactly the same raw alignment score (of 2) when matched with `**ab` and with `ab**ba`, even though the latter seems a far better match. This is because the second match (`ba`) is not accounted for.

This was less of an issue in the predecessor to this paper, [9], where LCSS was used as part of a multilevel melodic search algorithm, since search algorithms are typically trying to find the best matches of a short phrase in a dataset of complete melodies. However, for classification it is crucial to distinguish between tunes which match well across their entire length and those which perhaps only match for a short segment.

Interestingly Smith & Waterman touch on this in their original paper where they say "the pair of segments with the next best similarity is found by applying the trace-back procedure to the second largest element of [the matrix] not associated with the first trace-back", [7]

Unfortunately, working from the existing matrix may lead to overlapping local alignments and instead subsequence alignment may be applied recursively as follows: when applied to two strings, S1 and S2, subsequence alignment splits both into three substrings S1 = L1 + A1 + R1 and S2 = L2 + A2 + R2, where A1 and A2 are the aligned substrings (exact matches for LCSS or potentially with gaps and substitutions for LA), L1 and L2 are the left hand side unmatched substrings and R1 and R2 are the right hand side unmatched substrings (where any of the these unmatched substrings may be of length 0). Thus, having found A1 & A2 and split S1 & S2, subsequence alignment can then be applied to compare L1 with L2 and R1 with R2.

This procedure continues recursively, terminating when no alignment is found, or one or both lengths of the substrings being aligned are 0. For example, if the start of S1 is aligned with the end of S2 no further recursion is possible as the lengths of L1 and R2 are 0.

This recursion effectively turns the local alignment algorithms LCSS or LA into a globalised similarity measure, giving an alignment score along the length of both strings being compared. Henceforth these recursive algorithms will be referred to as RLCSS and RLA.

2.4.2 Biased recursive local alignment

An issue that became apparent when using recursive alignment, is that just adding all the scores together makes no distinction between one long aligned sequence and several shorter ones. For example (using RLCSS) `abcd****` has the same alignment score (of 4) when compared with `abcd****` and with `**a**b**c**d**`, even though the former seems a good match and the matching with the latter is essentially noise.

To address this, the similarity measure can be biased towards longer aligned sub-sequences by taking the 2-norm (square root of the sum of squares) of the alignment scores found by the recursive local alignment. In the above example this means that the biased recursive local alignment score is $\sqrt{4^2} = 4$ when matching `abcd****` with `abcd****`, whereas when matching with `**a**b**c**d**` it is $\sqrt{1^2 + 1^2 + 1^2 + 1^2} = 2$.

2.5 Multilevel Similarity

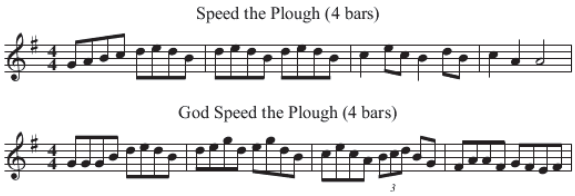

Figure 1. Two tune variants for Speed the Plough.

Multilevel similarity was first introduced in [5] and subsequently developed further in [6]. The idea is motivated in Fig. 1 which shows two versions of the first 4 bars of Speed the Plough, a tune well-known across the British Isles. Clearly these tunes are related but with distinct differences, particularly in the second and fourth bars.

It is typical in tunes like this that the emphasis is placed on the odd numbered notes, and in particular the first note of each beam. The strongest notes of the bar are thus 1 and 5, followed by 3 and 7.

To capture this emphasis when matching tune variants it might be possible to use some sort of similarity metric which weights stress (so that matching 1st notes carry more importance than, say, 2nd notes, e.g. [8]). However, an alternative approach is to build a multilevel (hierarchical) representation of the tunes.

Figs. 3 & 4 show multilevel coarsened versions of the original tunes, where the weakest notes are recursively replaced by removing them and extending the length of the previous note by doubling it.

At level 0, i.e. the original, the tunes are quantised to show every note as a sixteenth note, thus simplifying the coarsening process. In addition the triplet in bar 3 of "God Speed the Plough" is simplified by representing it as two eighth notes, the first and last notes of the triplet.

To generate level 1, the 2nd, 4th, 6th and 8th notes are removed from each bar. (Interestingly this accords with an idea used by Breathnach, [10], the renowned collector of Irish traditional music, who developed a system for indexing melodies based on the accented notes of each tune – effectively level 1 in the multilevel hierarchy).

For level 2, the original 3rd and 7th notes (which are now the 2nd and 4th) are removed; for level 3, the original 5th note (now the 2nd) is removed.

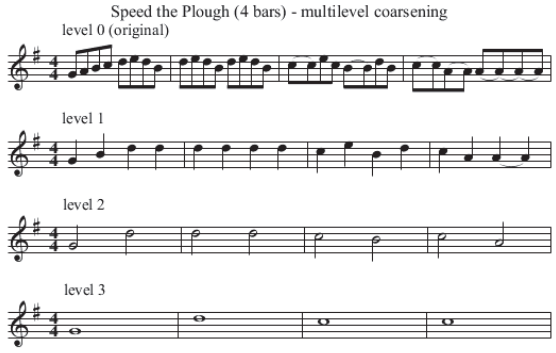

Figure 2. Multilevel coarsening of Speed the Plough

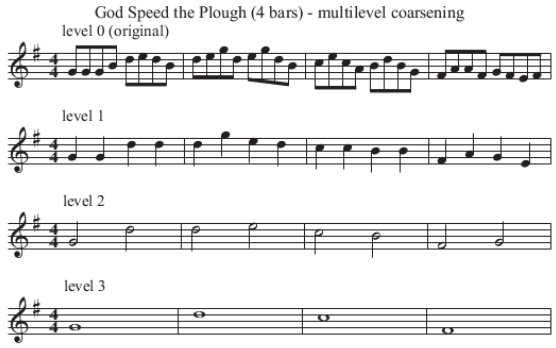

Figure 3. Multilevel coarsening of God Speed the Plough

As can be seen, as the coarsening progresses the two versions become increasingly similar and thus provide a good scope for melodic comparisons by ignoring the finer details of the tunes.

The implementation of this scheme is discussed more fully in [6] but is mostly straightforward. Each tune is initially normalised & quantised and then recursively coarsened down to a skeleton representation with just one note per bar. Melodic similarity calculations can then take place at every level and the (possibly weighted) sum of the similarities at each level used to provide a multilevel similarity measure (see section 4.5).

The coarsening works by recursively removing "weaker" notes from each tune to give increasingly sparse representations of the melody. In the current implementation the coarsening strategy considers that the weaker notes are the off-beats or every other note and it is these which are removed (see Figs. 3 & 4). However, it should be stressed that the multilevel framework is not tied to a particular coarsening strategy and in principle any algorithm that can reduce the detail in the melody (preferably recursively) could be used. For example, it should even be possible to use something as complex as a Schenkerian reduction, [11]; conversely multilevel algorithms in other fields often use randomised coarsening, [12].

Exceptions to the "remove every other note" rule are handled with heuristics, typically for tunes in compound time. Thus for jigs in 6/8, 9/8 & 12/8, which are normally written in triplets of eighth notes, the weakest notes are generally the second of each triplet. The same applies for waltzes, mazurkas and polskas in 3/4, so that for 3 quarter notes in a bar, the weakest is generally the second. The heuristics for dealing with these, and other less common time signatures, are discussed in [9].

Coarsening progresses until there is one note remaining in each bar; it would be possible to take it further, coarsening down to one single note for a tune, but experimentation suggests that the bar is a good place to stop. In fact some tunes in the dataset under investigation are in free meter, with no bar lines, and hence are coarsened down to a single note. An artificial limit of 4 levels (typical for many time signatures) was tested, but made very little difference to the results, particularly since any excess levels are ignored when comparing melodies with differing numbers of levels.

Once the multilevel representation is constructed, a variety of methods (including the alignment algorithms discussed here) can be used to compare each level. Again, this is a strength of the multilevel paradigm which is not reliant on a particular local search strategy, [12].

In the experiments below, multilevel variants are referred to as ML-*, where the * indicates the sub-sequence alignment algorithm (e.g. RLA). Conversely, if the multilevel representations are not used the similarity framework is referred to as SL-* (i.e. single level).

3. EVALUATION

3.1 The dataset

The dataset used is the Annotated Corpus of the Meertens Tune Collection, version 2.0.1, [4]. This contains 360 melodies each identified as belonging to one of 26 tune families. It also includes further annotations, splitting each melody into phrases, with three annotators manually assigning labels to each phrase. These have been used by Janssen *et al*., [1], [2], for testing search queries based on phrases, rather than whole melodies. However, since the investigation under discussion deals with globalised a2lignment algorithms it was decided to ignore the breakdown of the melodies into their constituent phrases.

3.2 Representation - transposition & time dilation

In [2], Janssen *et al*. present a comparison of several algorithms and indicate that what can make a big difference to the results is the representation of the music. In particular they find that using pitch adjustment in order to resolve transposition differences (i.e. similar melodies transcribed in different keys) can significantly improve the performance of the algorithms. However, the premise for their research is that the tune families are known in advance and so the pitch adjustment scheme uses this information and aims to transpose all the melodies in a given family into the same key. This does not apply for the work presented here where the aim is to classify each query melody into one of the 26 tune families, under the assumption that this is not known beforehand.

Perhaps a more appropriate scheme would be the pairwise pitch adjustment used by van Kranenburg *et al*., [3]. However, in the experimentation below, the algorithms are made transposition invariant by representing each melody as a sequence of pitch intervals. In contrast with Janssen *et al*., [2], this was not found to deteriorate

the algorithmic performance and it may be the case that their pitch adjustment scheme provides better results than intervals because it uses tune family information that would not normally be available for an arbitrary dataset.

Another interesting representational idea is the duration adjustment scheme, again from Janssen *et al*., [2], which seeks to adjust the note durations in a similar manner to pitch adjustment so that melodies transcribed in different meters (e.g. 3/4 and 6/8) are more closely comparable. However, that has not yet been tested with the algorithms described here.

In terms of the variants discussed by Janssen *et al*., the musical representation used in all experiments presented here is duration weighted pitch intervals, i.e. a representation which repeats each pitch according to the length of the note and with a sequence of integers expressing the difference in semitones between successive pitches.

3.3 Evaluation – Receiver Operating Curves (ROC)

The evaluation of each algorithmic variant is straightforward. Each of the 360 melodies is used as a query and compared against the other 359 melodies with the algorithm assigning a similarity score between the query and each melody. This results in 360 arrays, each containing 359 similarity scores.

Each array can then be used to generate a Receiver Operating Characteristic (ROC) curve which plots the true positive rate (TPR) against the false positive rate (FPR) in the ground truth as the results array is traversed. ROC curves are an elegant, generic tool often used to evaluate classification experiments across a wide range of disciplines, [13]. In fact they do not even require the similarity scores as input, they just need the results sorted in order of decreasing similarity and the ground truth (in this context whether a melody belongs to the same tune family as the query or not) to determine positive or negative outcomes.

Typically ROC curves are compared by measuring the Area Under the Curve (AUC). Since any ROC is confined to the unit square, the corresponding AUC is a value between 0.0 and 1.0 with higher values indicating a better classification algorithm. An AUC value of 1.0 indicates that the algorithm has done a perfect classification with all the true positives sorted by the similarity scores to one end of the array (and hence all the true negatives sorted to the other end). Conversely an AUC of 0.5 indicates that the algorithm has essentially done no better than a random classification.

Since each algorithmic variant results in 360 ROC curves, a method for combining them together is required. Janssen *et al*., [1], [2], aggregate all of the similarity results into one ROC curve and then measure the area underneath. However although this is a recognised technique, e.g. [13], it is not area-preserving in the sense that the average area under the individual curves is not necessarily the same as the area under the aggregated curve.

To see this, suppose an algorithm produces similarity scores of [1.00, 0.49, 0.00] for a particular search query and dataset of 3 melodies when the corresponding ground truth is [true, true, false] (in other words the melody with the similarity score of 0.00 is not a member of the same family as the search query, whereas the other two are). Since the similarity measure has done a perfect job of ordering the dataset using the similarity scores (perfect in the sense that all the true matches are at left hand end of the array and all the false matches at the other end), the ROC curve representing this would actually run up the x-axis and then along the line y = 1, giving the maximum possible AUC of 1.0.

Now suppose that a second search query produces similarity scores of [1.00, 1.00, 0.51] with the same corresponding ground truth of [true, true, false]. Once again the ordering is perfect and the area under the curve is 1.0. So the average AUC across the 2 queries is 1.0.

However, if the scores and ground truths are aggregated to form a single curve the results are no longer perfect as 0.49 is smaller than 0.51 and so the ordered ground truth array is [true, true, true, false, true, false]. The AUC for the corresponding ROC is 0.875.

Conversely, consider 2 search queries used on a dataset of 4 melodies and producing the results [1.00, 1.00, 0.52, 0.51] and [1.00, 1.00, 0.49, 0.48], both with corresponding ground truth of [true, true, false, true]. In this case the classifier has not done a perfect job and the AUC for each ROC is 0.667. When the results are aggregated the ordered ground truth array is [true, true, true, true, false, true, false, true] and the corresponding aggregated ROC has an AUC of 0.75.

Thus it is possible that the area under the aggregated curve can be significantly different (either lower or higher) from the average area under the individual curves.

Of course, as more results (more search queries, a larger dataset) are included, it is likely that the differences between the average AUC and the AUC for the aggregated ROC will diminish. Nonetheless, the aggregated ROC may not be telling the whole story.

In this paper the results for each algorithm are aggregated simply by taking an average of all the AUC values for that algorithm. Then, in order to draw the ROCs in Fig. 1, it is possible to use Fawcett's vertical averaging algorithm in [13] (Algorithm 3). Although Fawcett describes vertical averaging by sampling the ROC space at regular intervals, this is easily adapted to the non-parametric scheme described by Chen & Samuelson, [14], where it is sampled at every possible FPR value. With this adaptation in place, Chen & Samuelson have proven that the averaged ROC is area preserving, i.e. the AUC for the averaged ROC is the same as the average AUC across the individual ROCs (up to rounding differences).

3.4 Classification success rate (CSR)

Finally, to evaluate the quality of the tune classification into families, the nearest neighbour scheme described by van Kranenburg *et al*., [3], is applied. Specifically the melody in use567 as a query is assigned to the tune family of the nearest neighbour in terms of similarity. This assumes that the tune families of the 359 other melodies are known and that of the query is the unknown. However, for an arbitrary unannotated dataset, with no known tune families, it should be possible to use proximity graphs, similar to those described in [6] and with suitably chosen thresholds, to suggest tune family membership.

In the event that a number of melodies are nearest neighbours (i.e. have the same similarity with the query) then here ties are broken by considering the set of all such melodies and picking the tune family with the largest similarity contribution across the set.

Finally since, unlike van Kranenburg *et al*., [3], this experimentation is only applied to the small annotated dataset of 360 tunes, the classification success rate (CSR) can be calculated as a simple percentage |S|/360 where S is the set of tune family labels successfully identified.

4. EXPERIMENTATION

This section discusses the results: throughout the algorithms are applied cumulatively with the best perorming approach from each section used in the following section.

4.1 Baseline results

Table 1 shows the results for the baseline algorithms, single level LCSS & LA, showing the average AUC across all of the queries (which as mentioned above is the same as the AUC for the averaged ROC) and the classification success rate (CSR).

Algorithm	Variant	Avg AUC	CSR
SL-LCSS	baseline	**0.787**	0.697
SL-LA	baseline	**0.787**	0.814

Table 1. Results from the baseline algorithms, LCSS & LA (see section 2.1).

In this table (as all others) the best AUC figures for LCSS & LA variants are shown in boldface to highlight the key performance indicator.

What is perhaps surprising is that the LCSS algorithm appears to perform as well as the LA algorithm although it does a worse job of classification (69.7% correct as compared with 81.4%). However, the average AUC is 0.787 for both algorithms indicates a generally high quality similarity measure and, although the figures cannot be directly compared (for the reasons given in sections 3.1, 3.2 & 3.3), is broadly comparable to the 0.790 figure for LA in [1].

It should not be a surprise that there is such a wide difference in the Classification Success Rate (CSR). In fact CSR is not such a good performance indicator as AUC, since essentially it only applies to the nearest neighbour (highest similarity) for each query, whereas the AUC measures the performance of the similarity measure across the entire dataset. Thus, as well as indicating the similarity with all other melodies in the tune family, the AUC is a better indicator of how the algorithm might perform for other melodic similarity tasks, such as search and matching.

4.2 Length normalisation

Table 2 shows the effect of applying different length normalisations to the similarity measure – i.e. when comparing two tunes of different lengths, dividing the raw similarity score by the minimum, the average and the maximum length of the two tunes (of course, if the tunes are the same length then these three values are the same).

As can be seen, this can make a small improvement to the results, with minimum length giving the worst results and average length the best. Surprisingly here, LCSS even outperforms LA.

Algorithm	Length	Avg AUC	CSR
SL-LCSS	Min	0.787	0.697
SL-LA	Min	0.787	0.814
SL-LCSS	Avg	**0.818**	0.853
SL-LA	Avg	**0.810**	0.872
SL-LCSS	max	**0.818**	0.872
SL-LA	max	0.802	0.883

Table 2. Results showing the effects of different length normalisation (see section 2.2).

From here on all results use average length as the chosen normalisation, unless otherwise indicated

4.3 Bar indicators

Table 3 shows the effect of including bar indicators in the representation. As can be seen, bar markers and even bar numbers can improve some results significantly, with bar numbers being somewhat less effective. This is perhaps to be expected; bar numbers tie the bars down to a particular part of the melody whereas in fact there are known instances where the ordering of the phrases may change.

Algorithm	Bar indicators	Avg AUC	CSR
SL-LCSS	none	0.818	0.853
SL-LA	none	0.810	0.872
SL-LCSS	markers	**0.827**	0.853
SL-LA	markers	**0.849**	0.911
SL-LCSS	numbers	0.814	0.853
SL-LA	numbers	0.846	0.906

Table 3. Results showing the effects of using bar indicators (see section 2.3).

Furthermore, with bar markers and average length normalisation LA is now seen to give better results than LCSS. Again this is to be expected since it is a more sophisticated (though computationally costly) algorithm.

4.4 Recursive sub-sequence alignment

Table 4 shows the effect of including recursive variants of the sub-sequence alignment algorithms, i.e. Recursive Local Alignment (RLA) and Recursive Longest Common SubString (RLCSS). As can be seen, the crucial feature is the use of biased similarity (biased to favour longer matches, rather than a series of short matches) which uses the 2-norm of the recursive similarity scores (section 2.4.2); just adding the recursive similarity scores together (1-norm) actually makes the recursive results worse than the non-recursive versions.

Algorithm	Recursive score	Avg AUC	CSR
SL-LCSS	none	0.827	0.853
SL-LA	none	0.849	0.911
SL-RLCSS	1-norm	0.813	0.828
SL-RLA	1-norm	0.842	0.878
SL-RLCSS	2-norm	**0.845**	0.889
SL-RLA	2-norm	**0.854**	0.914

Table 4. Results showing the effects of using recursive sub-sequence alignment (see section 2.4).

4.5 Multilevel similarity

Table 5 presents perhaps the biggest performance enhancement which comes from the multilevel similarity measure, adding all the similarity scores from all coarsened versions of the melody. This significantly improves on the single level versions, SL-RLCSS and SL-RLA.

Algorithm	Framework	Avg AUC	CSR
SL-RLCSS	single level	0.845	0.889
SL-RLA	single level	0.854	0.914
ML-RLCSS	multilevel	**0.870**	0.900
ML-RLA	multilevel	**0.887**	0.922

Table 5. Results showing the effects of using multilevel similarity (see section 2.5).

Note that other experiments were performed to vary the weight of the similarity contributions from each level (e.g. as suggested in [5], giving greater weight to the finer, more accurate representations of the melody). However, none of the variants gave consistently better results.

4.6 Parameter cross-checking

Finally Tables 6 & 7 provide some cross-checks to further validate the results above. Table 6 shows the results for the multilevel schemes, using bar markers but with different length normalisations (minimum, average & maximum). As in section 4.2, the average length is seen to give the best normalisation.

Algorithm	Length	Avg AUC	CSR
ML-RLCSS	Min	0.840	0.811
ML-RLA	Min	0.865	0.872
ML-RLCSS	Avg	**0.870**	0.900
ML-RLA	Avg	**0.887**	0.922
ML-RLCSS	max	0.866	0.900
ML-RLA	max	0.878	0.917

Table 6. Results showing the effects of different length normalisation for the multilevel algorithms.

Meanwhile Table 7 shows the results using average length normalisation, but comparing bar indicators (no indicators, bar markers & bar numbers). As in section 4.3, bar markers are seen to give the best results.

Algorithm	Indicators	Avg AUC	CSR
ML-RLCSS	none	0.880	0.908
ML-RLA	none	0.883	0.922
ML-RLCSS	bar markers	**0.870**	0.900
ML-RLA	bar markers	**0.887**	0.922
ML-RLCSS	bar numbers	0.849	0.883
ML-RLA	bar numbers	0.868	0.925

Table 7. Results showing the effects of using bar indicators for the multilevel algorithms.

4.7 Discussion

Fig. 1 shows the ROC curves for 4 of the algorithmic variants, the two baseline algorithms (SL-LCSS & SL-LA) and the two final algorithms with all four enhancements (ML-RLCSS & ML-RLA using average length normalisation and bar markers). As can be seen the baseline algorithms have very similar curves with SL-LCSS marginally worse than SL-LA for smaller values of TPR / FPR (i.e. with high similarities) and marginally better at the other end of the range. Of the final algorithms, ML-RLA is better than ML-RLCSS although their performance is almost indistinguishable for larger values of TPR / FPR.

Note also that although LA versions of the algorithms generally outperform LCSS, the LCSS results are of interest because they achieve nearly the same quality and are much faster (e.g. in the tests presented here LCSS variants are about 1.4 to 1.6 times faster than the LA counterparts).

As mentioned before, the results here are not directly comparable with those of Janssen *et al*., [1], [2] (for the reasons given in sections 3.1, 3.2 & 3.3). Nonetheless they are of the same order and for example the best AUC value presented here (0.887), even without knowledge of the tune families, is broadly comparable to the best value in [2] (0.893, achieved using a hand-adjusted representation). It is even possible that by combining some of the techniques (e.g. the duration adjustment scheme, [2]) the results could be improved still further.

The results are more directly comparable with those of van Kranenburg *et al*., [3]. Unfortunately the best CSR of 0.925 presented here does not match the CSR of 0.99 achieved there and again this argues for further integration of techniques.

5. CONCLUSIONS

This paper has investigated several enhancements to two well-established sub-sequence alignment algorithms, in the context of their use for melodic similarity and in particular classification of queries into tune families. It uses the annotated dataset from the well-known Meertens Tune Collection to provide the ground truth with which to evaluate the quality of the algorithms.

In particular, recursive application of the alignment algorithms applied to a multilevel representation of the melodies is shown to be very effective for improving the accuracy of the classification.

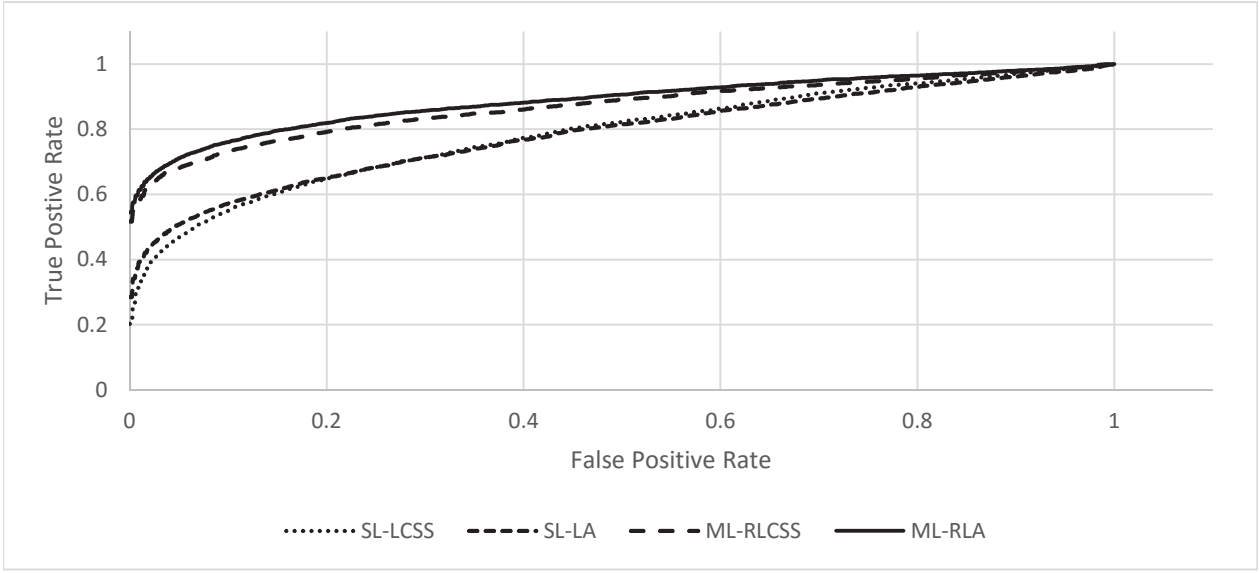

Figure 4. The averaged ROC curves for four algorithmic variants: the baseline algorithms and the best variants.

The other enhancements include length normalisation of the similarity measure (which can be tailored according to the problem – for example minimum length might be more appropriate where all the queries are expected to be short phrases but the dataset contains complete melodies). In addition, the use of bar indicators can improve the results still further.

In broad terms, the impact of these enhancements (along with other representational variants, e.g. [2]) suggest that sub-sequence alignment (both the local alignment version, LA, and the special case, LCSS) are flexible and robust in terms of how the music is represented and how the algorithms are applied.

Finally it should be stressed that these enhancements do not appear to be mutually dependent. In other words, it should be possible for other authors to adopt some or all of the algorithmic enhancements discussed here to improve melodic similarity algorithm(s), and the ideas should be equally applicable to music search and melodic matching.

6. REFERENCES

[1] B. Janssen, P. van Kranenburg, and A. Volk, "A Comparison of Symbolic Similarity Measures for Finding Occurrences of Melodic Segments," in *Proc. ISMIR*, 2015, pp. 659–665.

[2] B. Janssen, P. van Kranenburg, and A. Volk, "Finding occurrences of melodic segments in folk songs employing symbolic similarity measures," *J. New Music Res.*, p. (to appear), 2017.

[3] P. van Kranenburg, A. Volk, and F. Wiering, "A Comparison between Global and Local Features for Computational Classification of Folk Song Melodies," *J. New Music Res.*, vol. 42, no. 1, pp. 1–18, 2013.

[4] P. van Kranenburg, B. Janssen, and A. Volk, "The Meertens Tune Collections : The Annotated Corpus (MTC-ANN) Versions 1.1 and 2.0.1," 2016.

[5] C. Walshaw, "Multilevel Melodic Matching," in *5th Intl Workshop on Folk Music Analysis*, 2015, pp. 130–137.

[6] C. Walshaw, "Constructing Proximity Graphs To Explore Similarities in Large-Scale Melodic Datasets," in *6th Intl Workshop on Folk Music Analysis*, 2016.

[7] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Mol. Biol.*, vol. 147, pp. 195–197, 1981.

[8] R. Typke, "Music Retrieval based on Melodic Similarity," Utrecht University, Netherlands, 2007.

[9] C. Walshaw, "TuneGraph: an online visual tool for exploring melodic similarity," in *Proc. Digital Research in the Humanities and Arts*, 2015, pp. 55–64.

[10] B. Breathnach, "Between the Jigs and the Reels," *Ceol*, vol. V, no. 2, pp. 43–38, 1982.

[11] A. Marsden, "Schenkerian Analysis by Computer: A Proof of Concept," *J. New Music Res.*, vol. 39, no. 3, pp. 269–289, 2010.

[12] C. Walshaw, "Multilevel Refinement for Combinatorial Optimisation: Boosting Metaheuristic Performance," in *Hybrid Metaheuristics - An emergent approach for optimization*, C. Blum, Ed. Springer, Berlin, 2008, pp. 261–289.

[13] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, pp. 861–874, 2006.

[14] W. Chen and F. W. Samuelson, "The average receiver operating characteristic curve in multireader multicase imaging studies," *Br. J. Radiol.*, vol. 87, no. 1040, pp. 1–8, 2014.