A Phylogenetic Analysis of Near-Duplicate Audio Tracks

Matteo Nucci[1], Marco Tagliasacchi[2], Stefano Tubaro[3]

Dipartimento di Elettronica, Informazione e Bioingegneria
P.zza Leonardo da Vinci, 32 - 20133 Milano (Italy)
[1]matteo.nucci@mail.polimi.it
[2]marco.tagliasacchi@polimi.it
[3]stefano.tubaro@polimi.it

***Abstract*—We present a content-based system for the analysis of near-duplicate audio tracks. The objective is to infer the structure of modifications, represented as trees, underneath a pool of near-duplicates, possibly specifying the operations the tracks have gone through. A pilot study was carried out for a set of plausible processing operators, including trim, fade and perceptual audio coding, generating near-duplicates from an original audio track. The proposed method measures the similarity between pairs of near-duplicates and reconstructs a tree representing the causal dependencies in the analyzed pool. Experimental results demonstrate that the structure of the tree can be successfully recovered, also in the challenging case in which some pieces of information are missing, i.e., when only a subset of near-duplicate tracks is available.**

I. Introduction

In the last two decades, we have witnessed an exponential growth in the creation and redistribution of digital content. Thanks to the ever increasing popularity of the Internet and social media, it is now extremely easy to access, modify and share multimedia data through numerous platforms. Hence, images, videos and audio tracks can be seen as objects that change as they are distributed over the Web, a process that often involves only slight alterations (such as resizing or re-encoding) and which gives birth to similar, but not identical, copies of the original files.

The detection of exact duplicates is a fairly straightforward goal. Conversely, designing a system that is able to identify modified copies of a given object is a much more challenging task. In general, this problem has been referred to as *near-duplicate detection and recognition* (NDDR) [1][2][3][4]: *detection* focuses on verifying if two objects are near copies of each other, while *recognition* searches for all near-duplicates of a given query in a collection.

Techniques dealing with the detection of near-duplicates can be broadly categorized in two groups. Watermarking-based methods [5][6][7] rely on the embedding of a signature within the original document before its distribution. The presence of embedded data is verified to detect the original document. Instead, content-based methods [8][9][10][11][12][13] rely on the analysis of data to extract distinctive features, e.g., robust hashes. Based on these features, similarity measures are adopted to determine when a document is close to the original.

Once a pool of near-duplicate objects is isolated, it is interesting to identify the structure of relationships among them, so as to reveal, for example, which was the first object the other objects were derived from or understand the sequence of operations performed to obtain a given object. Such a challenging problem was recently addressed for the case of near-duplicate images in [14], [15] and [16], and further extended to videos in [17]. In particular, the overall scheme in these works is built upon the analogy between an object's generating structure and the concept of biological evolutionary development (i.e., *phylogeny*), which share the peculiarity of being a progressive branching process, graphically represented by trees.

Prompted by such studies, this work extends the set of techniques and procedures to the case of audio signals, a media widely used, and abused, in several contexts. Hence, given a set of near-duplicate audio tracks, the main objective is to identify the causal relationships and the processing operators that were applied, including the challenging case in which some pieces of information are missing, i.e., when only a subset of near-duplicate tracks is available. Based on the analogy with the referenced work, we name this problem *audio phylogeny*.

II. Near-duplicate audio tracks

According to the formulation given in [18] , a near-duplicate is a transformed version of an object that remains recognizable. Referring to the notation used in Figure 1, an object $\mathcal{D}_1$ is a near-duplicate of an object $\mathcal{D}$, if $\mathcal{D}_1 = T(\mathcal{D})$, $T \in \mathcal{T}$, where $\mathcal{T}$ is a set of *tolerated* processing operators. $\mathcal{D}$ is called the original object or *root*. A family of processing operators $\mathcal{T}$ can contain several combinations of operators, such as $\mathcal{D}_3 = T_3 \circ T_2 \circ T_1(\mathcal{D})$, $T_{\boldsymbol{\beta}=[1,2,3]} \in \mathcal{T}$. A duplicate is a pairwise equivalence relationship: it links the original document to any of his duplicates through a transformation. Thus, if $\mathcal{D}$ has a direct duplicate $\mathcal{D}_1$ and $\mathcal{D}_1$ has a direct duplicate $\mathcal{D}_2$, then document $\mathcal{D}_2$ is in turn a duplicate of document $\mathcal{D}$.

MMSP'13, Sept. 30 - Oct. 2, 2013, Pula (Sardinia), Italy.
978-1-4799-0125-8/13/$31.00

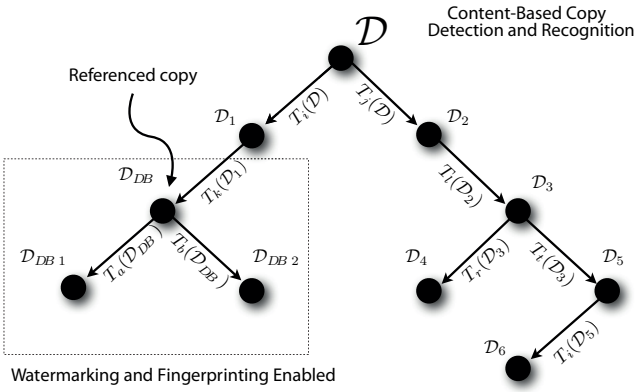


Fig. 1. Example of a tree representing the processing operators between near-duplicate objects [15].

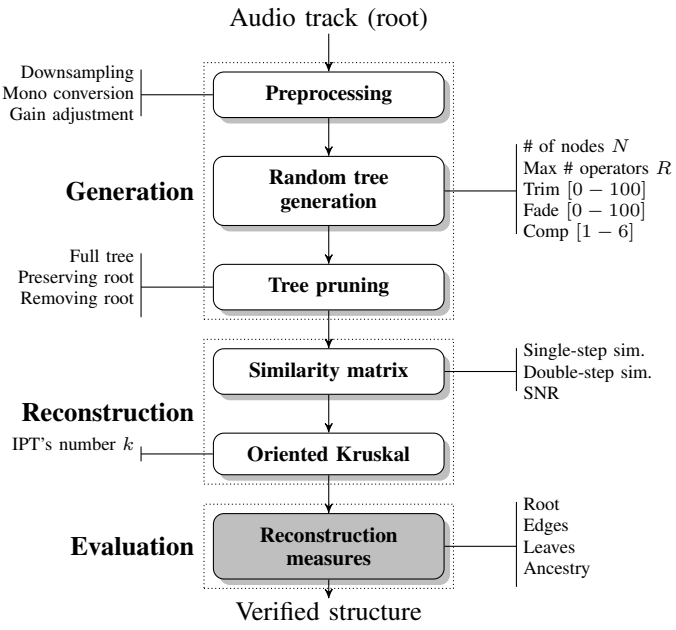


Fig. 2. Audio phylogeny framework. The workflow comprises generation, pruning and reconstruction of audio phylogeny trees (APTs) created using the tolerated processing operators.

An audio track can be transformed into a near-duplicate track by applying simple processing operators. In our study, the set $\mathcal{T}$ includes three types of operators, namely perceptual audio coding, fade and trim. These operators capture the actions performed when ripping an audio track from a CD or, in more general terms, when processing an uncompressed audio file to be shared online, or stored for playback on a mobile device.

Trim. The trim operator consists in the removal of the leading/trailing parts of a track, implemented by copying the audio samples while retaining only those within a specified range. Trimming is controlled by means of a parameter which takes values in the interval $[0, 100]$, where 0 indicates no trimming whereas 100 applies 3 second trimming at both ends of the audio track.

Fade. The fade operator modulates the sample amplitudes at the leading and trailing ends of the track, smoothing lead-in and lead-out. The maximum length of the fade applied was set to 3 seconds for both ends. The implementation was performed using *Sound eXchange*[1]. The parameter controlling the amount of fade was scaled to the interval $[0, 100]$, where 0 indicates no fading whereas 100 applies 3 second fading at both ends, with a logarithmic envelope.

Perceptual coding. Most audio tracks are available in a lossy coding format. In our work, we considered two different formats which are widely adopted, namely: i) mp3 (at 192, 256 kbps CBR and VBR V0), obtained with the *lame*[2] encoder; ii) m4a (at 256, 192, 128 kbps ABR), generated with the *faac*[3] encoder. Each of the six format@bitrate configuration was indexed with an integer in the range $[1, 6]$.

III. Generating a tree of near-duplicates

We generated different sets of near-duplicate audio tracks by randomly cascading the processing operators described in the previous section. Specifically, for each set, we considered an original, uncompressed, audio track as the root of a tree. Then, we grew a tree of near-duplicate objects by selecting at random up to R tolerated processing operators and the corresponding parameters. The resulting tree structures were stored as ground truth to evaluate the performance of the reconstruction algorithm detailed in Section IV. Three scenarios were tested. They are listed below sorted from the simplest to the most challenging.

- **Full trees:** all nodes of the original tree were retained.
- **Preserving root:** some of the tracks were removed to simulate missing links in the overall structure, with the additional constraint that the root track is always retained.
- **Removing root:** some of the tracks were removed, with the additional constraint that the root is always removed.

A. Preprocessing

A set of 50 audio tracks drawn from a music collection were used as root tracks. Songs were available as uncompressed stereo wave files (44100 Hz, 16 bit / sample) and belonged to various music genres. Using the *GoldWave*[4] editor, tracks with leading or trailing segments of silence lasting more than 5 seconds were trimmed to a maximum of 2 seconds. In addition, gain was slightly lowered (-1.5 dB) to avoid clipping in subsequent stages. Signals were downsampled to 22050 Hz and converted to mono, to reduce computational burden.

B. Random trees

Given an input track, the tree generation process is controlled by means of two parameters: the total number of near-duplicates tracks N (including the root), which equals the

[1]http://sox.sourceforge.net/
[2]http://lame.sourceforge.net/
[3]http://www.audiocoding.com/
[4]http://www.goldwave.com/

number of nodes in the tree; the maximum number R of processing operators between two adjacent nodes. Tree generation proceeds in an iterative fashion, adding near-duplicates one-by-one. Every time a near-duplicate is generated, the algorithm chooses at random the parent node (among all available near-duplicates), the number of operators to apply (between 1 and R), the kind of function(s) (within the tolerated set) and the relative parameter(s) (in the allowed range). The operation is repeated until a tree with N nodes is generated.

C. Tree pruning

In order to mimic cases that arise in realistic scenarios, we considered the challenging case in which some of the near-duplicate audio tracks were missing. This is achieved by pruning the tree, i.e., removing some of the nodes. The pruning procedure differs depending on the tested scenario (i.e., preserving root or removing root). The pruning function removes a randomly selected set of nodes from the full tree and and rebuilds a reference structure linking children of removed nodes to their closest ancestor. In case the root is removed, the nodes directly connected to it become roots themselves, thus generating a forest.

IV. Tree reconstruction

After building a reference tree (either full or pruned), the information regarding its structure (i.e., parent-child relationships and corresponding processing operators) is put aside and the analysis is carried out considering only the available audio tracks, according to a content-based paradigm. Hence, the input data of the tree reconstruction algorithm consists of a set of $N - P$ tracks, where $P \in [0, N-2]$ indicates the number of pruned nodes. The output is a tree representing the estimated relationships among audio tracks.

A. Similarity function

The key element to reconstruct the relationships among near-duplicates is the definition of a function that measures the (asymmetric) degree of similarity between a pair of audio tracks. Hence, given two audio tracks S_A and S_B, we want to estimate the most likely processing operators $T_{\boldsymbol{\beta}}$ that could potentially transform S_A into S_B. Note that $\boldsymbol{\beta}$ indexes the sequence of the candidate functions of the set $\mathcal{T}$, i.e., $\mathcal{T}_{\boldsymbol{\beta}}(\cdot) = \mathcal{T}_{\beta_R}(\cdots(\mathcal{T}_{\beta_1}(\cdot))$. The similarity value is computed as:

$$s_{A,B} = \arg\min_{\boldsymbol{\beta}} \mathcal{L}\left(S_B, T_{\boldsymbol{\beta}}(S_A)\right). \tag{1}$$

In this work the sample-wise measure $\mathcal{L}$ used is the SNR (*signal to noise ratio*),:

$$\mathcal{L}\left(S_B, T_{\boldsymbol{\beta}}(S_A)\right) = 20\log_{10}\left(\frac{\|T_{\boldsymbol{\beta}}(S_A)\|_2}{\|S_B - T_{\boldsymbol{\beta}}(S_A)\|_2}\right) \tag{2}$$

The similarity value given in (2) is large when S_A is the direct parent of S_B, or more generally belongs to its ancestors, and $T_{\boldsymbol{\beta}}(\cdot)$ approximates the sequence of operators applied to S_A to get S_B.

B. Operator and parameter search

To calculate the similarity s_{S_A,S_B} between two tracks, an analyst would ideally want to enumerate the candidate processing operators (and corresponding parameters). This can be a time-consuming task, also when a small number of tolerated operators is allowed. Hence, for each tolerated operator $\mathcal{T}_\beta$, we defined a *check function*, which accepts as input two tracks to compare and produces as output: i) an estimate of the operator-specific parameter $\hat{\alpha}_\beta$; the SNR calculated as in 2, when $\boldsymbol{\beta} = \beta$. Below we detail the check functions for the three operators considered in our study.

Trim is detected computing the cross correlation between S_A and S_B on downsampled signals. This returns the offset, in samples, for which the correlation between the two considered signals is maximum. Then, this is normalized to the value $\bar{\alpha}_t$ in the range $[0, 100]$. The estimated parameter $\hat{\alpha}_t$ is the one in the set $[\bar{\alpha}_t - 1, \bar{\alpha}_t, \bar{\alpha}_t + 1]$ that leads to the largest value of SNR.

Fade is applied on the input track S_A for a subset of potential values of the parameter, namely within the interval $[3, 100]$ with a step of 5. Parameter values are iteratively tested from the smallest to the largest. To speed up processing, the iteration is stopped and the value of $\hat{\alpha}_f$ is returned whenever the calculated value of SNR is above a given threshold (tuned manually to 65 dB). Otherwise, let $\bar{\alpha}_f$ be the value of the parameter that leads to the largest SNR. The value of $\hat{\alpha}_f$ is obtained testing $[\bar{\alpha}_f - 2, \bar{\alpha}_f, \bar{\alpha} + 2]$. The check function applies fade on an audio track only slightly longer than the maximum interval allowed, i.e., 3 seconds. Once $\hat{\alpha}_f$ is found, fade is applied to the full tracks to get a consistent measure of the SNR.

Comp applies perceptual audio coding enumerating all 6 combinations of format@bitrate. This operation is the most time consuming, as it requires repeated encoding. Therefore, compression is applied on a short segment of the track, about one tenth of the original track length. It was found that the algorithms behind the various formats actually processed the samples in a very characteristic way (leaving a sort of *footprint*, similarly to the case of video coding [19]), which could be revealed by considering only a short segment, thus speeding up the process. The estimated format $\hat{\alpha}_c$ is the one that maximizes the SNR and it is used as the input parameter to recompress the whole track and get a consistent measure of the SNR.

An exhaustive search of the space of candidate operators (and corresponding parameters) is practically unfeasible, since the asymptotic time complexity is exponential in the number of operators applied. In our study, we considered that the analysis entails up to two cascaded operators (double-step similarity). We propose to apply a greedy strategy, which considers one operator at the time, as illustrated in Figure 3. At each step, the method estimates the most likely operator and the corresponding parameter, and returns the SNR. This is repeated twice, although it can be generalized to a larger number of steps. The algorithm might be prematurely terminated if a perfect

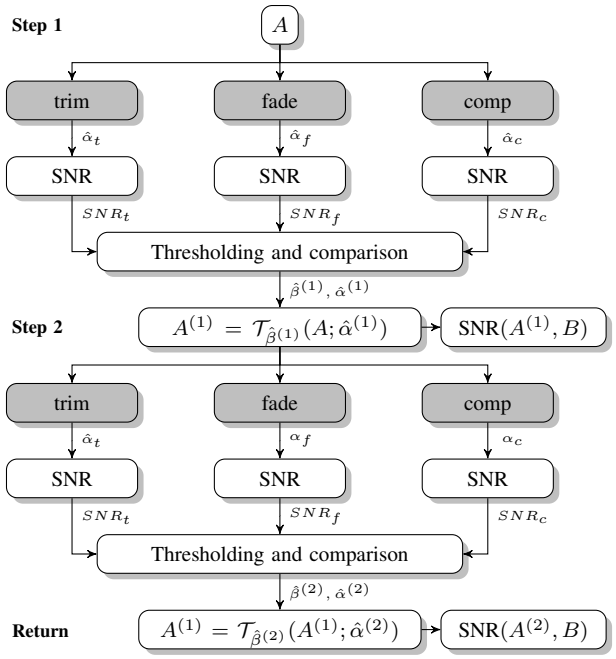


Fig. 3. Computation of double-step similarity function. In case of single-step similarity, the computation is terminated after the first step.

match is found, which can be detected when observing a SNR value above a specified threshold (tuned manually to 65 dB).

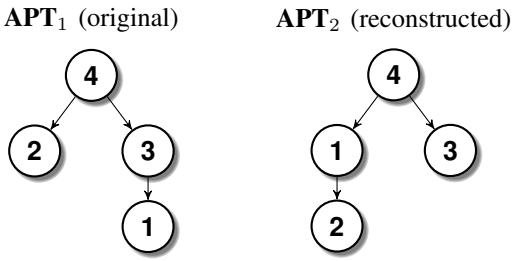
C. From similarities to trees

Given a pool of $N-P$ near-duplicates, an $(N-P)\times(N-P)$ similarity matrix M is populated with the pair-wise similarity scores computed as in (1). This matrix is fed as input to the oriented Kruskal's algorithm, introduced in [16], a heuristic method that relies on Kruskal's procedure for finding a minimum spanning tree in weighted, oriented graphs. Generally, Kruskal's algorithm implements a greedy procedure selecting edges of minimum weight successively from a connected weighted graph without forming cycles, until a spanning tree has been produced. Oriented Kruskal's algorithm takes as inputs the similarity matrix M and the number of roots k to search (e.g., $k = 1$ implies a single tree). The algorithm starts by initializing a tree for every node. Next, the similarity matrix M is sorted in descending order of similarity. Values and related nodes are analyzed to assess if an edge links them, that is, if they have different roots and whether the potential child has not been already linked. The algorithm stops when the number of drawn edges reaches $N - P - k$. This implies that all nodes but k (the roots) have been connected to their estimated parent.

V. Experimental evaluation

A. Evaluation metrics

In our work, we adopted the metrics devised in [16] to objectively evaluate the correctness of the solution found by the proposed method. For completeness, these metrics are briefly summarized in the following.



Trees

$\text{APT}_1 = [3, 4, 4, 4]$
$\text{APT}_2 = [4, 1, 4, 4]$

Root

$R(\text{APT}_1) = 4$
$R(\text{APT}_2) = 4$

Edges

$E_1 = \{(3 \to 1), (4 \to 2), (4 \to 3)\}$
$E_2 = \{(4 \to 1), (1 \to 2), (4 \to 3)\}$

Leaves

$L_1 = \{1, 2\}$
$L_2 = \{2, 3\}$

Ancestry

$A_1 = \{(4 \to 1), (4 \to 2), (4 \to 3), (3 \to 1)$
$A_2 = \{(4 \to 1), (4 \to 2), (4 \to 3), (1 \to 2)$

Fig. 4. Example of comparison of APTR vs. APT reference tree. Metrics scores: *root* = 1, *edges* = 33%, *leaves* = 33%, *ancestry* = 60%.

Let APT (Audio Phylogeny Tree) and APTR (Audio Phylogeny Tree - Reconstructed) denote, respectively, the data structures representing the ground truth and the output of the tree reconstruction algorithm. Two different sets of metrics are defined, depending on whether the root is removed or not. When the root is retained, the reconstructed tree APTR is compared against the reference tree APT using the following metrics and setting $\text{APT}_1 = \text{APT}$ and $\text{APT}_2 = \text{APTR}$:

$$\textbf{Root}: R(\text{APT}_1, \text{APT}_2) = \begin{cases} 1, \text{if root}(\text{APT}_1) = \text{root}(\text{APT}_2) \\ 0, \text{otherwise} \end{cases}$$

$$\textbf{Edges}: E(\text{APT}_1, \text{APT}_2) = \frac{|E_1 \cap E_2|}{|E_1|}$$

$$\textbf{Leaves}: L(\text{APT}_1, \text{APT}_2) = \frac{|L_1 \cap L_2|}{|L_1 \cup L_2|}$$

$$\textbf{Ancestry}: A(\text{APT}_1, \text{APT}_2) = \frac{|A_1 \cap A_2|}{|A_1 \cup A_2|}$$

The *root* metrics checks if the original track was detected; *edges* compares the lists of parent-child links; *leaves* compares nodes that do not have children; and *ancestry* compares lists identifying all children deriving from every single node, of any generation. Figure 4 illustrates an example, computing these metrics for two exemplary trees.

Instead, if the root is removed, the APTR is tested against the resulting reference forest APF (Audio Phylogeny Forest), and the metrics are modified accordingly. In our experiments, APTR is always a tree ($k = 1$). Determining the correct value

of k from the available near-duplicates is subject of current investigation.

$$\textbf{Root}: R_F(\text{APF}, \text{APTR}) = \begin{cases} 1, \text{if root(APTR)} \in \text{root(APF)} \\ 0, \text{otherwise} \end{cases}$$

$$\textbf{Edges}: E_F(\text{APF}, \text{APTR}) = \frac{|E_T \cap E_F|}{|E_F|}$$

$$\textbf{Leaves}: L_F(\text{APF}, \text{APTR}) = \frac{|L_T \cap L_F|}{|L_T \cup L_F|}$$

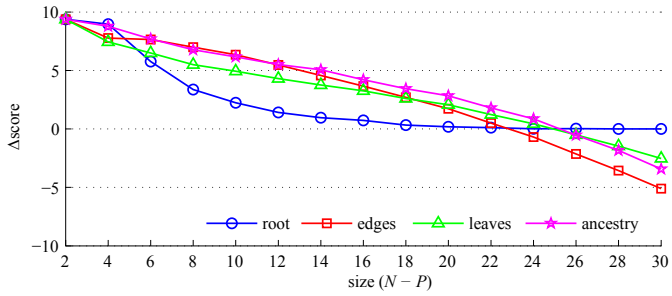
$$\textbf{Ancestry}: A_F(\text{APF}, \text{APTR}) = \frac{|A_T \cap A_F|}{|A_F|}$$

B. Results

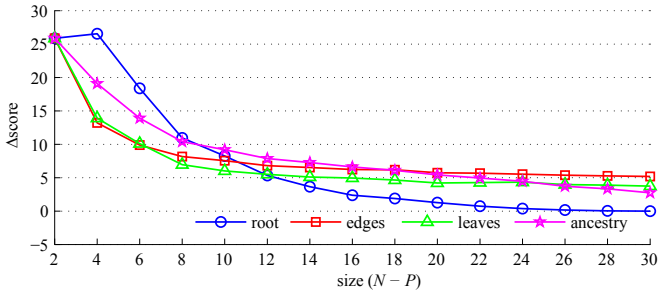
First, we tested the proposed algorithm in the case of full trees, i.e., all near-duplicates were available as input to the tree reconstruction algorithms and $R = 1$, i.e., only one operator was applied between a parent and its child. Similarly to [16], we considered trees with up to $N = 50$ nodes. For each tree size, results were averaged over 50 randomly generated ground-truth reference trees. In this simple scenario, the root was always correctly recovered. As for the other metrics, they were always above 90% and the same results were obtained using either single-step or double-step similarity.

Then, we turned our focus to more challenging cases, in which some of the near-duplicates were not available. We started with reference trees with $N = 50$ nodes and we pruned $P = 2, \ldots, 48$ nodes. Figure 5(a) and Figure 5(b) show the results in the case the root node is, respectively, either retained or removed. The root node is correctly identified in nearly all cases. The non-monotone behaviour of the curves can be explained as follows. For large values of $N - P$, the tree is almost full, with just a few missing nodes, thus being relatively easy to reconstruct. For small values of $N-P$, just a few nodes are retained. Hence, the trees are quite simple and can can be easily recovered.

We also tested the effectiveness of double-step similarity, with respect to the computationally simpler single-step similarity. Figure 6 illustrates the gain, defined as $\Delta score = score_{\text{double}} - score_{\text{single}}$, where $score$ is the value of one of the tested metrics. We considered two cases, $R = 1, 2$, i.e., either one or a maximum of two operators were applied between a parent and its child in the full tree (with $N = 30$ nodes), before pruning. When $R = 1$, Figure 6(a) shows that double-step similarity is useful especially for pruned trees. This was expected, since in the case of missing nodes, there might be more than one operator between two related near-duplicates. When the size approaches that of the full tree, single-step similarity is preferable. Indeed, in this case the tree is almost full, and when $R = 1$ only one operator is in between two related nodes in most cases. Instead, when $R = 2$ (see Figure 6(b)), double-step similarity is always beneficial, and gains can be significant, especially for highly pruned trees.



(a)



(b)

Fig. 6. Comparison of tree reconstruction using either double-step or single-step similarity. a) $R = 1$; b) $R = 2$. Trees with missing links preserving root.

VI. Conclusions and Future Work

In this work we presented a system that performs content-based analysis of near-duplicate audio tracks. Specifically, the objective was to infer the structure of modifications underneath a pool of near-duplicates, possibly specifying the operations the tracks had gone through.

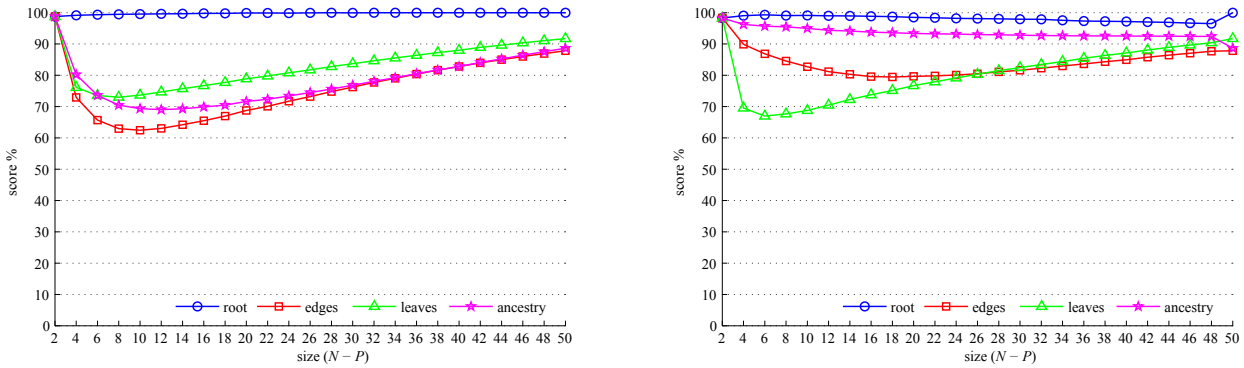
A pilot study was carried out considering three operators typically used to generate near-duplicates: perceptual audio coding, fade and trim. The core contribution was the definition of an algorithm to measure the similarity between pairs of audio tracks and an extensive experimental evaluation of the method.

Results were aligned with those obtained in the case of image near-duplicates, revealing that, in most cases, it is possible to successuffly recover the structure of the tree, also in the challenging case in which some of the near-duplicates are missing.

While this work considered data generated in a controlled environment, future work will analyze near-duplicates retrieved from real world data sources. In addition, we will investigate how to extend the set of tolerated processing operators to include, for example, time- and pitch-scaling.

Acknowledgment

The project REWIND acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 268478.



(a) (b)

Fig. 5. Reconstruction of trees with missing links: a) preserving root; b) removing root.

REFERENCES

[1] H. Hajishirzi, W.-t. Yih, and A. Kolcz, "Adaptive near-duplicate detection via similarity learning," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '10. New York, NY, USA: ACM, 2010, pp. 419–426.

[2] E. Vallés and P. Rosso, "Detection of near-duplicate user generated contents: the sms spam collection," in *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, ser. SMUC '11. New York, NY, USA: ACM, 2011, pp. 27–34.

[3] L. Bueno, E. Valle, and R. da S. Torres, "Bayesian approach for near-duplicate image detection," in *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, ser. ICMR '12. New York, NY, USA: ACM, 2012, pp. 15:1–15:8.

[4] M. Yang, G. Qiu, J. Huang, and D. Elliman, "Near-duplicate image recognition and content-based image retrieval using adaptive hierarchical geometric centroids," in *Proceedings of the 18th International Conference on Pattern Recognition - Volume 02*, ser. ICPR '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 958–961.

[5] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Syst. J.*, vol. 35, no. 3-4, pp. 313–336, Sep. 1996.

[6] M. Barni, F. Bartolini, V. Cappellini, and A. Piva, "A dct-domain system for robust image watermarking," *Signal Processing*, vol. 66, no. 3, pp. 357–372, 1998.

[7] I. Cox, M. L. Miller, and J. A. Bloom, *Digital watermarking*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[8] D. P. Huijsmans and M. S. Lew, "Efficient content-based image retrieval in digital picture collections using projections (near)-copy location," in *Proceedings of the International Conference on Pattern Recognition (ICPR '96) Volume III-Volume 7276 - Volume 7276*, ser. ICPR '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 104–.

[9] M. S. Kankanhalli, K. R. Ramakrishnan, and Rajmohan, "Content based watermarking of images," in *Proceedings of the sixth ACM international conference on Multimedia*, ser. MULTIMEDIA '98. New York, NY, USA: ACM, 1998, pp. 61–70.

[10] J. Fridrich, "Robust hash functions for digital watermarking," in *ITCC*. IEEE Computer Society, 2000, pp. 178–183.

[11] J. H. Ton and T. Kalker, "Robust audio hashing for content identification," in *In Content-Based Multimedia Indexing (CBMI*, 2001.

[12] A. Saracoglu, E. Esen, T. K. Ates, B. O. Acar, U. Zubari, E. C. Ozan, E. Ozalp, A. A. Alatan, and T. Ciloglu, "Content based copy detection with coarse audio-visual fingerprints," in *Proceedings of the 2009 Seventh International Workshop on Content-Based Multimedia Indexing*, ser. CBMI '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 213–218.

[13] M. Tagliasacchi, G. Valenzise, and S. Tubaro, "Hash-based identification of sparse image tampering," *IEEE Transactions on Image Processing*, vol. 18, no. 11, pp. 2491–2504, 2009.

[14] A. De Rosa, F. Uccheddu, A. Costanzo, A. Piva, and M. Barni, "Exploring image dependencies: a new challenge in image forensics," pp. 75 410X–75 410X–12, 2010.

[15] Z. Dias, A. Rocha, and S. Goldenstein, "First steps toward image phylogeny," in *IEEE Workshop on Information Forensics and Security (WIFS)*, 2010.

[16] ——, "Image phylogeny by minimal spanning trees," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 774–788, 2012.

[17] ——, "Video phylogeny: Recovering near-duplicate video relationships," in *WIFS*. IEEE, 2011, pp. 1–6.

[18] A. Joly, O. Buisson, and C. Frelicot, "Content-based copy retrieval using distortion-based probabilistic similarity search," *Trans. Multi.*, vol. 9, no. 2, pp. 293–306, Feb. 2007.

[19] P. Bestagini, A. Allam, S. Milani, M. Tagliasacchi, and S. Tubaro, "Video codec identification," in *ICASSP*. IEEE, 2012, pp. 2257–2260.