Title: SuMoTED: An intuitive edit distance between rooted unordered uniquely-labelled trees

Authors: Matt McVicar, Benjamin Sach, Cédric Mesnage, Jefrey Lijffijt, Eirini Spyropoulou, Tijl De Bie

The final publication is available at Elsevier via
http://dx.doi.org/10.1016/j.patrec.2016.04.012

Pattern Recognition Letters 79 (2016) 52–59

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

SuMoTED: An intuitive edit distance between rooted unordered uniquely-labelled trees☆

Matt McVicar[a,*], Benjamin Sach[b], Cédric Mesnage[a], Jefrey Lijffijt[a,c], Eirini Spyropoulou[a], Tijl De Bie[a,c]

[a] Department of Engineering Mathematics, University of Bristol, Woodland Road, Bristol BS81UB, England
[b] Department of Computer Science, University of Bristol, Woodland Road, Bristol BS81UB, England
[c] Data Science Lab, Ghent University, Technicum, Ghent 9000, Belgium

ARTICLE INFO

Article history:
Received 10 November 2015
Available online 4 May 2016

Keywords:
Tree edit distance
Taxonomies

ABSTRACT

Defining and computing distances between tree structures is a classical area of study in theoretical computer science, with practical applications in the areas of computational biology, information retrieval, text analysis, and many others. In this paper, we focus on rooted, unordered, uniquely-labelled trees such as taxonomies and other hierarchies. For trees as these, we introduce the intuitive concept of a 'local move' operation as an atomic edit of a tree. We then introduce SuMoTED, a new edit distance measure between such trees, defined as the minimal number of local moves required to convert one tree into another. We show how SuMoTED can be computed using a scalable algorithm with quadratic time complexity. Finally, we demonstrate its use on a collection of music genre taxonomies.

1. Introduction

The problem of computing how (dis)similar two trees are, and the related problem of computing a consensus between a set of trees, has applications in computational biology, chemistry, music genre analysis, and automatic theorem-proving [14,18,23,25]. For example, calculating the distance between RNA secondary structures (which have a tree structure) is necessary to understand their comparative functionality [26]. Taxonomies, such as the one shown in Fig. 1, offer another natural application area. Indeed, quantifying the similarity between different taxonomies may provide insight into what might be the consensus as well as the nature of any subjective differences between different taxonomy creators.

Given the wide range of application areas listed above, it is not surprising that computing the similarity between trees is an actively studied problem within computer science, and the literature is abundant with similarity measures for various types of trees. However, computational tractability is often a problem. For example, for rooted, unordered, fully-labelled trees (trees with a root, in which every vertex is labelled and the left-to-right order of siblings carries no significance, such as taxonomies and other hierarchies), a recent survey [5] discusses three distances that are all NP-hard. More details are presented in Section 2.

The current paper aims to tackle this problem in a specific setting by introducing the Subtree Moving Tree Edit Distance (SuMoTED): a new tree distance measure with several appealing properties. First, it is an edit distance, defined intuitively as the minimum number of atomic *local moves* of vertices up and down required to turn one tree into the other, weighted by the size of the moved subtree. Second, it is not only intuitive but is also a metric distance, meaning it is easy to use in a wide range of information retrieval and machine learning algorithms. For example, distance-based methods for clustering often require the distance measure to be metric, and metric properties are also used for efficient document retrieval in databases. Third, it can be computed in a time that is quadratic in the total number of vertices in the trees. Finally, our method produces a consensus tree as part of the procedure, allowing us to compute the agreement between a set of trees at no additional cost.

We begin this paper with a literature survey on tree distances in Section 2. Subsequently, we define SuMoTED as a novel distance measure between two rooted, unordered, uniquely-labelled trees (Section 3). We then give an efficient algorithm for its computation (Section 4), before evaluating SuMoTED experimentally in Section 5 and concluding in Section 6.

☆ This paper has been recommended for acceptance by Dr. D. Coeurjolly.

* Corresponding author. Tel.: +44 7739901492.
E-mail address: mattjamesmcvicar@gmail.com (M. McVicar).

http://dx.doi.org/10.1016/j.patrec.2016.04.012

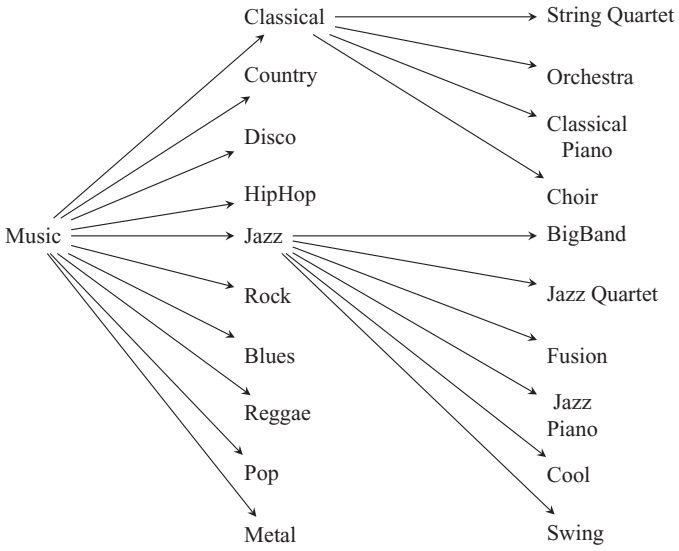

Fig. 1. Music genre hierarchy proposed by Tzanetakis and Cook [25].

2. Related work

2.1. Distances between trees

The task of defining and computing distances between trees can be considered a special case of the graph comparison problem, which has an extensive literature: see Gao et al. [13] for a summary. When the graphs are directed and contain no cycles, the graph becomes a Directed Acyclic Graph (DAG)—some authors have studied the distances between DAGs [6,16]. Most graph and DAG distances could ofcourse be deployed for trees. Despite this, often the particular structure of trees such as the notion of a root, the unique parent of a vertex, or sibling relationships, are important in designing meaningful tree distance measures, such that measures for graphs become unnatural when applied to trees.[1] For this reason, defining tree distances has become and active research topic. Bille [5] offers a comprehensive overview of the most common methods for comparing trees, including best-known time and space complexity bounds. Here we give only an overview, referring the reader to Bille for further details.

The first and most widely-used method for comparing trees is the *tree edit distance*, introduced by Tai [24] as an extension of the well-known string edit distance. Tai allows insertion, deletion and substitution of vertices in order to convert a source tree T_s into a target tree T_t. A cost function is then applied to these operations (most commonly setting the cost of each transformation to unity), and the minimum number of these operations is defined to be the distance between T_s and T_t. Several algorithms have been proposed to efficiently solve the tree edit distance, but only on ordered trees or other special cases. The case for unordered trees is known to be NP-hard [5]. In the original formulation of the tree edit distance, inserting a vertex u between a vertex v and its parent p meant that u became a child of p, and v and all of its descendants became a child of u. Restrictions were also introduced such as the top-down distance [21] which only allowed insertions to occur at leaves. Another modification is the bottom-up distance: let the number of nodes in the source and target tree be n_s and n_t and the size of the largest common forest of T_s and T_t be f. The distance between T_s and T_t is then defined to be $1 - f/\max(n_s, n_t)$. The best known algorithm for the bottom-up problem is linear in n_s and n_t, and is applicable to both ordered and unordered trees.

Tree alignment is an alternative method and proceeds as follows. Nodes with no labels are inserted into T_s and T_t until they are isomorphic[2], producing T_s' and T_t'. This produces an alignment tree A, whose vertex labels are pairs of labels taken from T_s' and T_t'. The cost of A is the total cost of substituting each vertex pair such that they are equal—the tree alignment distance between T_s and T_t is the minimum such cost. Finally, the *tree inclusion* problem is to determine if T_t may be obtained from T_s via deleting nodes. As with the tree edit distance for unordered trees, computing either the tree alignment distance or the tree inclusion problem is MAX SNP-hard [5].

2.2. Computing consensus trees

Given a set of trees, a distinct but clearly related task to the tree distance problem is to determine what information is shared by the set. Shasha et al. [22] claims there are five commonly-used methods for achieving this, which we review here. The first was introduced by Adams III [1] and is known as the *Adams consensus* in the literature. This method is applicable to both fully-labelled and leaf-labelled trees (where only leaf vertices have labels). Leaf-labelled trees are more common in taxonomic biological applications. Next, Day [11] proposed a new method for computing the consensus, and also introduced a distance measure based on the number of common subtrees found within two trees in the collection—this method is known as the *strict consensus*.

Margush and McMorris [17] pointed out that in the case that many of the trees in a large set are identical (say, equal to T) and one differs from T by a single edge, that the consensus should be equal to T. To achieve this, he introduced the *majority rule consensus*, where a parent–child relationship in the consensus is only introduced if at least half the trees share the same link. The *semi-strict consensus* tree for leaf-labelled trees [7] includes all subtrees by Adams' method, but also any subtrees which are not contradicted by other members of the group. Finally, the *Nelson consensus* [20] consists of the set of mutually compatible subtrees that are most frequently replicated in the group.

Interestingly, the computation of a consensus tree can be considered a special case of frequent subtree mining, an area of research which has received a good deal of attention in recent years [4,8,9].

2.3. Limitations of existing work

As seen above there are many existing methods which either compute the distance between trees, or compute a consensus between a set of trees. Yet, all existing distance measures suffer from one of both of the following problems:

Computational cost We are interested in the case of uniquely-labelled unordered trees as these occur frequently in application areas (such as biological sequence analysis, text mining and music information retrieval). Although, to the best of our knowledge, this specific case has not been studied, for general unordered labelled trees the three existing distances discussed above (tree edit distance, alignment distance and tree inclusion) are not efficiently computable.

Interpretability The top-down and bottom-up distances, which *can* be computed efficiently, are defined in terms of disruptive edit operations that may occur at any point in the tree, irrespective of the depth of the vertex they occur on. We find this unsatisfactory,

[1] This is particularly true for *edit* distances: edit operations for graphs could create loops in a tree, which would lead to problems in interpretability.

[2] T_1 and T_2 are isomorphic if there exists a tree isomorphism between them: a bijection of the nodes which preserves the edges and maps the root of T_1 to the root of T_2.

especially for applications related to taxonomies, where, for example, substituting/deleting a child of the root has a dramatic effect on the taxonomy, while that is not accounted for in the measure.

As noted above, there exist efficient methods for computing the consensus between sets of trees. We discovered during the development of our own algorithm however that it produces a consensus tree as a natural part of the procedure, which is equivalent to the strict consensus.

3. SuMoTED — Subtree Moving Tree Edit Distance

This section introduces the first main contribution of the paper: a novel distance measure between trees, named SuMoTED (Subtree Moving Tree Edit Distance). For the sake of mathematical rigour, we formalise this distance in terms of vertex-labelled graphs and supply all main proofs. The high-level concept however may be understood by defining a *local move* on a tree as a small 're-wiring' of the edge connecting a vertex to its parent. The set of all possible local moves from a source tree then forms an extremely large graph, with the edit distance being the shortest path from source to target tree over this graph. We begin by introducing some notation and basic definitions.

Throughout, $T(V, E, R)$ will represent a tree with directed edges $(v, w) \in E \subseteq V \times V$ over a set of vertices V, and root $R \in V$. We denote the set of all possible trees with a given vertex set V and root R as $\mathcal{T}_{V,R}$. We will also use T and $\mathcal{T}$ for brevity when V, E, R are clear from the context. Note that $|\mathcal{T}_{V,R}| = n^{n-2}$ with $n = |V|$ (Caley's number, [15]). To simplify notation we also define the *parent function* of a vertex:

Definition 1 (Parent function)**.** The **parent function** of a tree $T(V, E, R)$ on a vertex v, denoted $\mathrm{Pa}_T(v)$ is defined as $\mathrm{Pa}_T(v) = w \Leftrightarrow (w, v) \in E$.

We also define a particular type of tree in which each vertex is the child of the root, known as the *bush over V with root R*.

Definition 2 (Bush)**.** A tree $T(V, E, R)$ is called the **bush over *V* with root *R***, denoted $B(V, R)$ if the edge set E is such that $E = \{(R, v) | v \in V \setminus \{R\}\}$.

3.1. Tree edit distances

This subsection introduces our proposed method for computing the edit distance between trees[3]. Quantifying the distance between arbitrary trees $T, T' \in \mathcal{T}$ directly is challenging. However, for certain pairs of extremely similar trees, such a quantification is often intuitive—for example, if there is exactly one edge which differs. Denoting the set of (ordered) tree pairs (T, T') between which this distance can be quantified as $\mathcal{E}$, and the corresponding distance as the output of a real-valued *weight function* $W(T, T')$, we can define a weighted directed graph over the set of all trees $\mathcal{T}$, which we call an *edit graph*:

Definition 3 (Edit graph, local tree edit)**.** Let $\mathcal{E} \subseteq \mathcal{T} \times \mathcal{T}$ represent a set of ordered tree pairs, with a positive and finite real-valued weight function $W : \mathcal{E} \to \mathbb{R}^+$ mapping each pair $(T, T') \in \mathcal{E}$ onto a weight $W(T, T')$. Then, the weighted graph $G(\mathcal{T}, \mathcal{E}, W)$ will be referred to as the **edit graph**. The operation of changing a tree T into a neighbour of this tree in the edit graph, will be referred to as a **local tree edit** applied to T.

Clearly, to construct this graph, weights representing distances need to be specified only for the pairs of nearby trees in $\mathcal{E}$. Yet, it allows one to define a distance $d(T, T')$ for *any* pair of trees (T, T') as the *tree edit distance*—the weight of the shortest path between the vertices representing T and T' in the edit graph:

Definition 4 (Tree edit distance)**.** Given an edit graph $G(\mathcal{T}, \mathcal{E}, W)$ over all trees $\mathcal{T}$, the **tree edit distance** $d : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ between T and $T'$is defined by:

$$d(T, T') = \min_{n, T_0, T_1, \ldots, T_n} \sum_{i=1}^{n} W(T_{i-1}, T_i),$$

$$\text{s.t.} \quad n \in \mathbb{Z}^+, \quad (T_{i-1}, T_i) \in \mathcal{E} \ \forall i, \quad T_0 = T, \quad T_n = T',$$

if this problem is feasible, and ∞ otherwise.

Criteria for a good edit graph. The challenge in defining a good tree edit distance is twofold: deciding which trees are not too distant (which amounts to specifying $\mathcal{E}$), and deciding how distant precisely these trees are (specifying W). Intuitively, we wish the edit graph to satisfy the following two criteria:

1. Symmetry: if $(T, T') \in \mathcal{E}$, then also $(T', T) \in \mathcal{E}$, and $W(T, T') = W(T', T)$.
2. Connectedness: all $T, T' \in \mathcal{T}$ are connected by a path in the edit graph. This means that the distance between any pair of trees is finite: $d(T, T') < \infty, \forall T, T' \in \mathcal{T}$.

Properties of tree edit distances Any tree edit distance d, satisfying symmetry and connectivity has two appealing properties:

Proposition 1. *d is a distance metric over the set of trees $\mathcal{T}$.*

Proof. Non-negativity and identity of indiscernibles follow from the definition. Symmetry follows from the symmetricity of the edit graph. Finally, the triangle inequality follows directly from the fact that for any $T, T', T'' \in \mathcal{T}$, the shortest path between T and T'' is at most as long as the sum of the distances of the shortest paths between T and T', and T' and T'' □

This proposition has an important immediate corollary regarding bushes which we will later rely on:

Corollary 1. *For any $T, T' \in \mathcal{T}_{V,R}$:*

$$d(T, T') \leq d(T, B(V, R)) + d(T', B(V, R)).$$

Proof. From symmetricity and the triangle inequality on d. □

In Subsection 3.2 and 3.3 we discuss how the criteria of symmetricity and connectedness of the edit graph can be realised. Note that for simplicity in these sections we assume the label sets of the source and tree are identical—this assumption will be relaxed in Subsection 3.5.

3.2. Local moves as tree edits

We will now define the set of edges $\mathcal{E}$ of the edit graph in terms of a tree operation we refer to as a *local move*, which amounts to deleting the edge between a vertex and its parent, and adding an edge between either the edge's grandparent, or one of its siblings:

Definition 5 (Local move)**.** A **local move** on a tree $T(V, E, R)$ is an operation that changes it into a tree $T'(V, E', R)$ with $E' = (E \setminus \{(\mathrm{Pa}_T(v), v)\}) \cup \{(w, v)\}$ where w is either $\mathrm{Pa}_T(\mathrm{Pa}_T(v))$—the grandparent of v—or w is a sibling of v. A local move is called *upward* when w is $\mathrm{Pa}_T(\mathrm{Pa}_T(v))$, and *downward* otherwise.

Local upward and downward moves are illustrated in Fig. 2. Local moves satisfy both our desired criteria:

Proposition 2. *With local moves as edits, and a weight function which assigns equal weights to upward and downward moves, the edit graph $G(\mathcal{T}, \mathcal{E}, W)$ is symmetric.*

[3] Note that Tai [24] coined their method *tree edit distance*. In this paper we use this term to refer to a class of methods that quantify the distance between trees in terms of edit operations, rather than this specific method.

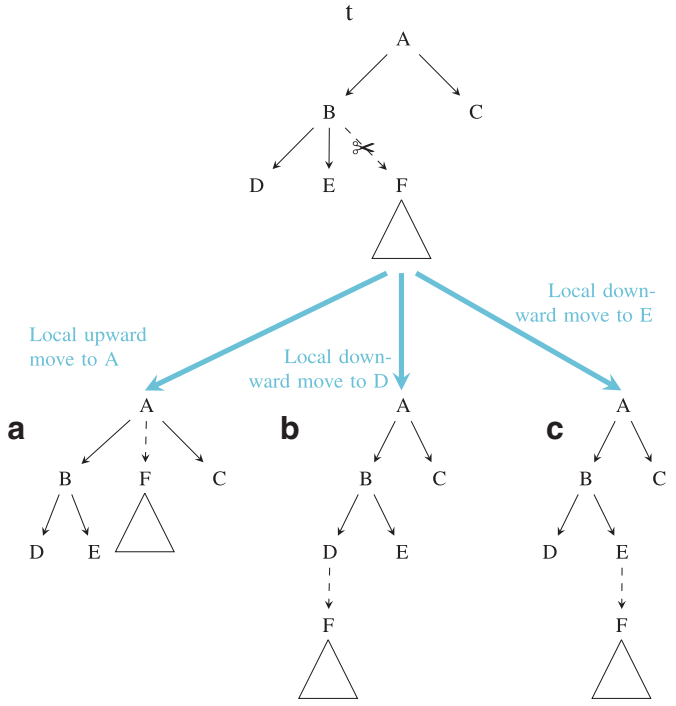

Fig. 2. Local move operations on vertex *F*. In (a), the vertex F and all its descendants (the triangle) are moved from being a child of B to be a child of A (F's grandparent). (b): F and its descendants become a child of one of F's siblings, D. Finally, in (c), F moves to be a child of its other sibling E.

Proof. After a local upward move, the parent of the child vertex becomes its sibling. Thus, this type of move can be undone by rewiring the affected edge to a sibling. Similarly, after a downward move, the parent of the child vertex becomes its grandparent. Thus, this local move can be undone by rewiring the rewired edge to its grandparent. Since the costs of moves are equal, both operations have equal cost and G is therefore symmetric. □

Proposition 3. *With local moves as edits, the edit graph* $G(\mathcal{T}, \mathcal{E}, W)$ *is connected.*

Proof. Any tree $T(V, E, R)$ is connected to the bush $B(V, R)$. Indeed, an upward move on a tree decreases the sum of the depths of the vertices in the tree by an amount of at least 1. Furthermore, any tree that is not a bush will have a vertex to which an upward move can be applied. Thus, we can successively apply upward moves to T and be sure that eventually the bush $B(V, R)$ will be reached. From the symmetry of the edit graph, this also implies that any tree can be reached from the bush by a sequence of downward moves. Put together, there exists a path between any arbitrary pair of trees $T(V, E, R)$ and $T'(V, E', R)$, namely one that passes via the bush $B(V, R)$. □

3.3. The weight function for local moves

Having defined the set of edges $\mathcal{E}$ of the edit graph as those between any pair of trees that are separated by one *local move*, we now need to define the weight function $W(T, T')$ of such an edge. A simple approach would be to set $W(T, T') = 1$ for any pair $(T, T') \in \mathcal{E}$. However, note that a local move allows for arbitrarily large groups of vertices to move up or down the tree quickly and cheaply. For example, in Fig. 2(*a*), we see that all the descendants of F have been moved up to be children of the root vertex at no extra cost. To account for the varying number of vertices that are affected, we therefore define the weight function as the total size of the subtree with root v. This means that the weight is equal to 1 in the case where v is a leaf, and equal to 1 more than the number of descendants of v in general.

3.4. A normalised similarity measure

A weakness of our proposed measure is that it will tend to be larger for larger $|V|$, such that distances between pairs of trees of different sizes are hard to compare. Recall that two trees can always be reached using local moves which passes through the bush—this is therefore an upper bound on the distance between two trees. In order to be able to compare scores of trees of different sizes, we propose a normalisation scheme in which we divide the distance between two trees $d(T, T')$ by the sum of the distances from T, T' to the bushes: $d(T, B(V, R)) + d(T', B(V, R))$. Often it is also more convenient to use similarity measures, so we define the normalised similarity between $T, T' \in \mathcal{T}_{V,R}$ as:

$$s(T, T') = 1 - \frac{d(T, T')}{d(T, B(V, R)) + d(T', B(V, R))} \in [0, 1] \tag{1}$$

3.5. An extension to trees with different label sets

So far, we have assumed that the trees we compare are label-set consistent (meaning that the number of vertices and number of labels coincide). When $T \in \mathcal{T}_{V,R}$ and $T \in \mathcal{T}_{V',R}$ with $V \neq V'$, we generalise the tree edit distance metric d as follows: Add each vertex $v \in V \setminus V'$ as a direct child of the root R in T, yielding $T_+ \in \mathcal{T}(V \cup V', R)$. Similarly, add each vertex $v \in V' \setminus V$ as a direct child to the root R in T', yielding $T'_+ \in \mathcal{T}(V \cup V', R)$. We then define the distance $d(T, T')$ as $d(T_+, T'_+)$. Placing 'unseen' vertices as children of the root is conducted as we have no prior information on any better position to place them. Note that if we consider this step to be preprocessing, the (un-normalised) distance maintains its metric property. An example of the optimal set of operations to convert a source tree into a target tree (as well as to the normalising bush) is shown in Fig. 3, and is also available animated interactively online at http://www.interesting-patterns.net/ds4dems/sumoted-demo/. Python code to compute SuMoTED is available online[4].

4. An efficient algorithm to compute SuMoTED

Computing SuMoTED amounts to finding the shortest path from the source tree to the target tree $(T, T') \in \mathcal{E}$ in the edit graph $G(\mathcal{T}, \mathcal{E}, W)$. Effective algorithms (polynomial complexity in number of vertices and edges) for computing the shortest path between a given pair of vertices in a graph exist [10]. However, the graph in our case is far too large for such an approach to be feasible (recall from earlier: $|\mathcal{T}_{V,r}| = |V|^{|V|-2}$). Remarkably, we have discovered a fast algorithm for computing SuMoTED between any pair of trees that is polynomial (quadratic) *in the size of the trees*, rather than in the size of the edit graph. The current section outlines this algorithm, which is based on the following theorem:

Theorem 1. *Given trees* $T, T' \in \mathcal{T}$, *the shortest path in the edit graph between* T *and* T' *is equally as long as the shortest path that consists of a sequence of local upward moves, followed by a sequence of local downward moves.*

The proof of the theorem rests on the following Lemma:

Lemma 1. *Let* $(T_0, \ldots, T_n)$ *be a shortest path of trees between* T_0 *and* T_n. *Assume that there exists* $0 < i < n$ *such that* T_i *is reached from* T_{i-1} *by a downward move, and* T_{i+1} *is reached from* T_i *by an upward move. Then it is always possible to replace the subpath* (T_{i-1}, T_i, T_{i+1})

[4] https://github.com/mattmcvicar/SuMoTED

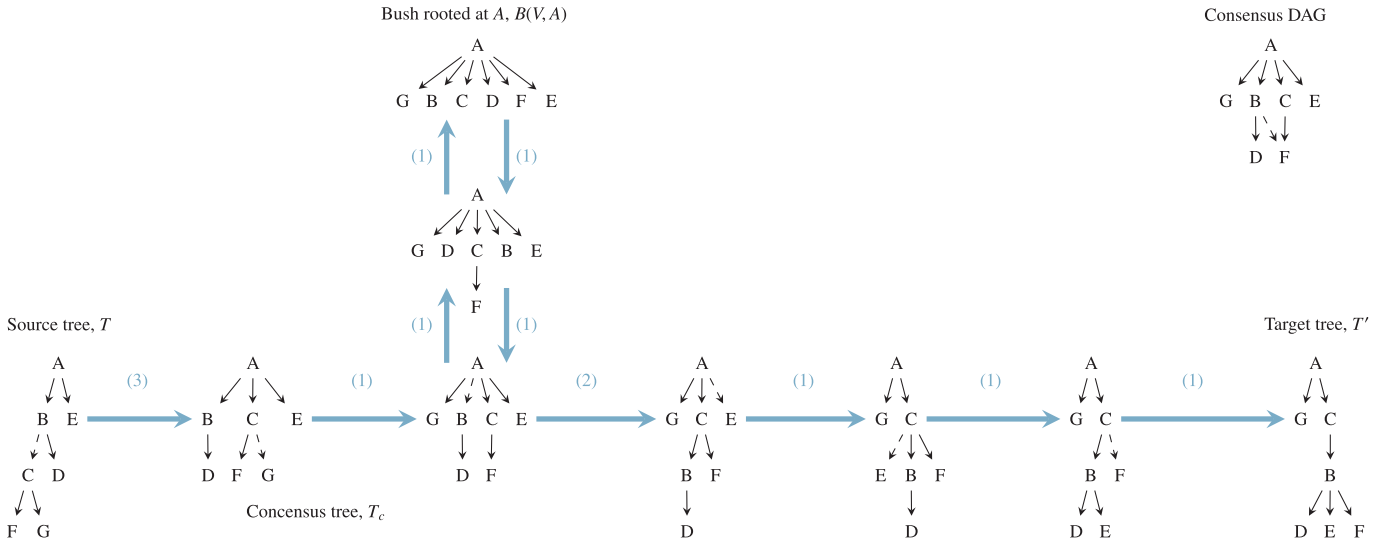

Fig. 3. Example of our proposed edit distance $d(T, T')$. Intermediate trees are shown between blue arrows, together with the cost of edit. The severed edge for each tree in the bottom row are shown as dashed arrows. This (optimal) overall path has length $3+1+2+1+1+1=9$, normalised similarity $1-(9/(9+4))\approx 0.31$. The consensus DAG is shown in the top-right, from which we have generated the tree using solid arrows. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

for another subpath between T_{i-1} and T_{i+1} of equal cost that consists of a sequence of upward moves followed by a sequence of downward moves.

Proof. Let $(Pa_{T_{i-1}}(v), v)$ be the edge involved in the downward move on T_{i-1}, and $(Pa_{T_i}(w), w)$ the edge involved in the upward move on T_i. These moves can simply be swapped without altering the resulting tree T_{i+1} and the total cost, as long as $Pa_{T_{i-1}}(v) \neq Pa_{T_i}(w)$. When $Pa_{T_{i-1}}(v) = Pa_{T_i}(w)$, on the other hand, the two moves can be replaced with two upward moves followed by one downward move. Referring to $Pa_{T_{i-1}}(v) = Pa_{T_i}(w)$ as u, and to the parent of u (in T_{i-1}, T_i, as well as T_{i+1}) as z, these moves should be:

1. An upward move of edge (u, v), replacing (u, v) with (z, v).
2. An upward move of edge (u, w), replacing (u, w) with (z, w).
3. A downward move of edge (z, v), replacing (z, v) with (w, v).

It is easy to verify that these three moves have the same cost as the total cost of the original downward and upward moves. □

Proof of Theorem 1. Given any optimal path, iteratively apply Lemma 1 until no more downward moves can be found that are followed by an upward move. □

Theorem 1 implies that the edit distance $d(T, T')$ can be expressed in terms of a *consensus tree* T_c:

Definition 6 (Consensus tree)**.** A **consensus tree** for two trees $T, T' \in \mathcal{T}$ is a tree $T_c \in \mathcal{T}$ that can be reached from T as well as from T' using local upward moves only.

By symmetry, Theorem 1 can be rephrased as saying: a shortest path from T to T' exists in the edit graph that consists of upward moves to T_c, followed by downward moves to T'. Theorem 2 shows that the distance between T and T_c which can be reached using upward moves only from T depends only on T, T_c:

Theorem 2. *Define a partial order between all vertices in a given tree T as follows:*[5]

$$P_T = \{(v, w) \mid w \text{ is a descendant of } v \text{ in } T\},$$

where we consider v to be a trivial descendant of itself. The total distance of any path $(T_0, \ldots, T_c)$ in the edit graph for which T_i is reached by a local upward move from T_{i-1} for all i, is given by $d(T_0, T_c) = |P_{T_0}| - |P_{T_c}|$.

Proof. When an upward move on vertex v is applied to T_i to yield T_{i+1}, the number of pairs removed from P_{T_i} is equal to the size of the subtree rooted at v. Indeed, the only change is that all descendants of v (including v itself) no longer have $\mathrm{Pa}_{T_i}(v)$ as an ancestor. Thus, $d(T_i, T_{i+1}) = |P_{T_i}| - |P_{T_{i+1}}|$. For a sequence of local upward moves $(T_0, T_1, \ldots, T_c)$, this means that the total path length in the edit graph is $\sum_{i=1}^{c} d(T_{i-1}, T_i) = \sum_{i=1}^{c} |P_{T_{i-1}}| - |P_{T_i}| = |P_{T_0}| - |P_{T_c}|$. □

The following is a direct consequence of the definition of SuMoTED and the previous theorem:

Corollary 2. *Given $T, T' \in \mathcal{T}$: $d(T, T') = \min_{T_c} |P_T| + |P_{T'}| - 2|P_{T_c}|$, subject to T_c being a consensus tree.*

Thus, to compute the $d(T, T')$, all that is needed is to compute the size of the largest partial order P_{T_c} over all consensus trees T_c for T and T'. Clearly, $P_{T_c} \subseteq P_T \cap P_{T'}$, and when the Hasse diagram [3] of $P_T \cap P_{T'}$ is a tree, the optimal $P_{T_c} = P_T \cap P_{T'}$. However, in general, the Hasse diagram of $P_T \cap P_{T'}$ is a DAG. The task of maximizing $|P_{T_c}|$ then amounts to finding a subtree of this DAG representing the largest possible partial order. This optimal consensus tree can be found via a layer-assignment algorithm known as the *Longest Path Algorithm*, which has linear time complexity [19]. Briefly, given $P_T \cap P_{T'}$ this algorithm proceeds as follows:

- Initialise $T_c(V_c, E_c, r)$ with $V_c = \{r\}$, $E_c = \{\}$.
- Iterate: For each vertex v for which all w with $(w, v) \in P_T \cap P_{T'}$ are in T_c, identify the deepest such vertex w in T_c, and insert v into V_c and (w, v) into E_c.

This algorithm ensures that each vertex is maximally deep in the tree, such that the transitive closure of the tree is as large as possible. The detailed proof works by induction: It is true for the root, and given that it is true for a partial tree already built, it is also true for the new vertices and edges added in each iteration. It can be verified that the overall computational complexity of computing SuMoTED for $T, T' \in \mathcal{T}_{V,r}$ is $O(|V|^2)$.

[5] P_T can equivalently be defined as the reflexive transitive closure of the edge set E for a tree T(V, E, R).

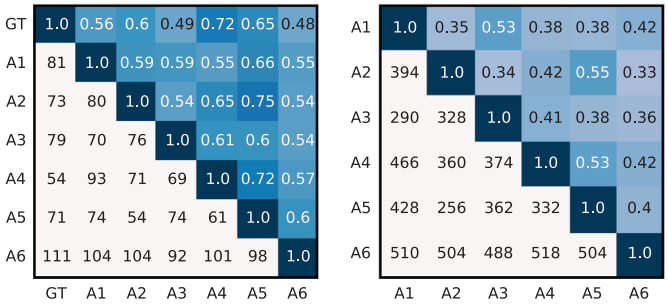

Fig. 4. Edit cost (below diagonal) and normalised similarity (above diagonal) between the ground truth Deezer taxonomy and annotators (*A*1–*A*6) and for the Ground Truth. Right: for the ReverbNation dataset.

5. Experiments

In this section we conduct a case study, applying SuMoTED to three hierarchical datasets which describe popular music genres. The genre of a song is a high-level musical attribute frequently used for music organisation, playlisting, searching, and recommendation. Often, songs are tagged with a set of labels which are hierarchically arranged into a musical genre taxonomy. Unfortunately, different musical experts and professional music services use very different sets of genre labels in their categorisation schemes. Even when these label sets overlap, they are often structured differently which complicates their use for the applications listed above. To investigate how SuMoTED could be used to analyse these kinds of data, we experimented with three datasets: a small dataset where the "true" hierarchy is known (5.1), a medium-size dataset with no existing ground truth (5.2), and a large-scale dataset consisting of commercially-used music genre hierarchies where the label sets do not coincide (5.3). Finally, we investigate the scalability of our method in 5.4.

5.1. Deezer dataset

The music genre taxonomy used by the web-based music streaming service Deezer was used in these experiments, featuring $n = 101$ genres. We asked 6 annotators (referred to as *A*1–*A*6 hereafter) to construct a taxonomy from these genres without consulting the reference annotation or each other. We then computed the SuMoTED (via corollary 2) and normalised similarity (Eq. 1) between each pair of annotations. Results can be seen on the left of Fig. 4. From this Figure, we see that the normalised similarities are all equal to unity when the taxonomies are equal (diagonal entries), as expected. We see that annotator *A*4 was the closest to the Deezer reference (normalised similarity 0.72), and that annotators *A*2 and *A*5 were the most similar to each other (0.75). Annotator *A*5 has the highest mean similarity to other taxonomies (0.66), meaning *A*5 could be considered the 'centre of mass' of the set of references. We were also interested in the overlap between annotations, so we computed the Hasse diagram of the intersection of all annotations in Fig. 5. Interestingly, this Figure shows that there was no consensus as to placement of *rock* and its descendants in the taxonomy. For example, *A*2 listed *alternative* as a child of *rock*, whereas in the Deezer reference this relationship was reversed.

5.2. ReverbNation dataset

From an existing project, we had 251 unique genre labels stored from a set of over 50, 000 independent UK music artists from ReverbNation.com. As before, *A*1–*A*6 were asked to make a taxonomy from this larger dataset. The annotator similarities are shown in the right matrix of Fig. 4. From this matrix, we see that similarities are generally lower—we found that this was a result of the increased depth of some of the taxonomies specified in the ReverbNation dataset. For example, *A*4 had one vertex of depth 6: *Music* → *electronic* → *edm* → *uk* → *dnb* → *breakbeat* → *breakcore*—two levels deeper than any vertex in 5.1. Interestingly, in both sets of experiments *A*2 and *A*5 had the highest similarity, followed by *A*4 and *A*5.

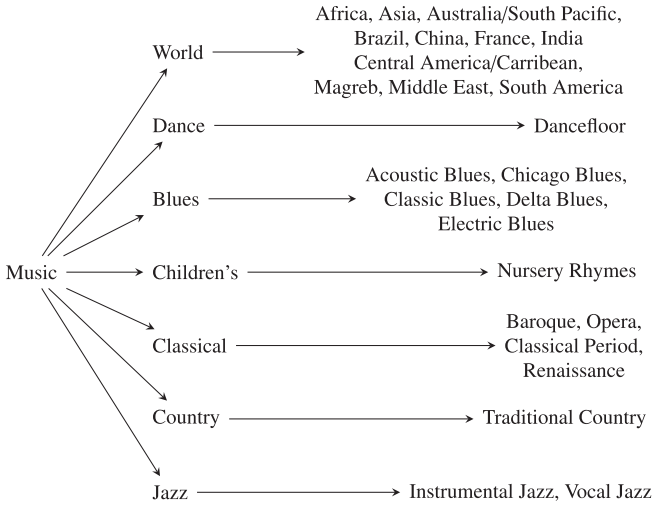

Fig. 5. Hasse diagram for the intersection of the Deezer taxonomies. Genres which were found to be a child of *Music* with no further children in common are omitted for brevity.

Table 1
Comparison of existing taxonomies used in industry. Above diagonal entries show normalised distance, below show Jaccard index.

		Normalised similarity			
		A(allmusic)	D(eezer)	iT(unes)	W(iki)
Jaccard	A	–	0.01	0.05	0.07
	D	0.04	–	0.12	0.01
	iT	0.16	0.16	–	0.06
	W	0.15	0.03	0.10	–

5.3. Commercial datasets

We sourced four genre hierarchies for use in these experiments: the Deezer dataset used above ($n = 101$ genres), Allmusic ($n = 1062$), iTunes (340), and Wikipedia (730). As the label sets of these taxonomies did not coincide, we computed the Jaccard similarity of the label sets to investigate how similar they were. See Table 1 for these results. We see from this Table that the similarities between taxonomies (below-diagonal) are low in magnitude. These values highlight and quantify the huge discrepancies between the choice of genre labels companies use when constructing a taxonomy. Above-diagonal entries are also close to 0.0, indicating that there is little similarity between industrially-used music genre hierarchies—something speculated about in previous work [2] but never quantified. Given that these similarities were close to zero, we wondered if they were significantly larger than random. To assess this, we conducted a *permutation test*: for each tree we generated a number of trees with identical topology but randomly permuted labels. For given similarity between 'true' trees *S*, an empirical *p*-value was then computed:

$$\hat{p} = \frac{|\text{permuted trees with similarity } \geq S| + 1}{|\text{permuted trees}| + 1}$$

However, in all our experiments (Subsection 5.1–5.3), we never found a random tree pair with similarity greater than or equal to

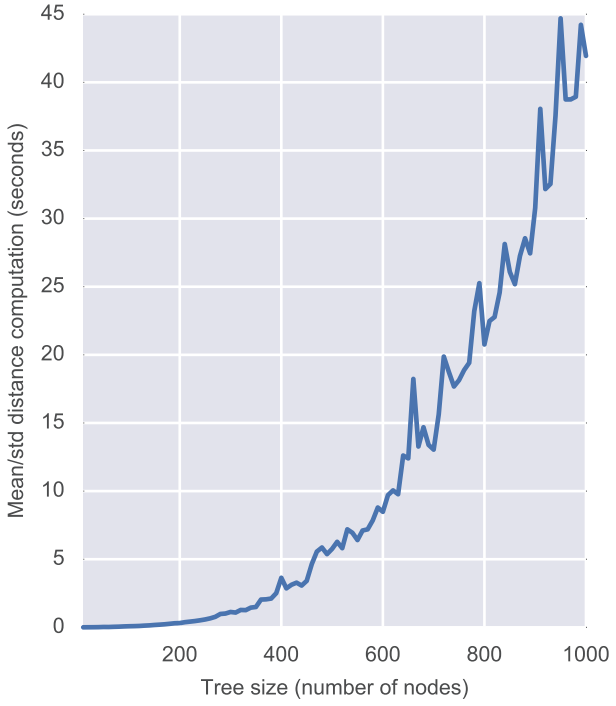

Fig. 6. Scalability of our algorithm in practice. 10 Random trees with n (horizontal axis) nodes were created and their average distance computation time (vertical axis) was measured.

the true tree pairs, for 99 randomly-generated trees. Results were in fact generally around 3 orders of magnitude lower. This resulted in empirical p-values of 0.01 and indicates that all the similarities we computed were significantly more self-similar than random taxonomies at the 1% level.

5.4. Scalability experiments

We were interested in seeing how our algorithm scales with input size. To this end, we computed the time required to compute the distance between several random trees with a fixed number of nodes. Random trees with n nodes labelled $\{1, \ldots, n\}$ with fixed root 1 were created as follows: labels $2, \ldots, n$ were first randomly permuted, and then attached to one of the existing nodes in the tree until all labels were exhausted.

We created 10 trees for each n using the above procedure, computed their pairwise distance, and recorded their average computation time. Experiments were conducted in the Python programming language on a laptop with 2.6 GHz Intel Core i5 processor and 8GB 1600 MHz DDR3 memory running OSX El Capitan 10.11.2. Results can be seen in Fig. 6. From this Figure, we see that our method scales reasonably well to large tree sizes. We can compute the distance between two trees with 1000 nodes (consistent with industrial datasets) comfortably in under 1 min. The quadratic trend seen in the Figure is consistent with the theoretical result presented in Section 4. The code was implemented in the most intuitive way possible, with no particular optimisation for data structures or subroutines—further improvements in time complexity may therefore improve the results seen in Fig. 6. Recall that the implementation is available online.

6. Conclusions and future work

We have presented a novel distance between trees, called SuMoTED, defined as an edit distance via *local moves*. SuMoTED has several appealing properties: it is a metric distance in the unnormalised setting, is computable in quadratic time, and is applicable to trees with different label sets. As a case study, we used this distance metric to investigate the consistency between annotators and existing music genre taxonomies, finding high similarity between human-generated taxonomies in the case of small label sets. We were also able to construct consensus annotations using our method, which gave musical insight into agreed-upon hierarchical genre relationships amongst annotators. Besides the study of commonalities and differences between various trees (such as taxonomies), SuMoTED is ideally suited for more advanced analyses such as clustering trees. Furthermore, it can be used to quantify the performance of methods designed for inferring taxonomies from data.

We focussed on music genre taxonomies in the current paper, but are excited by the prospect of using our method to compute taxonomy similarities in some of the domains listed in the introduction of this paper. For example, we could use SuMoTED to investigate the similarities between biological or textual trees. A further idea for future research is the investigation of information cascades [12], where trees are formed by information flowing through a network. Also, we would like to investigate if our method can be used to construct and evaluate methods which *infer* a taxonomy from data, as this could be useful is assessing how reliable such a taxonomy is. Finally, it appeared that there are no results on the complexity of existing tree distance measures for the case when all node labels are unique. This would also be worth investigating.

Acknowledgements

This work was sponsored by EPSRC grant number EP/M000060/1 and the ERC Consolidator Grant FORSIED (project reference 615517).

References

[1] E.N. Adams III, Consensus techniques and the comparison of taxonomic trees, Syst. Zool. 21 (4) (1972) 390–397.
[2] J. Aucouturier, F. Pachet, Representing musical genre: a state of the art, J. New Music Res. 32 (1) (2003) 83–93.
[3] K. Baker, P.C. Fishburn, F. Roberts, Partial orders of dimension 2, Networks 2 (1) (1972) 11–28.
[4] J.L. Balcázar, A. Bifet, A. Lozano, Mining frequent closed rooted trees, Mach. Learn. 78 (1-2) (2010) 1–33.
[5] P. Bille, A survey on tree edit distance and related problems, Theor. Comput. Sci. 337 (1) (2005) 217–239.
[6] F.J. Brandenburg, A. Gleißner, A. Hofmeier, Comparing and aggregating partial orders with kendall tau distances, in: WALCOM: Algorithms and Computation, Springer, 2012, pp. 88–99.
[7] K. Bremer, Combinable component consensus, Cladistics 6 (4) (1990) 369–372.
[8] Y. Chi, R. Muntz, J. Nijssen S.and Kok, Frequent subtree mining–an overview, Fundam. Inform. 66 (1-2) (2005a) 161–198.
[9] Y. Chi, Y. Xia, Y. Yang, R. Muntz, Mining closed and maximal frequent subtrees from databases of labeled rooted trees, Knowl. Data Eng. IEEE Trans. 17 (2) (2005b) 190–202.
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, et al., Introduction to algorithms, 2, MIT Press Cambridge, 2001.
[11] W.H. Day, Optimal algorithms for comparing trees with labeled leaves, J. Classif. 2 (1) (1985) 7–28.
[12] W. Galuba, K. Aberer, D. Chakraborty, Z. Despotovic, W. Kellerer, Outtweeting the twitterers-predicting information cascades in microblogs., WOSN 10 (2010) 3–11.
[13] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Anal. Appl. 13 (1) (2010) 113–129.
[14] J. Hsiang, M. Rusinowitch, Proving refutational completeness of theorem-proving strategies: the transfinite semantic tree method, J. ACM (JACM) 38 (3) (1991) 558–586.
[15] D.E. Knuth, The art of computer programming, volume 1: Fundamental algorithms. 1968, Seminumer. Algorithms 3 (1969).
[16] E. Malmi, N. Tatti, A. Gionis, Beyond rankings: comparing directed acyclic graphs, Data Min. Knowl. Discov. 29 (5) (2015) 1–25.
[17] T. Margush, F.R. McMorris, Consensusn-trees, Bull. Math. Biol. 43 (2) (1981) 239–244.
[18] T.A. McMahon, R.E. Kronauer, Tree structures: deducing the principle of mechanical design, J. Theor. Biol. 59 (2) (1976) 443–466.

[19] K. Mehlhorn, Data Structures and Algorithms: Graph Algorithms and NP-Completeness, Vol. 2, Springer-Verlag, Heidelberg, Germany, 1984.
[20] G. Nelson, Cladistic analysis and synthesis: principles and definitions, with a historical note on adanson's familles des plantes (1763–1764), Syst. Zool. 28 (1) (1979) 1–21.
[21] S.M. Selkow, The tree-to-tree editing problem, Inf. Process. Lett. 6 (6) (1977) 184–186.
[22] D. Shasha, J.T. Wang, S. Zhang, Unordered tree mining with applications to phylogeny, in: Data Engineering, 2004. Proceedings. 20th International Conference on, IEEE, 2004, pp. 708–719.
[23] R.E. Stobaugh, Chemical substructure searching, J. Chem. Inf. Comput. Sci. 25 (3) (1985) 271–275.
[24] K.-C. Tai, The tree-to-tree correction problem, J. ACM (JACM) 26 (3) (1979) 422–433.
[25] G. Tzanetakis, P. Cook, Musical genre classification of audio signals, IEEE Trans. Speech Audio Process. 10 (5) (2002) 293–302.
[26] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM J. Comput. 18 (6) (1989) 1245–1262.