New partially labelled tree similarity measure: a case study

David Rizo and José M. Iñesta

Dept. Lenguajes y Sistemas Informáticos, Universidad de Alicante,
E-03080 Alicante, Spain
{drizo,inesta}@dlsi.ua.es

Abstract. Trees are a powerful data structure for representing data for which hierarchical relations can be defined. They have been applied in a number of fields like image analysis, natural language processing, protein structure, or music retrieval, to name a few. Procedures for comparing trees are very relevant in many task where tree representations are involved. The computation of these measures is usually a time consuming tasks and different authors have proposed algorithms that are able to compute them in a reasonable time, through approximated versions of the similarity measure. Other methods require that the trees are fully labelled for the distance to be computed. In this paper, a new measure is presented able to deal with trees labelled only at the leaves, that runs in $O(|T_A| \times |T_B|)$ time. Experiments and comparative results are provided.

Key words: Tree edit distance, multimedia, music comparison and retrieval

1 Introduction

The computation of a measure of the similarity between two trees is a subject of interest in very different areas where trees are suitable structures for data coding. Trees are able to code hierarchical relations in their structure in a natural way and they have been utilized in many tasks, like text document analysis [8], protein structure [11], image representation and coding [2], or music retrieval [9], to name just a few.

Different approaches have been proposed in order to perform this comparison. Some of them pose a restriction on how the comparison is performed, other establish valid mappings. While some methods pay more attention to the tree structure, others do it to the content of the nodes and leaves. Most of them are designed to work with fully labelled trees.

The method proposed in this paper is designed to work with partially labelled trees, more precisely with those labelled only at the leaves. This fact, places the focus more on the coded content and the relations within its context. One of the fields where this situation is relevant is music comparison and

* This work was funded by the Spanish *DRIMS project* (TIN2009-14247-C02), and the research programme *Consolider Ingenio 2010* (MIPRCV, CSD2007-00018)

retrieval. Trees have been used for this task and a number of representation and comparison schemes have been applied based on tree edit distances [9] or probabilistic similarity schemes [3].

In any case, the computation of these measures is usually a time consuming task and different authors have proposed algorithms that are able to compute them in a reasonable time [12], through approximated versions of the similarity measure. In this paper, a new algorithm is presented, able to deal with trees labelled only at the leaves that runs in $O(|T_A| \times |T_B|)$ time, where $|T_x|$ stand for the number of nodes in tree T_x.

2 Tree comparison methods

A number of similarity measures for the ordered non-evolutionary trees have been defined in the literature. Some of them measure the sequence of operations needed to transform one tree in another one, others look for the longest common path from the root to a tree node, and there are methods that allow wildcards in the matching process in the so-called *variable-length doesn't care* (VLDC) distance. Several taxonomies of these measures have been proposed. The interested reader can look up a hierarchy of tree edit distance measures in [7] and [14], and a survey in [1].

2.1 Definitions and notations

In this section, the terms and notations that will be used in this paper will be defined.

Let $T = (V, E, L)$ be a labelled tree formed by a finite non-empty set V of vertices, a finite set $E \subseteq V \times V$ of arcs, and a set L of labels for nodes. Each node contains a label, possibly empty. The labelling function will be defined by $\text{label}\colon V \to L$. The empty tree will be denoted as λ, and the empty label as ϵ.

This tree is said to be a *labelled rooted tree* if there is a distinguished node $r \in V$, called the *root* of the tree and denoted by $\text{root}\colon T \to V$ such that for all nodes $v \in V$, there is an only path from root r to node v. All the trees used in this report are labelled rooted trees, so both terms will be used interchangeably.

The *level* or *depth* of a node $v \in V$, denoted by $\text{depth}\colon V \to \mathbb{N} \cup \{0\}$, is the length of the unique path from the root node $\text{root}(T)$ to node v. The *height* denoted by $\text{h}\colon T \to \mathbb{N} \cup \{0\}$ is defined as $\text{h}(T) = \max_{v \in V}\{\text{depth}(v)\}$.

Let two nodes $v, w \in V$, v is said to be the parent of w, if $(v, w) \in E$ and $\text{depth}(w) = \textit{depth}(v) + 1$. The parent of a node will be obtained by the function $\text{par}\colon V \to V$. The node w is said to be a child of v. Let's define the function $\text{children}\colon V \to 2^V$ as the set of all children of node v. $\text{children}(v) = \{w | \, \text{par}(w) = v\}$.

The arity, rank, or outdegree of a node $v \in V$, denoted by $\text{rank}\colon V \to \mathbb{N} \cup \{0\}$, is the number of children of a node. $\text{rank}(v) = |\, \text{children}(v)|$. When applied to a tree T, $\text{rank}(T) = \max_{v \in V}\{\text{rank}(v)\}$.

A node $v \in V$ is said to be a *leaf* if $\text{rank}(v) = 0$. A boolean function leaf : $V \to \mathbb{B}$ is defined to denote it. Similarly, leaves(T) is the set of nodes of that tree that have no children: $\{v \in V | \, \text{leaf}(v) = \text{true}\}$.

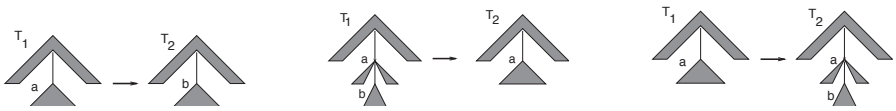
An *ordered tree* is a tree where the relative order of its children is fixed for each node. It allows us to define the function child: $\mathbb{N} \times V \to V$, such that $\text{child}_i(v)$ is just before $\text{child}_j(v)$ iff $i = j - 1$, $\forall i, j \in \mathbb{N}$.

The postorder numbering of a tree consists of giving the visit order of each node of the tree following a postorder traversal of the tree. To uniquely identify nodes in a tree T, lets define $T[i] \in \mathbb{N}$ as the ith node in a postorder numbering, beginning from 1.

A forest is a disjoint union of trees, and an ordered forest has the property that its components follow an order being this way a sequence of trees that will be denoted as $\mathcal{T}^+$. The operation $T[\text{child}_i(T)..\,\text{child}_j(T)]$ is the set composed by the children of T from positions i to j, both included, and it forms a forest. For abbreviating the notation, $T[i..j]$ will be used in the sequel to denote this operation.

Tree Edit Distance. The classical edit distance between two trees is the minimal cost to transform one input tree into an output one by edit operations. An *edit operation* over two trees $T_A = (V, E, L)$ and $T_B = (V', E', L')$ is any of the following:

- *relabel* the label l of a node $v \in V$ by the label l' of another node $w \in V'$, denoted by (v, w) (Fig. 1a).
- *deletion* of a non-root node $v \in V$, denoted by (v, λ), consists of deleting it, making the children of v become the children of $\text{par}(v)$, just in the position that was occupied by v, preserving this way the left to right ordering of leaves (Fig. 1b).
- *insertion* of a non-root node $w \in V'$, denoted by denoted by (λ, w). Given a sequence $w_i \cdots w_j$ of subtrees of a common parent w, the insertion of node w' makes those $w_i \cdots w_j$ subtrees children of w', and w' child of w (Fig. 1c).



(a) Substitute operation (b) Delete operation (c) Insert operation

Fig. 1: Tree edit operations (from [5]).

To each operation, a so-called *edit cost* $c_t \colon V \times V' \to \mathbb{R}$ is assigned based on that of the edit cost of the symbols at labels, $c \colon L \times L \to \mathbb{R}$ that depends on the given application. Therefore, $c_t(a, b)$ denotes the cost of applying the edit operation (v, w) where v is an input node and w is an output node. If $v = \lambda$, the operation denotes an insertion, if $w = \lambda$ the operation is a deletion. Note that the operation (λ, λ) is not allowed.

Definition 21 *An edit script* $e_t = e_{t_1} \cdots e_{t_n}$ *is a sequence of edit operations* $e_{t_i} = (a_i, b_i) \in (V \cup \{\lambda\}) \times (V \cup \{\lambda\})$ *allowing the transformation of a tree* X *into a tree* Y*. The cost of an edit script* $\pi_t(e_t)$ *is the sum of the costs of the edit operations involved in the script:* $\pi(e_t) = \sum_{i=1}^{n} c_t(e_{t_i})$.

Definition 22 *Let* $S_t(X, Y)$ *be the set of all the scripts that enable the emission of* Y *given* X*, the edit distance between* X *and* Y *is defined by:* $d_t(X, Y) = min_{e_t \in S_t(X,Y)} \pi(e_t)$.

2.2 Review of tree edit distances

The first author to give a solution to the general tree edit problem was Tai [13], proposing an algorithm with time complexity $O(|T_A| \times |T_B| \times \text{depth}(T_A)^2 \times \text{depth}(T_B)^2)$, where $|T_i|$ denotes the number of nodes in tree T_i. This algorithm was improved by Shasha and Zhang [15] giving a dynamic programming algorithm with time complexity $O(|T_A| \times |T_B| \times \min(\text{depth}(T_A), |\,\text{leaves}(T_A)|) \times \min(\text{depth}(T_B), |\,\text{leaves}(T_B)|))$.

The alignment distance is a restricted version of the edit distance based on forcing the application of all insertions before any deletions. Hence, the edit distance is always lower or equal than the optimal alignment [1]. It seems that alignment charges more for the structural dissimilarity at the top levels of the trees than at the lower levels, whereas edit treats all the levels the same [6]. In that work, an algorithm to solve the problem in $O(|T_A| \times |T_B| \times (\text{rank}(T_A) + \text{rank}(T_B))^2)$ time and $O(|T_A| \times |T_B| \times (\text{rank}(T_A) + \text{rank}(T_B)))$ space is given.

Another interesting variant of the edit distance was introduced by Selkow [12], where deletions and insertions are constrained to leaves. Thus, in order to delete an inner node, all its descendants must be deleted before. This algorithm has its strength in its low temporal cost $O(|T_A| \times |T_B|)$.

Finally, the bottom-up distance between two non-empty rooted trees T_A and T_B and is equal to $1 - f/\max(|T_A|, |T_B|)$, where f is the size of a largest common forest of T_A and T_B [14] . Valiente [14] reported a time complexity $O(|T_A| + |T_B|)$. However, this complexity is actually $O(|T_A| \times |T_B| \times \log(T_A + T_B))$, because in the original paper the computing of the bottom-up mapping is not included in the complexity calculation [1].

2.3 Proposed partially labelled tree comparison algorithm

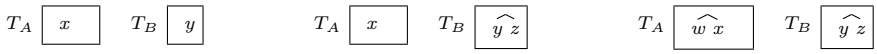
The presented similarity measures between trees are designed to work with fully labelled trees. In order to apply those algorithms to trees labelled only at leaves, the non-labelled inner nodes can be assigned a special label "empty". However it is expected that they don't work as well as they do with fully labelled trees.

[1] The nested loop in the mapping function (lines from 3 to 12 of the algorithm included in [14]) that traverses in level-order all the nodes of both trees leads to $O(|T_A| \times |T_B| \times \log(T_A + T_B))$, where the logarithm corresponds to the map operations on a $(|T_A| + |T_B|)$ size map inside the double loop.

In order to overcome this situation two approaches are possible. The first one consists of labelling all nodes using any bottom-up propagation scheme based on the application domain specific knowledge . The main drawback to that option is that any intermediate process might add noise to the resulting trees. The second approach is the definition of a similarity function designed just to compare those partially labelled trees.

The *partially labelled tree comparison algorithm* s_p is based on the assumption that the similarity value between a labelled leaf and a non-labelled inner node should be the average of chances of finding that leaf in the descendants of that inner node. Fig. 2a shows the simplest case of having two leaf trees: $s_p(T_A, T_B) = \delta(x, y)$, where $\delta(x, y) = 1 \iff x = y$, and 0 elsewhere. For comparing the trees shown in Fig. 2b, the chances of finding the label x in T_B are computed as $s_p(T_A, T_B) = (\delta(x, y) + \delta(x, z))/2$. If instead of being a label, y were another tree, the function should be computed recursively. Finally, when none of the trees is composed by a single leaf (Fig. 2c), the similarity of the ordered forests wx and yz can be computed like an edit distance between sequences wx and yz where each symbol is a tree.

This similarity method omits the accounting of the insertion or deletion of nodes and just measures the chance of finding coincident labels, giving more importance to the information hierarchically contained in the tree than to the tree structure.



(a) (b) (c)

Fig. 2: Similarity function s_p representative cases

Being designed for working with partially labelled trees, however, we can slightly adapt the original idea to work also with fully labelled trees. The case of comparing a leaf to a non-leaf tree (Fig. 3a), is computed as $s_p(T_A, T_B) = (\delta(x, b) + \delta(x, y) + \delta(x, z))/3$. Likewise, the similarity $s_p(T_A, T_B)$ between two fully labelled trees (Fig. 3b) is computed as the edit distance between sequences wx and yz, where each symbol is a tree, plus the similarity between labels a and b.

(a) (b)

Fig. 3: Similarity function s_p working on fully labelled trees.

Let $s_p \colon T \times T \to \mathbb{R}$ be a similarity function between trees and $sf_p \colon \mathcal{T}^+ \times \mathcal{T}^+$ be a similarity function between forests. Let us also use rlabel$\colon T \to L$ that returns the label of the root of the tree, and R_m as an abbreviation for rank(T_m). The similarity between two trees, fully or partially labelled, is defined as:

Definition 23

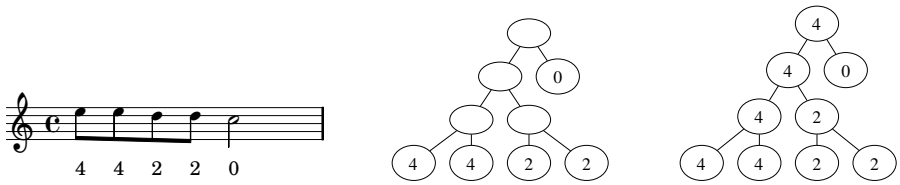
$$
\begin{aligned}
&(i) \quad s_p(T_A, T_B) = \\
&\quad \begin{cases}
\delta(\text{rlabel}(T_A), \text{rlabel}(T_B)) & : \textit{if } \text{leaf}(T_A) \wedge \text{leaf}(T_B) \quad (1) \\
\frac{\delta(\text{rlabel}(T_A), \text{rlabel}(T_B)) + \sum_{j=1}^{R_B} s_p(T_A, \text{child}_j(T_B))}{1+R_B} & : \textit{if } \text{leaf}(T_A) \wedge \neg\,\text{leaf}(T_B) \quad (2) \\
\frac{\delta(\text{rlabel}(T_A), \text{rlabel}(T_B)) + \sum_{i=1}^{R_A} s_p(\text{child}_i(T_A), T_B)}{1+R_A} & : \textit{if } \neg\,\text{leaf}(T_A) \wedge \text{leaf}(T_B) \quad (3) \\
\frac{\delta(\text{rlabel}(T_A), \text{rlabel}(T_B)) + sf_p(T_A, T_B)}{\max(R_A, R_B)+1} & : \textit{otherwise} \quad (4)
\end{cases} \\
&(ii) \quad sf_p(\lambda, \lambda) = 0 \\
&(iii) \quad sf_p(i..i', \lambda) = sf_p(i..i'-1, \lambda) \\
&(iv) \quad sf_p(\lambda, j..j') = sf_p(\lambda, j..j'-1) \\
&(v) \quad sf_p(i..i', j..j') = \\
&\quad \max \begin{cases}
sf_p(i..i'-1, j..j') \\
sf_p(i..i', j..j'-1) \\
sf_p(i..i'-1, j..j'-1) + s_p(T_A[i'], T_B[j'])
\end{cases}
\end{aligned}
$$

The simplest situation in Fig. 2a is solved by case (i)-(1). Cases (i)-(2) and (i)-(3) solve the problems depicted in Fig. 2b and 3a. Finally, (i)-(4) computes the similarity for Fig. 3b. After comparing the roots, the ordered forests composed by the tree children (Fig. 2c) are compared with the similarity function between forests sf_p in the the indirect recurrence (ii) to (v).

Complexity of the partial edit distance. In order to calculate the time complexity of s_p and sf_p, the functions $\mathcal{T}_s$ and $\mathcal{T}_{sf}$ will be used respectively. In both cases the size of the problem is the number of nodes of the compared trees: $(|T_A|, |T_B|)$.

$$
\mathcal{T}_s(|T_A|, |T_B|) = \begin{cases}
1 & : \quad if\ |T_A| = 1 \wedge |T_B| = 1 \\
R_B \times \mathcal{T}_s(|T_A|, |T_B| / \text{rank}(T_B)) & : \quad if\ |T_A| = 1 \wedge |T_B| > 1 \\
R_A \times \mathcal{T}_s(|T_A| / \text{rank}(T_A), |T_B|) & : \quad if\ |T_A| > 1 \wedge |T_B| = 1 \\
\mathcal{T}_{sf}(|T_A|, |T_B|) & : \quad if\ |T_A| > 1 \wedge |T_B| > 1
\end{cases}
$$

The function sf_p can be solved using a dynamic programming scheme as the used for any edit distance. On a classical edit distance, where the substitution cost has constant complexity, given the problem size $(|T_A|, |T_B|)$, the complexity is $O(|T_A| \times |T_B|)$ because its implementation is a simple double loop traversing a $|T_A| \times |T_B|$ matrix. However, in our case, in each step of that iteration, the s_p is called. Under these assumptions, the temporal complexity of the algorithm is obtained:



(a) The figures below the score indicate the Σ_p (interval from tonic that is C in this case).

(b) Corresponding tree representation.

(c) Corresponding propagated tree.

Fig. 4: A short melody sample and its representation as a tree.

$$\mathcal{T}_s(|T_A|, |T_B|) = c + \mathcal{T}_{sf}(|T_A| - 1, |T_B| - 1)$$
$$\mathcal{T}_{sf}(|T_A|, |T_B|) = \sum_{i=1}^{R_A} \sum_{j=1}^{R_B} \mathcal{T}_s(\frac{|T_A|}{R_A}, \frac{|T_B|}{R_B}) = R_A \times R_B \times \mathcal{T}_s(\frac{|T_A|}{R_A}, \frac{|T_B|}{R_B})$$

and it can be shown that this time is $O(|T_A| \times |T_B|)$.

3 Experiments

The experiments are devised to assess the suitability of the proposed algorithm working with both partially and fully labelled tree corpora, and compare it with classical tree comparison algorithms.

In the selected case study, the main goal is to identify a melody from a set of all the different variations played by the musicians. In our experiments, we use tree representations of monophonic music pieces [10] (Fig. 4c). The node labels are symbols from a pitch description alphabet Σ_p. In this paper, the interval modulo 12 from the tonic of the song is utilized as pitch descriptor (Fig. 4a): $\Sigma_p = \{p \mid 0 \leq p \leq 11\} \cup \{-1\}$, where -1 is used to encode rests. For measuring the similarity between the melody and each of the variations, the different tree comparison algorithms have been used.

Initially, only leaf nodes are labelled. Then, in order to reduce tree sizes and improve performing times, a pruning operation at level l can be applied that propagates bottom-up leaf labels until getting all leaves at level l or less labelled. For deciding which label to propagate, a melodic analysis [4] is applied to decide which notes are more important and then are more suitable to be promoted. Optionally, if we want to label all nodes in the tree, the same propagation method must be followed until reaching the root.

Corpora. In our experiments, we used a corpus consisting of a set of 420 monophonic 8-12 bar incipits of 20 worldwide well known tunes of different musical genres [2]. For each song, a canonic version was created by writing the score in a

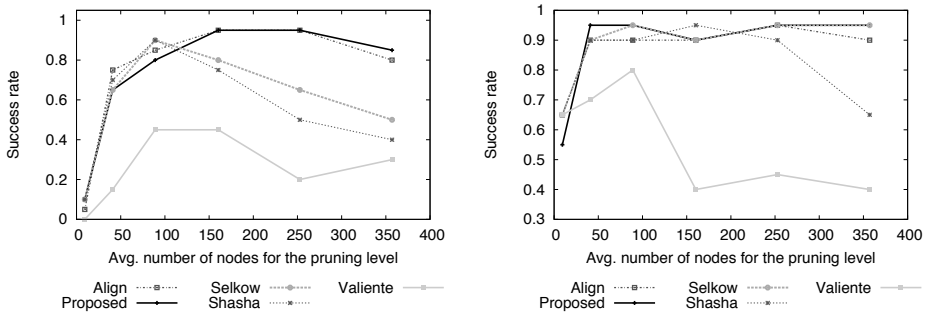
[2] The MIDI data set is available upon request to the authors.

musical notation application and exported to MIDI and MP3 format. The MP3 files were given to three amateur and two professional musicians who listened to the songs (mainly to identify the part of the tune to be played) and played with MIDI controllers the same tune several times with different embellishments and capturing performance errors. This way, for each of the 20 original scores, 20 different variations have been built.

Melody classification accuracy. The experiments have been performed using a query / candidates scheme, i.e., given a corpus, for each song the query prototype is compared to all the scores in the dataset. The similarity values are considered as distances and, following a nearest neighbor (NN) rule, the class of the file closest to the query will be taken as the retrieved song. This answer is correct if it corresponds to the same song of the query. Any other situation is considered as an error.

The success rate is measured as the ratio of correct answers to all the queries. Running times are measured in milliseconds taking into account only the test phase, leaving aside the construction of the representations that may be done off-line. All experiments have been performed using a MacBook Pro machine with 4 Gb RAM and 2 Intel(R) Core 2 Duo(R) CPU running at 2.26GHz, with a Java virtual machine 1.6.

Results. The experiments have been performed feeding all algorithms with several versions of the trees that have been pruned from level $l = 1$ to $l = 6$ (which is the maximum depth found in the corpus). For each pruning level the average number of nodes has been extracted to be used as x-axis in the plots.



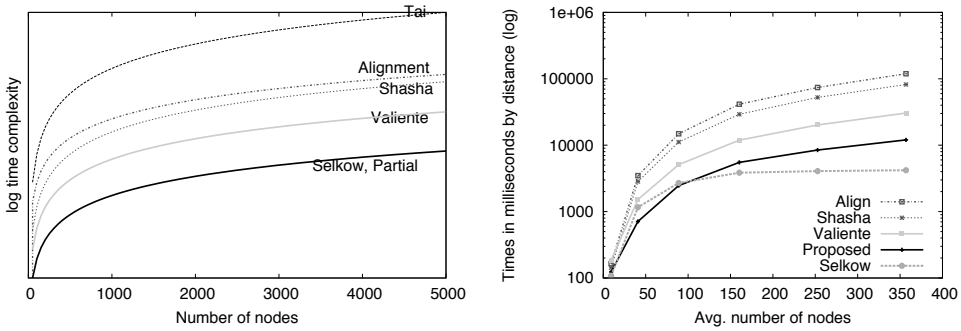
(a) Partially labelled trees (b) Fully labelled trees

Fig. 5: Success rates of proposed method compared to classical tree edit distances.

The results plotted in Fig. 5a show that the proposed algorithm behaves the best for not-pruned trees or $l = 6$ (see results for $x > 350$). For pruned trees, it behaves also the best in average.

When working with fully labelled trees (see Fig. 5b), the success rates of the proposed method are comparable to the success rates of the Selkow and

the Alignment distance. The plot in Fig. 6a shows the theoretical evolution of computing times of the main tree edit distance algorithms given their time complexity, and Fig. 6b describes the actual processing times of those distances in our experiments. The actual times confirm the prediction from the theoretical complexities, being our proposed algorithm the second faster one.



(a) Complexities theoretical processing times (b) Actual processing times

Fig. 6: Time processing evolution of algorithm.

Thus, it seems that the proposed similarity is suitable for its purpose, being able to compare successfully both trees labelled only at leaves and fully labelled trees better than the other methods in terms of trade-off between time and success rate.

4 Conclusions

In this paper a new similarity tree algorithm has been introduced for working with trees labelled only at the leaves. It has been applied to a real application: the similarity computation between monophonic symbolic music pieces encoded with both partially and fully labelled trees. The application has been tested using the proposed algorithm and classical tree similarity algorithms from the literature. From the results, it seems that the proposed algorithm outperforms the other ones in a trade-off among computing time and success rates.

In the future the algorithm must be applied to other applications that require this kind of measure to compare trees. Currently we are applying the same methodology to bigger corpora encoded with different pitch encodings and other propagation schemes, being the partial results obtained as good as those shown in this paper.

References

1. Philip Bille. A survey on tree edit distance and related problems. *Theorical Computer Science*, 337(1-3):217–239, 2005.

2. Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.
3. Amaury Habrard, José M.. Iñesta, David Rizo, and Marc Sebban. Melody recognition with learned edit distances. *Lecture Notes in Computer Science*, 5342:86–96, 2008.
4. Plácido R. Illescas, David Rizo, and José M. Iñesta. Harmonic, melodic, and functional automatic analysis. In *Proceedings of the 2007 International Computer Music Conferrence*, volume I, pages 165–168, 2007.
5. Carsten Isert. The editing distance between trees. Technical report, Institut für Informatik, Technische Universität München, 1999.
6. Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees – an alternative to tree edit. *Theoretical Computer Science*, 143(1):137 – 148, 1995.
7. Tetsuji Kuboyama, Kilho Shin, and Tetsuhiro Miyahara. A hierarchy of tree edit distance measures. *Theoretical Computer Science and its Applications*, 2005.
8. Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert Macintyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *In ARPA Human Language Technology Workshop*, pages 114–119, 1994.
9. David Rizo, Kjell Lemström, and José M. Iñesta. Tree representation in combined polyphonic music comparison. *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music. Lecture Notes in Computer Science*, 5493/2009:177–195, 2009.
10. David Rizo, F. Moreno-Seco, and José M. Iñesta. Tree-structured representation of musical information. *Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence*, 2652:838–846, 2003.
11. Robert B. Russell and Geoffrey J. Barton. Multiple protein sequence alignment from tertiary structure comparison: Assignment of global and residue confidence levels. *Proteins: Structure, Function, and Bioinformatics*, 14:309–323, 2004.
12. Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.
13. Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.
14. G. Valiente. An efficient bottom-up distance between trees. *International Symposium on String Processing and Information Retrieval*, 0:0212, 2001.
15. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.