Tree structured and combined methods for comparing metered polyphonic music

David Rizo[1], Kjell Lemström[2], and José M. Iñesta[1]

[1] Dept. Lenguajes y Sistemas Informáticos, Universidad de Alicante,
E-03080 Alicante, Spain
{drizo,inesta}@dlsi.ua.es
[2] Dept. of Computer Science
University of Helsinki
FIN-00014 Helsinki, Finland
klemstro@cs.helsinki.fi

Abstract. Identifying copies or different versions of a same musical work is a focal problem in maintaining large music databases. In this paper we introduce novel ideas and methods that are applicable to metered, symbolically encoded polyphonic music. We show how to represent and compare polyphonic music using a tree structure. Moreover, we put for trial various comparison methods and observe whether better comparison results can be obtained by combining distinct similarity measures. Our experiments show that the proposed representation is adequate for the task with good quality results and processing times, and when combined with other methods it becomes more robust against various types of music.

1 Introduction

The recent dramatic increase in the number of music databases available in the Internet has made the automatic music comparison/retrieval systems attractable, not only to researchers working in the area, but also to music consumers downloading midi files or new ringing tones, and organising personal music databases.

In this paper we consider and develop methods comparing symbolically encoded (e.g. MIDI) musical works. A central problem in music information retrieval is to recognise copies/versions of a same musical work, although the versions may considerably differ from each other. A relevant, intrinsic feature of music is that music presented in different keys (i.e., *transposed* in higher or lower pitch) are perceived by human listeners as the same work. This phenomenon also applies to differences in tempo.

In the literature, several methods have been developed for comparing monophonic [3] musical works (see e.g., [11, 15, 13]). One possibility for comparing two

[3] In monophonic music only one note is played at any time, while in polyphonic music there are simultaneous notes.

polyphonic works would be to use a monophonic reduction schema, such as the skyline algorithm [20]. However, even though the two versions to be compared may represent the same original work, because of differences in the accompaniments and harmonisations the reduction may produce two totally different monophonic melodies in which resemblance cannot be found.

If the musical works in hand are not allowed to be transposed and, therefore, the comparison algorithm does not have to be transposition invariant, one can apply a string representation based algorithm for the problem [3, 6]. If transposition invariance is required, most of the string-based methods fail because of the combinatorial explosion in the number of possible strings to be taken into account. Doraisamy and Rüger [5] avoids the worst explosion by chopping the possible strings in n-grams. Recently, various algorithms based on geometric representation of music [9] which are capable of finding occurrences of both a monophonic and a polyphonic pattern within a polyphonic musical work have been introduced [4, 22, 21, 10].

A tree structured comparison method was introduced in [15] for the corresponding monophonic task. Transposition and tempo invariances are obtained by conducting a preprocessing phase that finds the rhythm structure and the tonic of the music in hand [14]. In this paper we will elaborate this approach further and introduce novel tree based methods for the polyphonic task. Moreover, as it is well-known that by combining classifiers one can often achieve better results in accuracy and robustness (when compared to the performance of the individual classifiers; see e.g., Moreno-Seco et al. [12]). We have also experimented on whether one can achieve better comparison results when combining our novel methods with some existing methods.

2 Tree representations

2.1 Tree representation for monodies

A melody has two main dimensions: time (duration) and pitch. In linear representations, both pitches and note durations are coded by explicit symbols, but trees are able to implicitly represent time in their structure, making use of the fact that note durations are multiples of basic time units, mainly in a binary (sometimes ternary) subdivision. This way, trees are less sensitive to the codes used to represent melodies, since only pitch codes are needed to be established and thus there are less degrees of freedom for coding.

Duration in western music notation is designed according to a logarithmic scale: a *whole* note lasts twice than a *half* note, that is two times longer than a *quarter* note, etc. (see Fig. 1). The time dimension of music is divided into *beats*, and consecutive beats into bars.

In our tree model, each melody bar is represented by a tree, τ. Each note or rest resides in a leaf node. The left to right ordering of the leaves preserves the time order of the notes in the melody. The level of a leaf in the tree determines the duration of the note it represents, as displayed in Fig. 1: the root (level

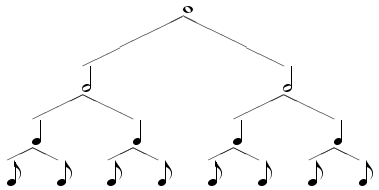

Fig. 1. Duration hierarchy for note figures. From top to bottom: whole (4 beats), half (2 beats), quarter (1 beat), and eighth (1/2 beat) notes.

1) represents duration of the whole bar (a *whole* note), the two nodes in level 2 duration of a *half* note. In general, nodes in level i represent duration of a $1/2^{i-1}$ of a bar.

During the tree construction, internal nodes are created on demand to reach the appropriate leaf level. Initially, only the leaf nodes contain a label value. Once the tree is built, a bottom-up propagation of these labels is performed to fully label all the nodes. The rules for this propagation are described below.

The tree labels represent the corresponding pitch information. In order to have a transposition invariant representation, in this paper we use the interval from the main key of the song obtained using the algorithm introduced in [14]. This way, the labels of the tree use the alphabet {0..11} corresponding to pitch classes relative to the tonality. This way, in 'G Major', pitch class 'G' is mapped to 0. Nodes representing rests have an empty label.

An example of this schema is presented in Fig. 2. In the tree, the left child of the root has been split into two subtrees to reach level 3 that corresponds to the first note duration (as eighth note lasts $1/2^2$ of the bar, pitch B coded as **11**). In order to represent the durations of the rest and note G, coded as **7** (both last 1/8 of the bar), a new subtree is needed for the right child in level 3, providing two new leaves for representing the rest (empty label) and the note G (**7**). The quarter note C (**0**) onsets at the third beat of the bar, so it is represented in level 2 according to its duration.

Fig. 2 depicts how the time order of the notes in the score is preserved by traversing the tree from left to right. Note also how onset times and durations are implicitly represented in the tree, compared to the explicit encoding of time when using strings. This representation is invariant under time scalings, as for instance, different meter representations of the same melody (e.g. 2/2, 4/4, or 8/8).

Processing non binary durations. In some occasions the situation can be more complicated. There are note durations that do not match a binary division of the whole bar. This happens, for example, for dotted notes (duration is extended in an additional 50%) or tied notes whose durations are summed. (see Fig. 3). In such a case, a note cannot be represented just by one leaf in the proposed schema. However, it is well-known [11] that our auditory system perceives

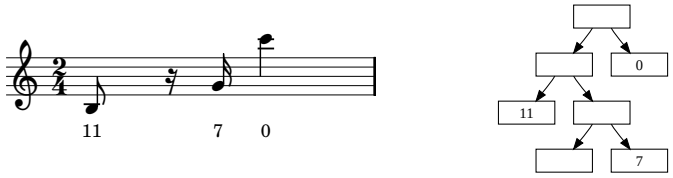

Fig. 2. Simple example of tree construction.

a note of a given duration the same way as two notes of the same pitch, played one after the other, whose durations sum to that of the single one. Therefore, when a note exceeds the considered duration, in terms of binary divisions of time, it is subdivided into notes of binary durations, and the resulting notes are coded in their proper tree levels. Fig. 3 depicts such an example and how it is handled by the schema.

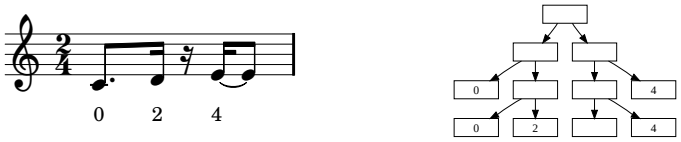

Fig. 3. Tree representation of notes exceeding their notation duration: dotted and tied notes. Both 0 leaves correspond to the same dotted quarter note. The two 4 leaves represent the two tied notes.

Other frequently used non binary divisions are ternary rhythms. In that case, the length of one bar is usually 3 beats and it is split into 3 quarter notes, etc. This is not a problem, since neither the tree construction method nor the metrics used to compare trees need to be binary; there can be any arbitrary number of children for a node. So, in ternary meters or ternary divisions, the number of children for a node is three. This can be generalized to other more complicated cases that can appear in musical notations, like tuplets or compound meters. Fig. 4 gives an example of compound meter based on ternary divisions and the corresponding tree.

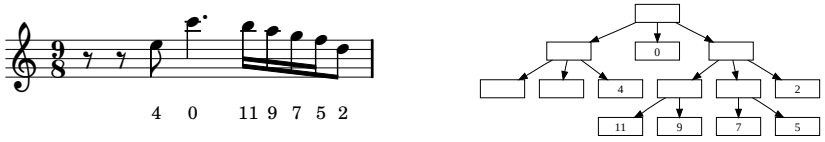

Fig. 4. The meter 9/8 is a compound one based on ternary divisions. The tree construction method can also represent this melody.

There are other subtle situations that may appear in a score, like for example grace notes[4], that are not included in the cases described above. Nevertheless, in digital scores (e.g. MIDI files) these special notes are represented by short notes that are subsequently coded in the level corresponding to their written duration by our schema.

Representation of complete melodies. The method described above is able to represent a single bar as a tree, τ. A *bar* (or a measure) is the basic unit of rhythm in music, but a melody is composed of a series of M bars. Let us now describe how to combine the set of trees $\{\tau_i\}_{i=1}^{M}$ representing the bars.

To build a tree, T, for a complete melody, the computed bar trees are joined in a particular order. For instance, the sub-trees can be grouped two by two, using always adjacent pairs. This operation is repeated hierarchically, bottom-up, with the new nodes until a single tree is obtained. Let us denote the depth (or height) of a tree T by $h(T)$. With this grouping method, the trees grow in depth quickly:

$$h(T) = \log_2 M + 1 + \max_i \, h(\tau_i),$$

making the tree edit distance computation very time consuming, as discussed in Section 4. The best choice is to build a tree with a root for the whole melody, whose children are the bar sub-trees. This way, the depth of a tree corresponding to a whole melody becomes

$$h(T) = 1 + \max_i \, h(\tau_i).$$

The smaller depth of the tree of the latter approach makes it the choice to be taken. Fig. 5 (to the left) displays an example of a simple melody, composed of three bars, and how it is represented by a tree composed of three sub-trees.

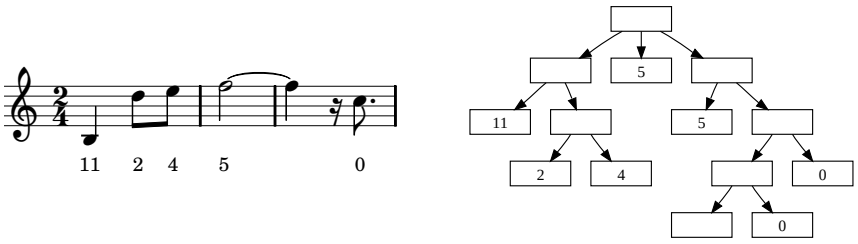

Fig. 5. Melody and the corresponding tree. The root of the tree connects all the bar sub-trees.

[4] A grace note is a very short note or a series or notes to achieve musical effects that occupies no time in the duration notation in a score. They are also known as *acciaccatura*.

Tree representation of polyphonic music. To represent polyphonic music all voices are inserted in the same tree following the rules of the monophonic music representation. Node labels now represent sets of pitch classes. Under this approach, each leaf will contain all the notes played at a given time (whose depth is conditioned by the shortest one). A node representing only rests has an empty set as the label.

Fig. 6 contains a polyphonic example and its tree representation.

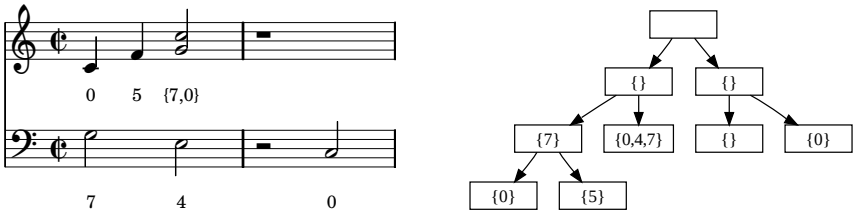

Fig. 6. An example of a polyphonic music and the corresponding tree. Note that in polyphonic trees empty labels are explicitly represented by the empty set.

Bottom-up propagation of labels. Once the tree is constructed, a label propagation step is performed. The propagation process is performed recursively in a post-order traversal of the tree. Labels are propagated using set algebra. Let $L(\tau)$ be the label of the root node of the subtree τ expressed as a set of pitch classes. When the label of the node is a rest, the label set is empty: $L(\tau) = \varnothing$. Then, given a subtree τ with children c_i, the upwards propagation of labels is performed as $L(\tau) = \bigcup_i L(c_i)$. The upwards propagation goes until level two, that is, the root representing the whole piece of music always remains empty. Fig. 7 shows the tree in Fig. 6 after propagating its labels.

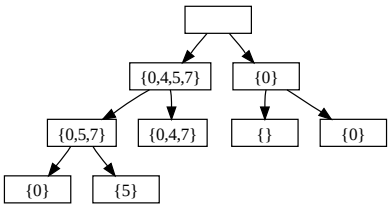

Fig. 7. Tree of Fig. 6 after bottom-up label propagation.

In Fig. 7, the half note C (pitch class 0) in the second bar, which shared a parent with the rest, is promoted ($\varnothing \cup \{0\} = \{0\}$). In the first bar, the node

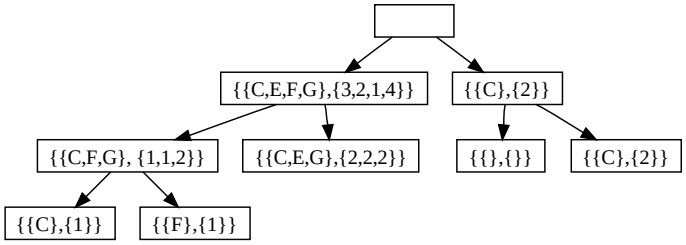

Fig. 8. Multiset label version of the tree in Fig. 7. Pitch names are used instead of pitch classes to avoid confusion with cardinalities. The labels contain multisets (B, f).

containing label $\{0, 5, 7\}$ contained only the quarter note F (pitch class 5) before propagation. The propagation operation merges all pitches in that branch ($\{0\} \cup \{5\} \cup \{7\} = \{0, 5, 7\}$).

Multiset labels. The current polyphonic representation may have a drawback after the propagation step: if the lower levels of the tree contain scales covering a whole octave, the sets of propagated inner nodes would contain all the pitch classes. This way, the inner nodes representing two distinct musical works would be the same and the comparison methods would always consider the two similar to each other.

To overcome this problem the set label is replaced with a multiset, where longer notes have higher cardinality than short ones, i.e., giving lower importance to those pitch classes propagated from deeper levels of the tree.

A *multiset* (aka. a bag) is a pair (X, f), where X is an ordered set, and f is a function mapping $f : X \to \mathbb{N}$. For any $x \in X$, $f(x)$ is called the *multiplicity* of x. Using this definition and expressing f as an ordered set, we see that the multiset $\{1, 1, 3\} = (\{1, 3\}, \{2, 1\})$ meaning that $f(1) = 2$ and $f(3) = 1$.

Now, all the node labels in a tree are represented by an assigned multiset (B, f). Once again, we start from the leaves by setting:

$$B = \{p \mid 0 \leq p \leq 11\}, \tag{1}$$

$$f(p) = \begin{cases} 2^{h-l}, & \text{if p} \in L(\tau) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where l gives the level of the node in the tree. Then the propagation is performed analogously to that of above, using the multiset union operation instead of that of sets.

Fig. 8 illustrates this representation. It can be noticed that the note 'F' has a low weight at root level as compared to the other longer notes.

Prunning. Due to the presence of very short notes that eventually will have a low weight in the final node labels, the pruning of the trees has been considered thus making trees smaller and tree edit algorithms faster.

3 Geometric algorithms for polyphonic pattern matching

Clausen et al. [4] used inverted file indices with the geometric representation. The onset times of the notes are quantized to a pre-selected resolution so that both the pitch and time dimensions are discrete. Moreover, onset times are represented relatively to their metrical position within the measure. The information within a measure constitutes one unit for the query. The retrieval method finds occurrences (total and partial) that have a similar metrical positions as the query; local time and pitch fluctuations cannot be dealt with. Tempo invariance can be obtained by conducting a metrical structure analysis phase, transposition invariance by using a mathematical trick that outperforms the brute-force solution.

Wiggins et al. [22] suggested to use the piano-roll representation and work on the translation vectors between the points in the piano-rolls. In this way, finding a transposition invariant exact occurrence of a query pattern becomes straightforward: find a translation vector that translates each point in the pattern to some point within the studied musical work. Ukkonen et al. [21] showed how to process the translation vectors in a suitable order to obtain an algorithm of a linear expected and a quadratic worst case time complexity (algorithm P1). Their method is also modified to the case of finding all partial occurrences of the pattern. This is doable in $O(nm \log m)$ time, where n and m refer to the length of the musical work and the pattern, respectively (algorithm P2). Moreover, they also suggested to replace the points in the piano-rolls by horizontal line segments (thus considering the length of the notes instead of the bare note-on information) and modified their algorithm to solve the problem of *the longest common total length*. For this problem, their P3 algorithm requires some extra data structures but runs also in $O(nm \log m)$ time (with discrete input). This setting takes into account tempo fluctuations but not pitch fluctuations. One attempt in this direction was presented by Lubiw and Tanur [10]. More recently, Lemström et al. [8] have developed more efficient versions of the P2 algorithm using indexing. In our experiments we have used one of their novel algorithms called P2v6.

4 Tree comparison methods

The edit distance between two trees is defined accordingly to the string edit distance: it is the minimum cost sequence of all possible sequences of operations that transforms one tree into another [18]. The standard editing operations are deletion, insertion, and substitution of a node label. Thus, in the straightforward case where the operations are assigned with unit costs, the distance of the trees is obtained by counting the number of required operations.

Shasha & Zhang tree edit distance. As the starting point, we use Shasha and Zhang's method to compute the edit distance between two trees, T_A and T_B. Their method runs in time $O(|T_A| \times |T_B| \times h(T_A) \times h(T_B))$, where $|T_i|$ denotes

the number of nodes in tree T_i and $h(T_i)$ its depth. Although, it is allowable to assign individual costs to the editing operations, in our experiments we obtained best results with unit costs.

Selkow tree edit distance. Because of the rather high time complexity of the Shasha and Zhang's method we wanted to experiment also with an alternative method introduced by Selkow [17]. The main functional difference is that node insertions and deletions can be done only at the level of the leaves. If an inner node needs to be deleted, all the subtrees rooted by it have to be deleted first. Naturally, these restrictions make the algorithm simpler but less accurate.

Having joined all the bar sub-trees in the root in the construction method, the Selkow method runs in time $O(n_A n_B h)$ where n_A, n_B and h are the maximum arities of the trees T_A and T_B, and their maximum depth, respectively.

Multiset substitution cost. For computing the distances, some approaches like the Euclidean distance between vectors has been tested. Finally, the distance that performed the best has been a bounded Manhattan distance.

Let $\mathcal{M} = (B, f)$ be a multiset that corresponds to a node label. We represent it by using a vector $\mathbf{v}_{\mathcal{M}} \in \mathbb{N}^{12}$, such that $\mathbf{v}_{\mathcal{M}}[p] = f(p), \forall p \in B$ according to definition in eq. (2). Then, the substitution cost c_{sbn} between two multisets $\mathcal{M}_a = (B_a, f_a)$ and $\mathcal{M}_b = (B_b, f_b)$ is defined as the following distance between the corresponding vectors:

$$c_{sbn}(\mathcal{M}_a, \mathcal{M}_b) \triangleq d_{eq}(\mathbf{v}_{\mathcal{M}_a}, \mathbf{v}_{\mathcal{M}_b}) = \frac{\sum_{p=0}^{11} min(1, (\mathbf{v}_{\mathcal{M}_a}[p] - \mathbf{v}_{\mathcal{M}_b}[p])^2)}{12}. \quad (3)$$

Note that the maximum difference between two components of the multiset has been limited to 1. Unit cost has been assigned to insertion and deletion operations.

Roots' edit distance. As the nodes in level two represent all the bars, an overview of the whole musical work can be obtained by only observing this level. In this way we compute the roots' edit distance.

To this end, let T be a tree representing a polyphonic musical work of M bars, and $\{T_2^k\}_{k=1}^{M}$ the siblings in level one of the tree rooted by T. The label function $l(T_2^k)$ returns the vector $\mathbf{v}_{\mathcal{M}}$ corresponding to the node T_2^k. The roots' edit distance $\textsc{RootED}(T, T')$ between two trees T and T' is the unit cost string edit distance between strings $S(T)$ and $S(T')$, such that for a tree τ, $S(\tau) \in (\mathbb{N}^{12})^*$, is constructed using the sequence of labels of $\{\tau_2^i\}$: $S(\tau) = l(\tau_2^1), l(\tau_2^2), \ldots, l(\tau_2^M)$.

The algorithm is not dependent on the depth of the tree, as the Selkow tree distance; it works in time $O(M_T M_{T'})$, where M_T and $M_{T'}$ are the number of bars of the musical works represented by trees T and T', respectively.

Longest common root subsequence. Accordingly to that of previous section, we also apply the classical longest common subsequence (LCS) measure [2] to the strings obtained from level one of the trees to be compared. We call this similarity measure the longest common root subsequence, or LCRS for short.

For the LCS computation we need a function that checks the equality of given two symbols that in our case are multisets. To this end, let $\mathcal{M}_a$ and $\mathcal{M}_b$ be multisets as detailed above. For an item-wise similarity of $\mathcal{M}_a$ and $\mathcal{M}_b$, denoted $\mathcal{M}_a \doteq \mathcal{M}_b$, we require:

$$\mathcal{M}_a \doteq \mathcal{M}_b \;\Leftrightarrow\; \forall_{p=0}^{11}\{\mathbf{v}_{\mathcal{M}_a}[p] = \mathbf{v}_{\mathcal{M}_b}[p]\}.$$

To solve the problem, we have applied a classical dynamic programming algorithm that runs in time $O(M_a M_b)$, where M_a and M_b represent the number of bars in the corresponding musical works. We are considering to improve the performance of this process by using bit-parallelism thus obtaining a speedup of a factor w denoting the size of the computer word in bits [7].

5 Classifier combination

The similarity results for different similarity models may differ substantially from each other. As exemplified by Moreno-Seco et al. [12], this feature can be exploited: by combining classifiers a better result is often obtained than when applying the same classifiers (similarity measures) individually. In this paper, we have tested some straightforward combinations. The combinations are always built in the same way: they accumulate the normalised similarity values from the included individual classifiers. The normalisation is necessary because of the distinct result spaces of the individual measures.

Given a collection G^N of N musical works, let $d_\alpha(G_x, G_y)$ be the similarity value between two musical works G_x and G_y ($1 \leq x, y \leq N$) using the algorithm α. The normalized distance $\tilde{d}_\alpha$ is computed as:

$$\tilde{d}_\alpha(G_x, G_y) = \frac{d_\alpha(G_x, G_y) - \min\{d_\alpha(G_i, G_j)\}}{\max\{d_\alpha(G_i, G_j)\} - \min\{d_\alpha(G_i, G_j)\}},$$

where $1 \leq i, j \leq N$ and $i \neq j$. Finally, the combination $\mathcal{C}$ of the similarity measures of a set of α^m algorithms ($m \in \mathbb{N}$) is performed as follows:

$$\mathcal{C}(\alpha^m, G_x, G_y) = \sum_{a=1}^{m} \tilde{d}_{\alpha_a}(G_x, G_y)$$

6 Experiments

In order to evaluate the algorithms, two corpora with different styles of polyphonic music has been built. The first one, called *ICPS*, has 68 MIDI files corresponding to covers of the incipits of seven musical works: Schubert's Ave Maria,

Ravel's Bolero, the children songs Alouette, Happy Birthday and Frère Jacques, the Carol Jingle Bells and the jazz standard When The Saints Go Marching In. All the works in this corpus have a similar kind of accompaniment tracks.

The second corpus, called *VAR*, consists of 78 classical works representing variations of 17 different themes as written by the composer: Tchaikovsky variations on a rococo theme op.33, Bach English suites BWV 806-808 (suite 1 courante II, suite 2, 3, and 6 sarabande), and Bach Goldberg variations.

To evaluate the methods, we have used the *leave-one-out*, *all-against-all* schema: for each work in the corpus we compute the similarity value against all the others providing 5281 comparisons in the whole experiment.

Four combinations of the individual similarity measures have been built. (see Figs. 9–12). They are named according to the measures they include: 'CombALL' includes all the measures, 'CombGeom' consists of all the geometric algorithms, 'CombTree' use all classifiers involving trees, and 'CombFast' include the methods G-P2v6, Selkow with trees pruned at level 2, RootED, and Lcrs.

In all the cases, given an algorithm and a corpus, the reported results are averages calculated from a corresponding trial. The reported execution times exclude all preprocessing. This is because preprocessings are executed only once, not when the actual comparison is carried out.

In order to be able to measure the accuracy of the algorithms, they return a list containing the candidates in a decreasing order in the similarity score. The observed accuracy measures are the following: The *top-n recognition rate* (denoted by TRR_n) indicates the presence percentage of the correct answer among the top n slots within the list in a trial. Finally, the *precision-at-n* is computed as the number of relevant hits among the first n elements in the result list.

In the plots, 'Precision at $|class|$' reveals the *precision-at-n*, and it is computed as the weighted average of the *precision-at-n* for each class, taking for n the number of prototypes for each class, or number of versions of each song.

6.1 Results

Figs.9 and 10 show the results for the corpus *ICPS* and *VAR*, respectively. The horizontal axis gives the considered algorithms: 'G-P2' and 'G-P3' stand for the geometric algorithms $P2$ and $P3$ (recall Section 3), the 'G-P2v6' the new version of G-P2. The notation for the tree algorithms have been appended with a suffix; for instance, an 'Lh' means that the trees have been pruned at level h before computing the distances. Recall the names for the combinations from the previous subsection.

The accuracy of the geometric methods G-P2 and G-P2v6 is high with the classical themes (Fig. 10) but seems to get confused by the accompaniments in the *ICPS* corpus. The tree methods behave in a complementary way, performing the best with *ICPS* and worse with the *VAR* corpus. One possible explanation for this is that our multiset approach may not suffice in a case of many distinct notes (or more specifically, pitch classes) appearing in a musical bar. The best running times are obtained with Lcrs and RootED, but with the cost of accuracy. The best tradeoff between time and precision seems to be obtained by the Selkow edit

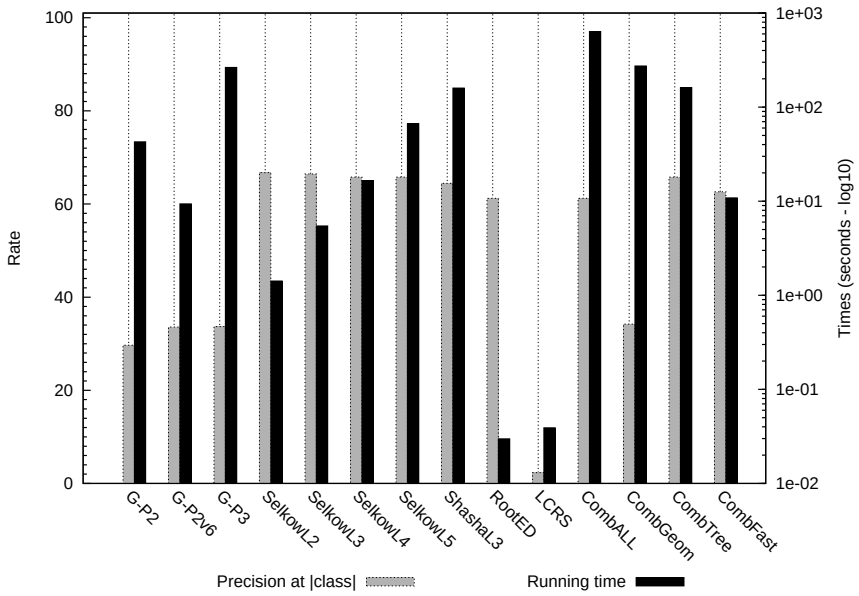

Fig. 9. Times and accuracy for corpus *ICPS*.

distance working with only two levels. The main difference between RootED and Selkow with two level trees, is that Selkow is able to identify the same song coded with two different meters, e.g. a 4/4 vs. two 2/4.

The plots in Figs. 11 and 12 depict the top-n recognition rates for the algorithms as a function of n. Only the best method of each family (geometric, trees, combination) is shown. A high recognition rate at a low n would be ideal. TRR_1 gives the percentage of times the returned prototype belongs to the same class as the query. Recall that the geometric methods used here were not developed for comparing whole musical works, but for musical pattern matching.

In terms of accuracy over both corpora, the combined method gives the best result. This is due to the complementary effect of joining the geometric and tree measures.

7 Conclusion

We have introduced a new paradigm to represent and compare polyphonic musical works. In our experiments, the method obtained promising results with two corpora, being comparable in accuracy to those obtained with geometric methods and faster in processing time. The simple combination of both paradigms has lead to overall better precisions behaving well with both corpora.

We carried out further experiments with monophonic corpora and with pop-rock MIDI files downloaded from the Internet. In both cases the combined

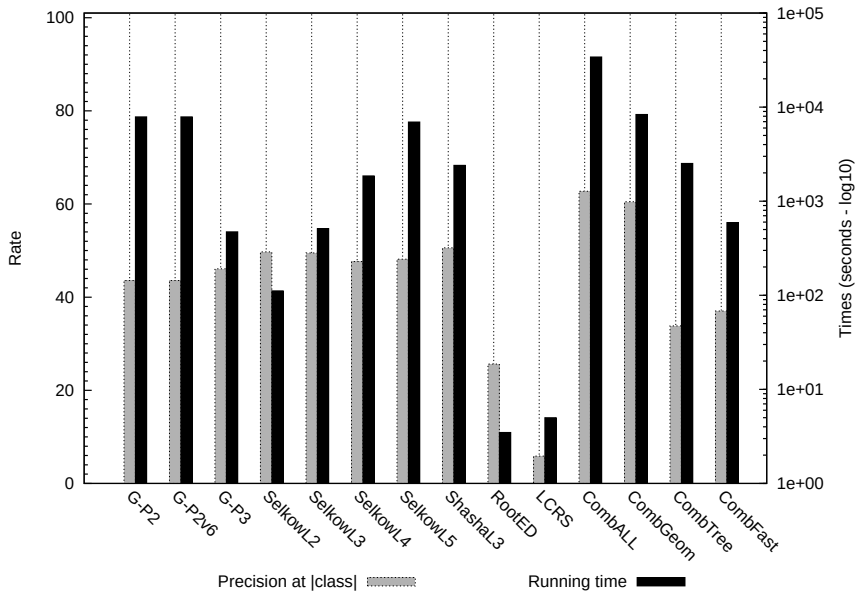

Fig. 10. Times and accuracy for corpus *VAR*.

method obtained the highest precision. However, with the pop-rock MIDI files the results were poor in average. This is unsurprising because in the MIDI files all the note information reside on a single track. As we played back the files, it was difficult even to a human to distinguish the songs because of the merged tracks and removed timbre. So, to preprocess the music by removing any accompaniment and ornament notes that represent noise to the comparing algorithms seems to be an appropriate task [16].

We have not yet done any fine-tunings to the LCRS and ROOTED methods. A better exploration of the representation space, and summarization at the top levels of the trees may conduce to better results in accuracy. Moreover, the processing times of LCRS can easily be improved by the substitution of the classical LCS algorithm with a faster implementation. Some experiments about including harmonic profiles, as described in [19], to refine that summarization have been done, but no successful results have been achieved so far.

An in depth study of the classification schemes and, hopefully, the inclusion of other paradigms of musical work comparison methods are to be tackled in the near future. In this way we expect that the precision results may still be improved considerably.

Finally, as the tree representation requires a metered input, we plan to include some existing methods, such as the ones described in [1, 19], in the preprocessor of our tree algorithm.

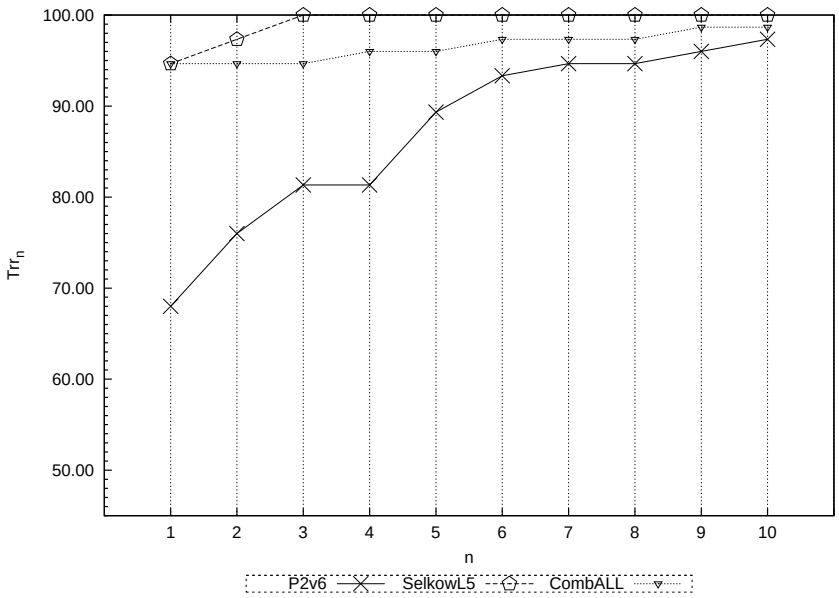

Fig. 11. TRR$_n$ for corpus *ICPS*.

References

1. B. Meudic, E. Staint-James. Automatic Extraction of Approximate Repetitions in Polyphonic Midi Files Based on Perceptive Criteria In *Proc. Int. Symp. Computer Music Modeling and Retrieval, 2003*, pages 124–142, 2003.
2. L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proc.* 7^{th} *Int. Symp. on String Processing Inf. Retrieval*, page 39–48, 2000.
3. J.J. Bloch and R.B. Dannenberg. Real-time accompaniment of polyphonic keyboard performance. In *Proc. Int. Comp. Music Conference*, pages 279–290, 1985.
4. M. Clausen, R. Engelbrecht, D. Meyer, and J. Schmitz. Proms: A web-based tool for searching in polyphonic music. In *Proc. Int. Symp. on Music Inf. Retrieval*, 2000.
5. S. Doraisamy and S.M. Rüger. A polyphonic music retrieval system using n-grams. In *Proc. Int. Symp. on Music Inf. Retrieval*, 2004.
6. M.J. Dovey. A technique for "regular expression" style searching in polyphonic music. In *Proc. Int. Symp. on Music Inf. Retrieval*, pages 179–185, 2001.
7. H. Hyyrö. Bit-parallel LCS-length computation revisited. In *Proc.* 15^{th} *Australasian Workshop on Combinatorial Algorithms*, pages 16–27, 2004.
8. K. Lemström, V. Mäkinen, and N. Mikkilä. Sublinear time filters using linear size index for content-based music retrieval. Submitted.
9. K. Lemström and A. Pienimäki. On comparing edit distance and geometric frameworks in content-based retrieval of symbolically encoded polyphonic music. *Musicae Scientiae*, 4A:135–152, 2007.
10. A. Lubiw and L. Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *Proc. Int. Symp. on Music Inf. Retrieval*, pages 289–296, 2004.

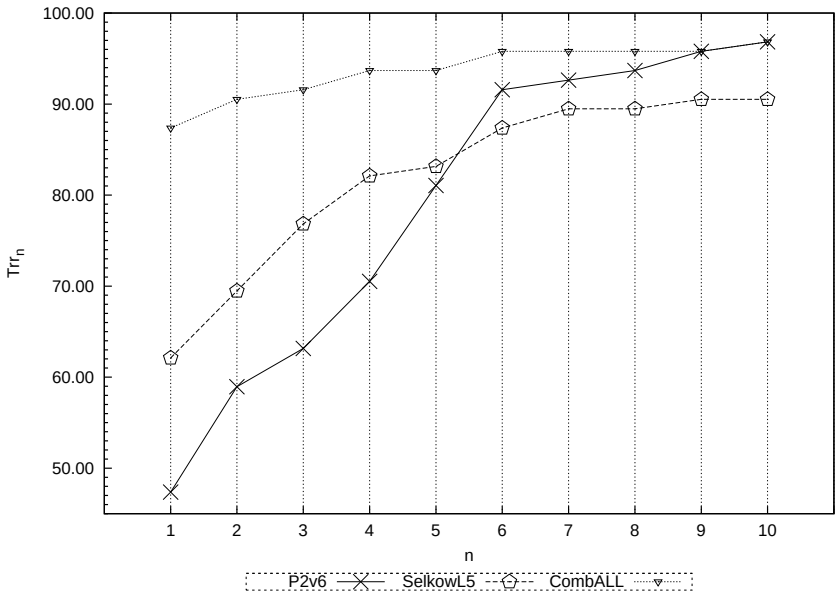

Fig. 12. $\textsc{Trr}_n$ for corpus *VAR*.

11. M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.
12. F. Moreno-Seco, J.M. Iñesta, P. Ponce de León, and L. Micó. Comparison of classifier fusion methods for classification in pattern recognition tasks. *Lecture Notes in Comp. Science*, 4109:705–713, 2006.
13. A. Pienimäki and K. Lemström. Clustering symbolic music using paradigmatic and surface level analyses. In *Proc. Int. Symp. on Music Inf. Retrieval*, pages 262–265, 2004.
14. D. Rizo, and J.M. Iñesta, P.J. Ponce de León. Tree model of symbolic music for tonality guessing. In *Proc. IASTED Int. Conf. on Artificial Intelligence and Applications*, pages 299–304, 2006.
15. D. Rizo and J.M. Iñesta. Tree-structured representation of melodies for comparison and retrieval. In *Proc.* 2^{nd} *Int. Conf. on Pattern Recognition in Inf. Systems*, 140–155, 2002.
16. D. Rizo, P.J. Ponce de León, and J.M. Iñesta. Towards a human-friendly melody characterization by automatically induced rules. In *Proc. Int. Symp. on Music Inf. Retrieval (to appear)*, 2007.
17. S.M. Selkow. The tree-to-tree editing problem. *Inf. Proc. Letters*, 6(6):184–186, 1977.
18. S. Shasha and K. Zhang. *Approximate Tree Pattern Matching. Pattern Matching Algorithms*, chapter 11, pages 341–371. Oxford Press, 1997.
19. D. Temperley. An evaluation system for metrical models. *Comp.Music J.*, 28(3):28–44, 2004.
20. A.L. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music databases. In *Proc. ACM Multimedia* , 1999.

21. E. Ukkonen, K. Lemström, and V. Mäkinen. Sweepline the music! In *Comp. Science in Perspective — Essays dedicated to Thomas Ottmann*, volume 2598 of *Lecture Notes in Comp. Science*, pages 330–342. Springer-Verlag, 2003.
22. G.A. Wiggins, K. Lemström, and D. Meredith. SIA(M)ESE: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *Proc. Int. Symp. on Music Inf. Retrieval*, 283–284, 2002.