Homeomorphic Alignment of Weighted Trees

Benjamin Raynal, Michel Couprie, Venceslas Biri

Homeomorphic Alignment of Weighted Trees

Benjamin Raynal[a], Michel Couprie[a], Venceslas Biri[a]

[a] *Université Paris-Est,Laboratoire d'Informatique Gaspard Monge, Equipe A3SI*
UMR 8049 UPEMLV/ESIEE/CNRS

Abstract

Motion capture, a currently active research area, needs estimation of the pose of the subject. For this purpose, we match the tree representation of the skeleton of the 3D shape to a pre-specified tree model. Unfortunately, the tree representation can contain vertices that split limbs in multiple parts, which do not allow a good match by usual methods. To solve this problem, we propose a new alignment, taking into account the homeomorphism between trees, rather than the isomorphism, as in prior works. Then, we develop several computationally efficient algorithms for reaching real-time motion capture.

Key words: Graphs, homeomorphism, alignment, matching algorithm

1. Introduction

1.1. Motivation

Motion capture without markers is a highly active research area, as shown by Moeslund et al. [1]: between 2000 and 2006, more than 350 papers on this topic were published. Motion capture is used in several applications which have different aims and constraints:

3D models animation, for movies FX or video games for example, requests an highly accurate model, but does not need real-time computation (offline video processing is acceptable).

Real-time interaction, for virtual reality applications, requests a fast computation, at the price of a lower accuracy.

This paper is placed in the context of real-time interaction.

Moeslund et al. describe in a previous work [2] the different steps of motion capture.

The first step (called *initialization step*) consists of finding the initial pose of the subject, represented here by a 3d shape (visual hull) constructed using a multi-view system with an algorithm of Shape From Silhouette [3].

Email addresses: `raynal@univ-mlv.fr` (Benjamin Raynal), `coupriem@esiee.fr` (Michel Couprie), `biri@univ-mlv.fr` (Venceslas Biri)

Most algorithms of 3D pose estimation use a manually initialized model, or ask the subject to move successively the different parts of his/her body [4], but several automatic approaches have been developed, using an a priori model. This a priori model can approximate different characteristics of the subject:

Kinematic structure: A structure containing a fixed number of joints, with specified degrees of freedom, and limb lengths.

Shape: A generic humanoid model, represented by simple shape primitives [5].

Appearance: The texture of subject surface.

This kind of complex a priori model is difficult to match with real data, and needs to be adapted to each subject (especially in the case of appearance).

In shape matching, a common approach consists of using surface (or contour, in case of 2D shapes) information, via curvature and distance to centroid information [6, 7, 8]. This kind of approach can not be used in our case for several reasons: it implies a complex a priori model, curvature is variable for articulate shape, and the visual hull can have a very noisy surface, due to the method of acquisition.

1.2. Our approach

Two of the most preserved characteristics of the real shape by its visual hull reconstruction are its topology and the distances between the different parts of the shape. A useful tool for representing these characteristics is the skeleton of the shape.

A lot of approaches using the skeleton of a shape have been developed. In motion capture research area [9, 10, 11], the best time obtained for finding the initial pose is around one second [10], which is too slow, even for interactive time interaction. In 2D shape matching research area, skeleton is a common tool of representation and comparison [12, 13]. A widely used method consist in comparison of shock graphs [14], built from both the skeleton and the radius distance of the shapes. However, even if this method can be applied in the case of 3D shapes [15], it is not interesting in our case, because the radius distance of the visual hull can be noisy.

For our purpose, the skeleton is still too complex, and we do not need all the involved information. It can be observed that only a small part of the skeleton points are interesting: the ending points and the intersection points. In addition to the position of these points, we need to know how they are linked together, and what is the length of the skeleton branch between them.

Considering these observations, we can represent the skeleton by a unrooted tree (called the *data tree*): the vertices represent the ending or intersection points, and the edges represent the links between the points. In addition, we give a weight to each edge, corresponding to the geodesic distance between the points involved.

Then, our a priori model is also an unrooted weighted tree (called the *pattern tree*), where vertices represent the different parts of the shape (head, torso,

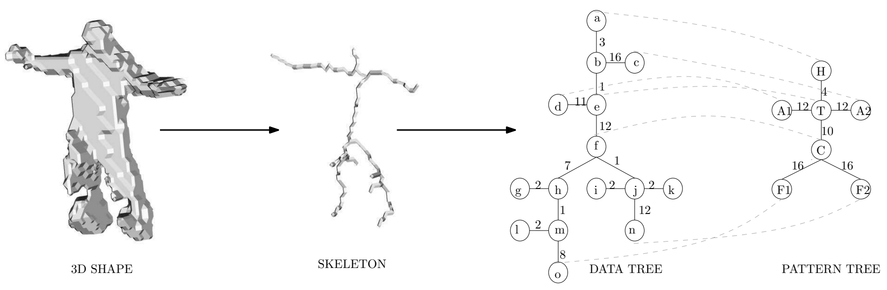

Figure 1: Example of data tree acquisition and expected alignment with pattern (model) tree. From the right to the left: the original 3D shape, its skeleton, the data tree computed from the skeleton, and the pattern tree. The numbers represent the weights of the edges (i.e. the geodesic distances in the skeleton for the data tree, and the expected ones for the pattern tree).The grey dashed lines represent the expected alignment between the data tree and the model.

crotch, hands and feet), and each edge represents the link between this parts, associated to a weight, representing the distance between two parts. It can be seen as a very simplified kinematic structure, without definition of degrees of freedom.

Our approach to find the initial pose of the subject is therefore to find the best alignment between the pattern tree and the data tree, that is to say, the matching which involves as few modifications as possible, to transform the data tree in the pattern tree (see Fig.1 for an example of our complete pipeline, and expected alignment).

As we know which part of the shape correspond to each vertex of the pattern tree, and which are the 3D coordinates of each vertex of the data tree, the alignment will give the position of each part of the 3D shape.

1.3. Problems

Several kinds of noise and deformities can appear in the data tree:

Ghosts limbs: Due to the reconstruction from silhouettes, parts of space cannot be carved, resulting in "limbs" of the object which do not exist on the real model. Our method must be accurate enough to distinguish these ghosts limbs from real ones.

Spurious branches: Due to the skeletonization algorithm and to the amount of noise of the shape surface, branches without important topological signification can appear on skeleton. For example, in Fig. 1, the edges $\{g, h\}, \{l, m\}, \{i, j\}, \{j, k\}$ are spurious branches. Our method must be robust enough to work on data trees with consequent amount of spurious branches.

Useless vertex: Vertices with exactly two neighbors are not useful to describe the topology of a shape, and then uselessly split an edge (and its weight) into two parts, making difficult a good matching. This kind of vertices can appear when removing spurious branches or ghosts limbs. For example, in Fig. 1, vertices j, k, m are useless after spurious branches deletion. Our method must be able to match two edges joined by this kind of vertex, with a unique edge.

Splitted vertex: Vertices with more than three neighbors in the pattern tree can correspond to a cluster of vertices linked by weakly weighted edges in the data tree, due to the skeletonization algorithm. For example, in Fig. 1, vertex T of pattern tree matches with vertices b and e in data tree. Our method must be able to match them.

1.4. Our contribution

Approaches found in the literature (see Sect. 3) do not permit to achieve a robust matching, with respect to all these perturbations. However, some existing edit-based distance, the alignment distance [22], is an interesting way to solve the problem of splitted vertices, and it preserves the topology during the matching. In addition, an operation described in [30], the cut operation, is specially designed for considering only a subpart of the tree. This operation is exactly what we need to avoid the problems of ghost limbs and spurious branches, which can be considered as useless parts of the data tree.

The problem of useless vertices cannot be solved by methods found in the literature. We introduce in this paper a new kind of alignment, which solves this problem by considering the homeomorphism between trees instead of the isomorphism.

This paper is organized as follow: In Sect. 2, we give the basic definitions and notations related to edge-weighted graphs that will be used in the sequel. In Sect. 3, we determine which edit-based distance of the literature is the most appropriate to base our method. In Sect. 4, we introduce our main contribution, the homeomorphic alignment. We also introduce algorithms to compute it efficiently for rooted trees, as well as unrooted trees. In Sect. 5, we show how to use the cut operation [30] with homeomorphic alignment. Finally, in Sect. 6, we show the results of different experimentations, and a comparison between our homeomorphic alignment distance and the classical alignment distance.

2. Basics notions and notations

2.1. Undirected graphs

An *undirected graph* is a pair (V, E), where V is a finite set, and E a subset of $\{\{x, y\}, x \in V, y \in V, x \neq y\}$. An element of E is called an *edge*, an element of V is called a *vertex*. If $\{x, y\} \in E$, then x and y are said to be *adjacent* or *neighbors*. The set of all neighbors of x is denoted by $\mathcal{N}(x)$. The number of vertices adjacent to a vertex v is called the *degree* of v, and is denoted by $deg(v)$. Let $G = (V, E)$ be an undirected graph, and let x, y be in V, a *path* from x to y in G is a sequence of vertices $v_0, ..., v_k$ such that $x = v_0$, $y = v_k$

and $\{v_{i-1}, v_i\} \in E, 1 \leq i \leq k$. The number k is called the *length* of the path. If $k = 0$ the path is called a *trivial* path. The path is *closed* if $x = y$. The path is *simple* when no vertex occurs more than once in the sequence of vertices of the path (except possibly $x = y$). A non-trivial simple closed path in which all edges are distinct is called a *cycle*. A graph is *connected* if for all $\{x, y\} \subset V$, a path from x to y exists in G. A *tree* is a connected graph with no cycles. A simple path from x to y in a tree is unique and is denoted by $\pi(x, y)$. A graph with no cycles is called a *forest*, each of its connected components being a tree.

2.2. Directed graphs

A *directed graph* is a pair (V, A), where V is a finite set, and A is a subset of $V \times V$. An element of A is called an *arc*, an element of V is called a *vertex*. Let $G = (V, A)$ be a directed graph, and let x, y be in V, a *path* from x to y in G is a sequence of vertices $s_0, ..., s_k$ such that $x = v_0$, $y = v_k$ and $(v_{i-1}, v_i) \in A, 1 \leq i \leq k$. The *undirected graph associated* to G is the undirected graph $G' = (V, E)$, such that $\{x, y\} \in E$ if and only if $(x, y) \in A$ or $(y, x) \in A$. A vertex $r \in V$ is a root of G if for all $x \in V \setminus \{r\}$, a path from r to x in G exists. The graph G is *antisymmetric* if for all $(x, y) \in A$ such that $x \neq y$, $(y, x) \notin A$. The graph G is a *rooted tree* (with root r) if r is a root of G, G is antisymmetric and if the undirected graph associated to G is a tree. A graph, where each of its connected components is a tree, is called a *rooted forest*.

Let $G = (V, A)$ be a rooted tree. If $(y, x) \in A$, we say that y is the *parent* of x (denoted by $par(x)$), and that x is a *child* of y. The set of all children of y is denoted by $\mathcal{C}(y)$. The maximum length of a path between the root and any node is called the *height* of the tree. The vertices on the path from the root to a vertex x are called the *ancestors* of x. We denote the set of the ancestors of x by $anc(x)$.

2.3. Common definitions

Unless otherwise indicated, all the other definitions and notations in this paper are similar for the two kinds of graphs. We give them for directed graphs, the versions for undirected graphs can be obtained by replacing arcs by edges.

Two graphs $G = (V_G, A_G)$ and $G' = (V_{G'}, A_{G'})$ are said to be *isomorphic* if there exists a bijection $f : V_G \to V_{G'}$, such as for any pair $(x, y) \in V_G \times V_G$, $(x, y) \in A_G$ if and only if $(f(x), f(y)) \in A_{G'}$.

A *weighted graph* is a triplet (V, A, ω), where V is a finite set, A a subset of $V \times V$, and ω a mapping from A to $\mathbb{R}$. In a weighted tree, the weight of the unique path from x to y, denoted by $\omega(x, y)$, is the sum of the weights of all arcs traversed in the path.

In this paper, we say that two weighted graphs (V, E, ω) and (V', E', ω') are isomorphic whenever the graphs (V, E) and (V', E') are isomorphic (regardless of the weights).

The *merging* is an operation that can be applied only on arcs sharing a 2-degree vertex. The merging of two arcs (u, v) and (v, w) in a weighted graph $G = (V, A, \omega)$ consists of removing v in V, replacing (u, v) and (v, w) by (u, w) in A, weighted by $\omega((u, w)) = \omega((u, v)) + \omega((v, w))$.

Two weighted graphs $G = (V_G, A_G, \omega_G)$ and $G' = (V_{G'}, A_{G'}, \omega_{G'})$ are *homeomorphic* [16] if there exists an isomorphism between a graph obtained by mergings on G and a graph obtained by mergings on G'.

See Fig.2 for some examples of isomorphism and homeomorphism.

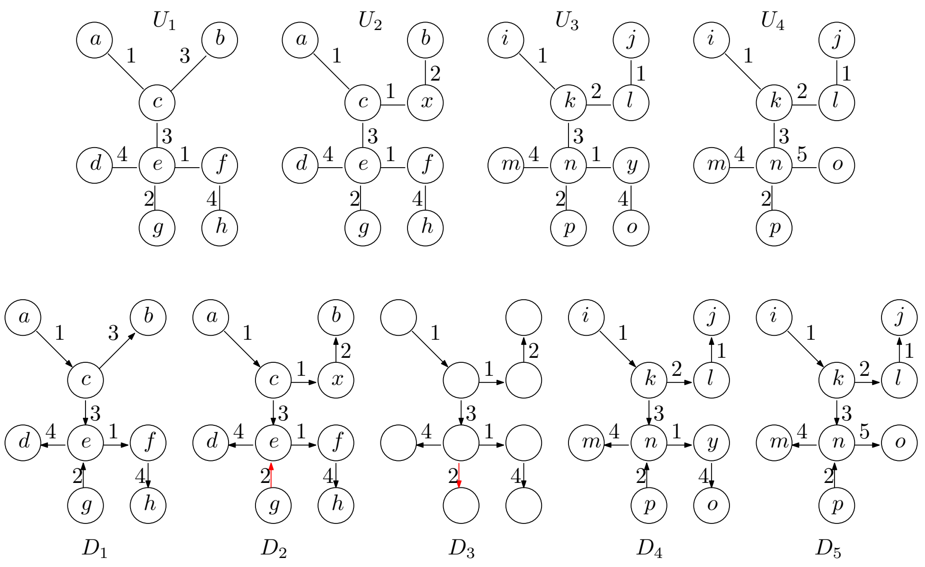

Figure 2: First row: U_1 is obtained from U_2 by merging on x. U_4 is obtained from U_3 by merging on y. U_2 and U_3 are isomorphic, U_1 and U_4 are homeomorphic. Second row: D_1 is obtained from D_2 by merging on x. D_5 is obtained from D_4 by merging on y. D_2 and D_4 are isomorphic, D_1 and D_5 are homeomorphic, D_2 and D_3 are not isomorphic (red arcs have different orientations).

3. Edit-based distances

The problem of comparing graphs (in particular trees) occurs in diverse areas such as computational biology, image analysis and structured databases. However, the graphs considered in these domains are most often with *labeled vertices*. Here, each notion will be introduced in the case of graphs with weighted edges/arcs.

In this section, we review different edit-based distances proposed in the literature, and select the one that is the best adapted our aims.

3.1. Edit operations

A lot of methods have been developed for tree comparison, as the maximal common subtree [17, 18], or the approximate subtree homeomorphism [19]. An approach widely used to compare trees is to search for a sequence of simple primitive operations (called *edit operations*) that transforms a tree into the other and that has a minimal cost.

For a graph $G = (V, A, \omega)$, classical edit operations are:

Resize: Change the weight of an arc $a = (u, v) \in A$.

Delete: Delete an arc $a = (u, v) \in A$ and merge u and v into one vertex.

Insert: Split a vertex in two vertices, and link them by a new arc.

See Fig.3 for illustrations.

The cost of these edit operations is given by a cost function $\gamma(w, w')$, where w (respectively w') is the total weight of the arcs involved in the operation before (respectively, after) its application. As a consequence, the cost of a resizement can be denoted by $\gamma(w, w')$, where w is the former weight of the resized arc and w' is the new weight, the cost of a deletion can be denoted by $\gamma(w, 0)$, where w is the weight of the deleted arc, and the cost of an insertion, $\gamma(0, w)$, where w is the weight of the created arc. Furthermore, we assume that γ is a metric. Typically, $\gamma(w, w') = |w - w'|$ or $(w - w')^2$.

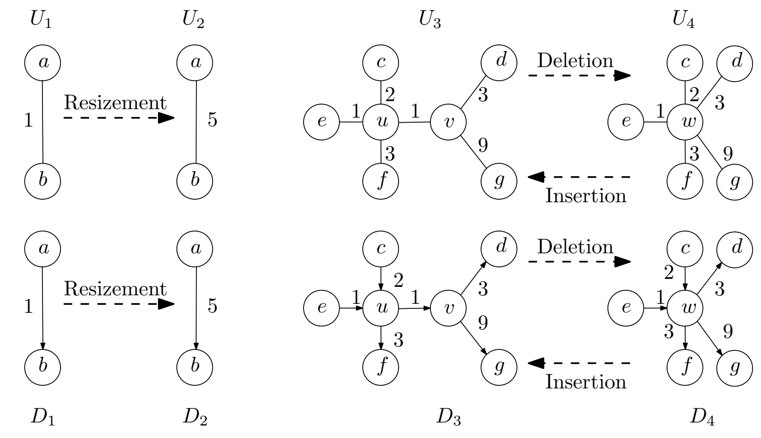

Figure 3: Examples of edit operations: on the top, on undirected graphs. On the bottom, on directed graphs.

3.2. *Choice of edit-based distance*

Various edit-based distances have been defined, using different constraints on the edit operations order or on the resulting correspondence between the vertices sets, and different definitions of operations. These edit-based distances can be classified, as proposed by Wang et al. [20]: edit distance [21], alignment distance [22, 23], isolated-subtrees distance [24, 25, 26], and top-down distance [27, 28, 29].

3.2.1. Edit distance

Let G_2 be the graph that results from the application of an edit operation s to graph G_1; this is written $G_1 \Rightarrow G_2$ via s. Let $\mathcal{S}$ be a sequence $s_1, s_2, ..., s_k$ of edit operations. We say that $\mathcal{S}$ transforms graph G to graph G' if there is a sequence of graphs $G_0, G_1, ..., G_k$ such that $G = G_0$, $G' = G_k$ and $G_{i-1} \Rightarrow G_i$ via s_i for $1 \leq i \leq k$. The cost of the sequence $\mathcal{S}$, denoted by $\gamma(\mathcal{S})$, is the sum of costs of the constituent edit operations. The *distance* from G to G', denoted by $\delta(G, G')$, is the minimum cost of all sequences of edit operations taking G to G'.

For our purpose, this kind of edit-based distance cannot be used, because the associated matching does not preserve topological relations between trees. In addition, the algorithm based on this distance is the one with the highest time complexity.

3.2.2. Isolated-subtrees distance

Isolated-subtrees distance is an edit-based distance such that two disjoint subtrees in the pattern tree will always be matched with two disjoint subtrees in the data tree.

3.2.3. Top-down distance

Top-down distance is an edit-based distance such that an arc (p, p') in the pattern tree can match an arc (d, d') in the data tree, only if $(par(p), p)$ matches $(par(d), d)$.

Isolated-subtrees distance and top-down distance cannot always match all the model tree, but only subparts, most often unconnected. However, we will see in the next subsection that it is not the case for alignment distance.

3.3. Alignment distance

In [22], Jiang et al. propose a similarity measure between vertex-labeled trees, that we transpose here for edge-weighted graphs.

Let $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, A_2, \omega_2)$ be two weighted graphs. Let $G'_1 = (V'_1, A'_1, \omega'_1)$ and $G'_2 = (V'_2, A'_2, \omega'_2)$ be weighted graphs obtained by inserting arcs weighted by 0 (zero) in G_1 and G_2, such that there exists an isomorphism $\mathcal{I}$ between (V'_1, A'_1) and (V'_2, A'_2). The set of all couples of arcs $\mathcal{A} = \{(a_1, a_2); a_1 \in A'_1, a_2 \in A'_2, a_2 = \mathcal{I}(a_1)\}$ is called an alignment of G_1 and G_2.

The *cost* $C_\mathcal{A}$ of $\mathcal{A}$ is the sum of the costs of all operations used to align G_1 and G_2: the insertions of arcs weighted by 0 (zero) (which is free, due to non-variation of weights), and the resizement for each arc $a_1 \in A'_1$ to the weight of $\mathcal{I}(a_1)$. More formally :

$$C_\mathcal{A} = \sum_{(a_1, a_2) \in \mathcal{A}} \gamma(\omega'_1(a_1), \omega'_2(a_2)) \ .$$

The minimal cost of all alignments from G_1 and G_2, called the *alignment distance*, is denoted by $\alpha(G_1, G_2)$. The alignment distance is a special case of the edit distance, where any insertion occurs before any deletion. As a result, $\alpha(G, G') \geq \delta(G, G')$.

Alignment distance is interesting in our case for three reasons: it takes into account topological relations between trees, it can be computed in polynomial time, and it enables to "remove edges", regardless of the rest of the graph, solving the problem of splitted vertices.

3.3.1. Illustration

We consider only the problem of splitted vertices in this example. In Fig.4, we show the way to obtain an alignment with minimal cost: only one edge insertion in the pattern tree, on the vertex T, representing the torso.

The resulting alignment, represented by dotted lines, has a cost of 12. The corresponding vertices matching is the following: H matches with a, T with $\{b, e\}$, A_1 with d, A_2 with c, C with f, F_1 with n, and F_2 with o.

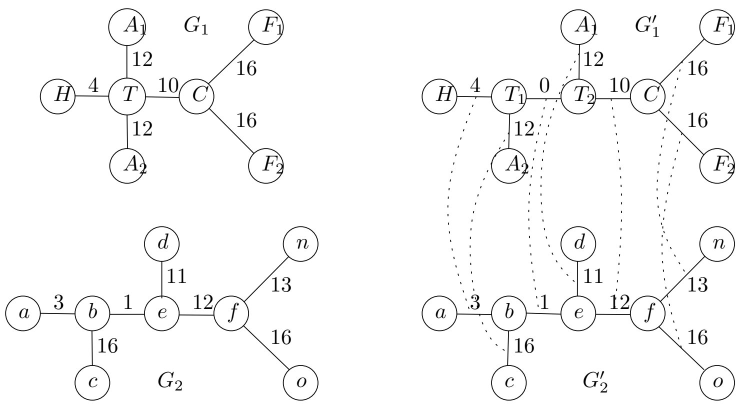

Figure 4: Example of alignment: G'_1 (resp. G'_2) is obtained from G_1 (resp. G_2) by insertions of edges (here $G'_2 = G_2$). The dotted lines represent one of the possible alignments of G'_1 and G'_2.

4. Homeomorphic alignment distance

If we now take into account the problem of useless 2-degree vertices, we can see that the alignment strategy does not work anymore (see Fig.5). In this example, the split of the "leg" edges of the data tree in several parts involves that the best alignment is obtained by matching the "arms" of the pattern (edges $\{T, A_1\}$ and $\{T, A_2\}$) with the parts of the legs in the data tree which are closest of them, in regard of their weights (edges $\{m, o\}$ and $\{j, n\}$). The cost of the alignment, shown by the dotted lines, is equal to 26 and is minimal.

For the purpose of solving the useless vertex problem, we propose a new alignment strategy.

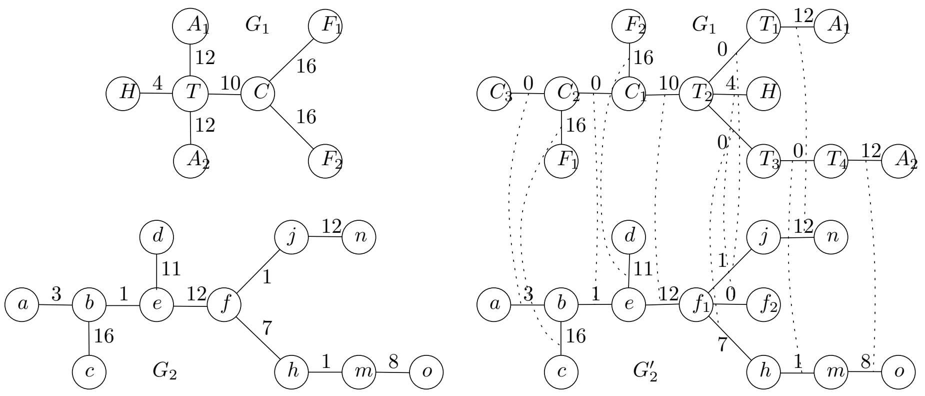

Figure 5: Example of bad alignment, due to useless 2-degree vertices.

4.1. Merging kernel

Considering that a merging on a vertex v on the graph $G = (V, A, \omega)$ does not affect the degree of any vertex in $V \setminus \{v\}$ (by definition of merging operation) and therefore the possibility of merging this vertex, the number of possible mergings decreases by one after each merging. In consequence, the maximal size of a sequence of merging operations, transforming G into another graph $G' = (V', A', \omega')$ is equal to the initial number of possible mergings in G. It can be remarked that any sequence of merging operations of maximal size yields the same result. The graph resulting of such a sequence on G is called the *merging kernel* of G, and is denoted by $\mathcal{MK}(G)$.

The following proposition is straightforward:

Proposition 1. *Two graphs $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, G_2, \omega_2)$ are homeomorphic iff $\mathcal{MK}(G_1)$ and $\mathcal{MK}(G_2)$ are isomorphic.*

4.2. Homeomorphic alignment distance

Let $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, A_2, \omega_2)$ be two weighted graphs. Let $G'_1 = (V'_1, A'_1, \omega'_1)$ and $G'_2 = (V'_2, A'_2, \omega'_2)$ be weighted graphs obtained by deleting arcs in G_1 and G_2, such that there exists an homeomorphism between G'_1 and G'_2 (not necessarily unique). Let $G''_1 = (V''_1, A''_1, \omega''_1)$ and $G''_2 = (V''_2, A''_2, \omega''_2)$ be the merging kernel of G'_1 and G'_2, respectively. From proposition 1, there exists an isomorphism $\mathcal{I}$ between G''_1 and G''_2. The set of all couples of arcs $\mathcal{H} = \{(a, a'); a \in A''_1, a' \in A''_2, a' = \mathcal{I}(a)\}$ is called an *homeomorphic alignment* of G_1 with G_2 (see figure 6). The graph G''_1 is called the *left graph* of $\mathcal{H}$. The graph G''_2 is called the *right graph* of $\mathcal{H}$.

The *cost* $C_{\mathcal{H}}$ of $\mathcal{H}$ is the sum of the costs of all operations used to homeomorphically align G_1 and G_2: the deletion of arcs in G_1 and G_2, to obtain G'_1 and G'_2 respectively, and the resizement for each arc $a_1 \in A''_1$ to the weight of $\mathcal{H}(a_1)$. More formally :

$$C_{\mathcal{H}} = \sum_{(a,a')\in\mathcal{H}} \gamma(\omega_1''(a), \omega_2''(a')) + \sum_{a_d\in A_1\setminus A_1'} \gamma(\omega_1(a_d), 0) + \sum_{a_d'\in A_2\setminus A_2'} \gamma(0, \omega_2(a_d')) \ .$$

This minimal cost of all homeomorphic alignments between G_1 and G_2, called the *homeomorphic alignment distance*, is denoted by $\eta(G_1, G_2)$.

4.2.1. Illustration

Let us consider again the pattern and data trees used in Fig. 5. In Fig.6 is shown the way to obtain a homeomorphic alignment with minimal cost: only one edge deletion in the data tree (edge $\{b, e\}$), is necessary to obtain a homeomorphism. The merging kernel of G_1' is equal to G_1', and the merging kernel of G_2' is obtained by merging on vertices j, h and m.

The resulting homeomorphic alignment, represented by dotted lines, has a cost of 12. The corresponding vertices matching is the following: H with a, T with $\{b, e\}$, A_1 with d, A_2 with c, C with f, F_1 with n, and F_2 with o.

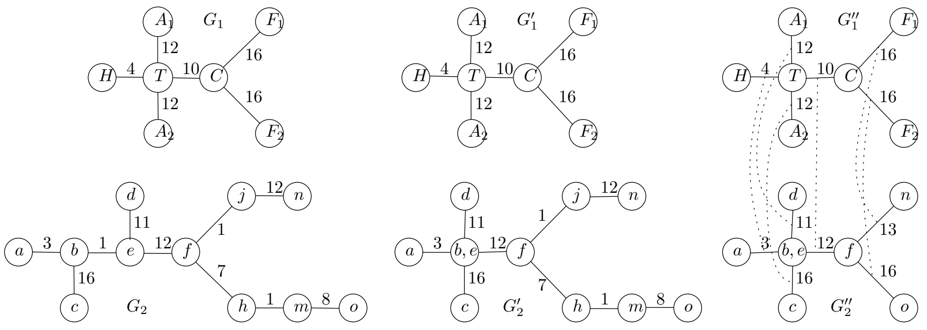

Figure 6: Example of homeomorphic alignment: G_1' (resp. G_2') is obtained from G_1 (resp. G_2) by deletions of edges. $G_1'' = \mathcal{MK}(G_1')$ and $G_2'' = \mathcal{MK}(G_2')$. The dotted lines represent one of the possible homeomorphic alignments of G_1 and G_2.

4.3. Algorithm for rooted trees

In order to compute homeomorphic alignment distance with good complexity, we will use a bottom-up approach. This approach, widely used in the literature on edit-based distances, is based on a reformulation of the distance between trees in terms of both the distance between their subtrees and the matching of their roots. Then, using this reformulation, the computation starts by the distances between the simplest subtrees (i.e the leafs), and goes up to the root. The distances between subtrees are kept in memory, avoiding redundancy of computation.

However, there are two main differences to take into account in the case of homeomorphic alignment distance: the fact that information is on arcs, rather than of vertices, and the fact that an arc can be matched with a set of arcs merged together (due to merging kernel application).

The first problem involves to take into consideration several arc weights in the reformulation (one for each arc between the root and its children), instead of the unique value linked to the root, as in the literature.

The second problem involves to take into account not only the subtrees of the tree, but also those which can be generated by some merging.

Our approach to solve these two problems is to use a special kind of tree, the root of which has only one child. Then, we make a reformulation of the homeomorphic alignment distance between this kind of trees, in function of subtrees with the same property. By this way, there is only one new arc to take into consideration at each step, the one between the root and its child. Moreover, we consider that this arc can be the result of a merging. In combination with the recursive nature of the bottom-up approach, it will lead to consider subtrees with all possible mergings. Finally, we also need a reformulation of the homeomorphic alignment distance between "classic" trees in function of the one between the subtrees described above.

4.3.1. Definitions and notations

First, we need to define more formally the particular kind of tree described above.

Let $T = (V, A, \omega)$ be a weighted tree rooted in r_T.

We denote by $T(v), v \in V$, the subtree of T rooted in v (see Fig.7).

Let v_a be an ancestor of v, we denote by $T_{cut}(v, v_a)$ the subgraph of T obtained from $T(v_a)$ by removing all complete subtrees which do not contain vertices of $T(v)$.

We denote by $T(v, v_a)$ the tree obtained from $T_{cut}(v, v_a)$ by merging on each vertex $n \in anc(v) \setminus \{v_a, v\}$. We say that $T(v, v_a)$ is the subtree of T rooted in v_a *pruned in* v, and we call this kind of trees a *pruned tree* (see Fig.7 for an example).

We denote by $\mathcal{F}(T, v)$ the *pruned forest*, the connected components of which are the trees $T(p, v)$, for all $p \in \mathcal{C}(v)$ (see Fig.7 for an example). By abuse of notation we also denote by $\mathcal{F}(T, v)$ the set of all connected components of this forest.

4.3.2. Reformulations

In order to use a bottom-up approach, we have to express the homeomorphic alignment distance:

- between trees in function of the distances between their pruned forests (Prop. 2).
- between pruned trees in function of the distances between their pruned subtrees (Prop. 4).
- between pruned forests in function of the distances between their pruned subtrees (Prop. 5).

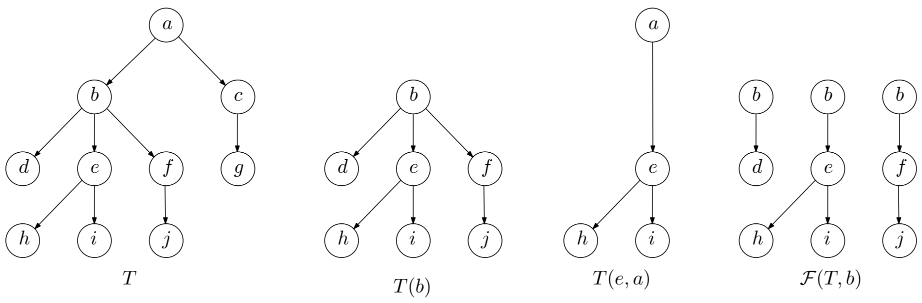

Figure 7: From the left to the right: a tree T, a subtree of T rooted in b, a subtree of T rooted in a and pruned in e, and a pruned forest of T with origin b.

In addition, we have to reformulate these distances in the special cases where at least one of the trees is empty (Prop. 3).

In the sequel we consider two weighted trees $P = (V_P, A_P, \omega_P)$ and $D = (V_D, A_D, \omega_D)$, rooted respectively in r_P and r_D.

Proposition 2. *We have :*

$$\eta(P, D) = \eta(\mathcal{F}(P, r_P), \mathcal{F}(D, r_D)).$$

Proof. Since the root of a tree cannot be eliminated by any operation (merging, deletion) involved in the definition of η, it may be seen that any homeomorphic alignment $\mathcal{H}$ of P with D has a corresponding alignment $\mathcal{H}'$ of $\mathcal{F}(P, r_P)$ with $\mathcal{F}(D, r_D)$ of same cost, and the converse also holds. □

Proposition 3. *Let $i \in V_P \setminus \{r_P\}, j \in V_D \setminus \{r_D\}, i_a \in anc(i), j_a \in anc(j)$,*

$$\begin{array}{l}
\eta(\emptyset, \emptyset) = 0 \\
\eta(P(i, i_a), \emptyset) = \eta(\mathcal{F}(P, i), \emptyset) + \gamma(\omega_P(i_a, i), 0) \\
\eta(\mathcal{F}(P, i_a), \emptyset) = \sum_{i' \in \mathcal{C}(i_a)} \eta(P(i', i_a), \emptyset) \\
\eta(\emptyset, D(j, j_a)) = \eta(\emptyset, \mathcal{F}(D, j)) + \gamma(0, \omega_D(j_a, j)) \\
\eta(\emptyset, \mathcal{F}(D, j_a)) = \sum_{j' \in \mathcal{C}(j_a)} \eta(\emptyset, D(j', j_a)).
\end{array}$$

Proof. Straightforward. □

Proposition 4. *Let $i \in V_P \setminus \{p\}, j \in V_D \setminus \{d\}, i_a \in anc(i), j_a \in anc(j)$.*

$$\begin{array}{l}
\eta(P(i, i_a), D(j, j_a)) = \\
min \left\{ \begin{array}{l}
\eta(P(i, i_a), \emptyset) + \eta(\emptyset, D(j, j_a)), \\
\gamma(\omega_P(i_a, i), \omega_D(j_a, j)) + \eta(\mathcal{F}(P, i), \mathcal{F}(D, j)), \\
min_{j_c \in \mathcal{C}(j)}\{\eta(P(i, i_a), D(j_c, j_a)) + \eta(\emptyset, \mathcal{F}(D, j) \setminus D(j_c, j))\}, \\
min_{i_c \in \mathcal{C}(i)}\{\eta(P(i_c, i_a), D(j, j_a)) + \eta(\mathcal{F}(P, i) \setminus P(i_c, i), \emptyset)\}.
\end{array} \right.
\end{array}$$

Proof. Let $\mathcal{H}$ be an homeomorphic alignment of $P(i,i_a)$ with $D(j,j_a)$, and let $\mathcal{H_L} = (V_L, A_L, \omega_L)$ and $\mathcal{H_R} = (V_R, A_R, \omega_R)$ the left and right graphs of $\mathcal{H}$, respectively.

There are seven possible cases:

1. $\mathcal{H}$ is an empty set (it is cheaper to remove both $P(i,i_a)$ and $D(j,j_a)$ than to align them).
2. $\{(i_a,i),(j_a,j)\} \in \mathcal{H}$.
3. $\exists f \in A_R$, f being obtained by merging (j_a,j) with other arcs. In this case, there is one and only one child j_c of j, such (j,j_c) is merged with (j_a,j) in f, and then all $D(j'_c,j), j'_c \in \mathcal{C}(j) \setminus \{j_c\}$ are deleted.
4. $\exists f \in A_L$, f being obtained by merging (i_a,i) with other arcs. In this case, there is one and only one child i_c of i, such (i,i_c) is merged with (i_a,i) in f, and then all $P(i'_c,i), i'_c \in \mathcal{C}(i) \setminus \{i_c\}$ are deleted.

Cases 1,2,3,4 justify, respectively, the lines 1,2,3,4 of the expression of $\eta(P(i,i_a),D(j,j_a))$ in the proposition.

The three last cases cannot lead to a better homeomorphic alignment:

5. The deletion of (i_a,i) and (j_a,j) cannot be preferred to the resizement (possible case 2), because $\gamma(\omega_P(i_a,i),0) + \gamma(0,\omega_D(j_a,j)) \geq \gamma(\omega_P(i_a,i),\omega_D(j_a,j))$.
6. If (i_a,i) was deleted, and not (j_a,j), then only one $P(i_c,i), i_c \in \mathcal{C}(i)$ is aligned with $D(j,j_a)$, the other being removed. It is less expensive to merge (i_a,i) with (i,i_c) (possible case 3), because $\gamma(\omega_P(i_a,i),0) + \gamma(\omega_P(i,i_c),\omega_D(j_a,j)) \geq \gamma(\omega_P(i_a,i_c),\omega_D(j_a,j))$.
7. The deletion of (j_a,j) is more expensive than the merging of (j_a,j) (possible case 4), for the same reasons as above.

□

Proposition 5. $\forall A \subseteq \mathcal{F}(P,i), B \subseteq \mathcal{F}(D,j)$,

$$\eta(A,B) = \\ min \begin{cases} min_{D(j',j)\in B} & \{\eta(A, B \setminus \{D(j',j)\}) + \eta(\emptyset, D(j',j))\}, \\ min_{P(i',i)\in A} & \{\eta(A \setminus \{P(i',i)\}, B) + \eta(P(i',i),\emptyset)\}, \\ min_{P(i',i)\in A, D(j',j)\in B} & \{\eta(A \setminus \{P(i',i)\}, B \setminus \{D(j',j)\}) \\ & +\eta(P(i',i), D(j',j))\}, \\ min_{P(i',i)\in A, B'\subseteq B} & \{\eta(A \setminus \{P(i',i)\}, B \setminus B'), \\ & +\eta(\mathcal{F}(P,i'),B') + \gamma(\omega_P(i,i'),0)\}, \\ min_{A'\subseteq A, D(j',j)\in B} & \{\eta(A \setminus A', B \setminus \{D(j',j)\})+ \\ & \eta(A', \mathcal{F}(D,j')j) + \gamma(0,\omega_D(j,j'))\}. \end{cases}$$

Proof. Let $\mathcal{H}$ be an homeomorphic alignment of $A \subseteq \mathcal{F}(P,i)$ with $B \subseteq \mathcal{F}(D,j)$, and let $P(i',i) \in A$ and $D(j',j) \in B$. There are five possible cases:

1. $D(j',j)$ is not aligned with element of A,

2. $P(i', i)$ is not aligned with element of B,
3. $P(i', i)$ is aligned with $D(j', j)$,
4. disjoint subparts of $P(i', i)$ are aligned with elements of B,
5. disjoint subparts of $D(j', j)$ are aligned with elements of A.

Cases 1, 2, 3, 4, 5 justify, respectively, the lines 1, 2, 3, 4, 5 of the expression of $\eta(A, B)$ in the proposition. □

4.3.3. Algorithm

Using the reformulations defined above, we can now design our algorithm following the bottom-up approach.

Algorithm 1: Homeomorphic Alignment Distance for Rooted Trees

```
Data: pattern rooted tree $P$, data rooted tree $D$
Result: $\eta(P, D) = \eta(\mathcal{F}(P, r_P), \mathcal{F}(D, r_D))$; // Prop.2
begin
    foreach $p \in V_P$, in suffix order do
        foreach $A \subseteq \mathcal{F}(P, p)$ do
            Compute $\eta(A, \emptyset)$; // Prop.3
        foreach $p_a \in anc(p) \setminus \{p\}$ do
            Compute $\eta(P(p, p_a), \emptyset)$; // Prop.3
    foreach $d \in V_D$, in suffix order do
        foreach $B \subseteq \mathcal{F}(D, d)$ do
            Compute $\eta(\emptyset, B)$; // Prop.3
        foreach $d_a \in anc(d) \setminus \{d\}$ do
            Compute $\eta(\emptyset, D(d, d_a))$; // Prop.3
    foreach $p \in V_P$, in suffix order do
        foreach $d \in V_D$, in suffix order do
            foreach $A \subseteq \mathcal{F}(P, p)$ do
                foreach $B \subseteq \mathcal{F}(D, d)$ do
                    Compute $\eta(A, B)$; // Prop.5
            foreach $p_a \in anc(p) \setminus \{p\}$ do
                foreach $d_a \in anc(d) \setminus \{d\}$ do
                    Compute $\eta(P(p, p_a), D(d, d_a))$; // Prop.4
end
```

4.3.4. Complexity

We denote by N the maximum degree of a vertex in the pattern tree or in the data tree, H the maximal height of both trees and S the maximal size (number of vertices) of both trees. We recall that a set of size n has 2^n subsets.

The time complexity of computing each reformulation used in the algorithm is:

- $\eta(P(i, i_a), \emptyset)$ and $\eta(\emptyset, D(j, j_a)) \rightarrow O(1)$.
- $\eta(A, \emptyset)$ and $\eta(\emptyset, B) \rightarrow O(N)$.
- $\eta(P(i, i_a), D(j, j_a)) \rightarrow O(N)$.
- $\eta(A, B) \rightarrow O(N * 2^N)$.

Combining these complexities with the different loops of the algorithm, we obtain that the total computation is in $O(S^2 * (N * 2^{3*N} + H^2 * N))$ time complexity.

If the maximal degree is bounded, the total computation is in $O(S^2 * H^2)$ time complexity.

4.4. Algorithm for unrooted trees

First, let us give an expression of the homeomorphic alignment distance between unrooted trees, in function of the distances between all their possible rooted versions.

Let $G = (V, E, \omega)$ be a weighted tree, let $r \in V$, we denote by G^r, the directed weighted tree rooted in r, such that G is the undirected graph associated to G^r (see Fig.8).

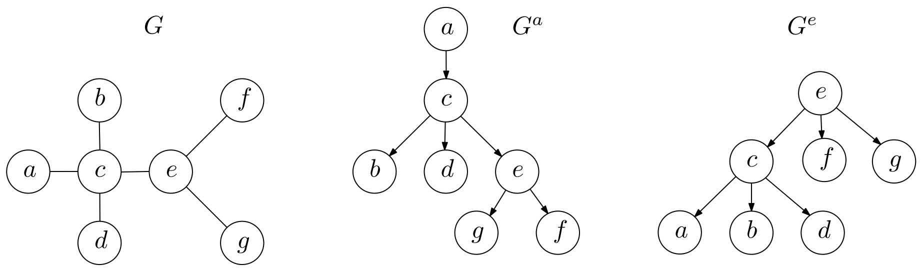

Figure 8: A tree G and the rooted trees G^a and G^e.

Proposition 6. *Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. We have:*

$$\eta(P, D) = min_{i \in V_P, j \in V_D}\{\eta(P^i, D^j)\}.$$

Proof. Let $G = (V, E, \omega)$ be a graph, and $r \in V$ a vertex of G. Notice that:

- a merging occurring on a vertex $v \in V \setminus \{r\}$ in G can occur in G^r,
- deletion, insertion, resizement, and division occurring in G can occur in G^r.

On the other hand, for each optimal homeomorphic alignment $\mathcal{H}$ of P in D, it is easy to see that there exists $p \in V_P$ and $d \in V_D$, such that p and d are not affected by a merging. For example, if $D' = (V'_D, E'_D, \omega'_D)$ is a subgraph such as $\eta(P, D) = \eta(P, D')$, p and d can be chosen as 1-degree vertices of V_P and

V'_D, respectively. As a result, $\eta(P,D) = \eta(P^p, D^d)$. Since the homeomorphic alignment is more constrained in the case of rooted trees than in the case of unrooted trees, we can assure than $\eta(P,D) \leq \eta(P^a, D^b), a \in V_P, b \in V_D$. To sum up, knowing that $\eta(P,D) \leq \eta(P^a, D^b), a \in V_P, b \in V_D$, and that there exists $p \in V_P$ and $d \in V_D$, such that $\eta(P,D) = \eta(P^p, D^d)$, we conclude that $\eta(P,D) = min_{i\in V_P, j \in V_D} \eta(P^i, D^j)$. □

4.4.1. Naive algorithm

Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. From proposition 5, we propose a first, naive algorithm to compute $\eta(P,D)$, consisting of computing the homeomorphic alignment distance for all couples of weighted rooted trees we can obtain from P and D, and keeping the minimum reached.

Complexity. Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. As the number of rooted trees we can obtain from an undirected tree is equal to the number of vertices of this graph, the number of couples of weighted rooted trees we can obtain from P and D is equal to $|V_P| * |V_D|$.

The total computation is then in $O(|V_P|^2 * |V_D|^2 * (2^{d_P} * 2^{d_D} * (d_D * 2^{d_P} + d_P * 2^{d_D}) + h_P * h_D * (d_P + d_D)))$ time complexity, where h_P (respectively, h_D) is the maximal height of a rooted tree obtained from P (respectively, D).

If the maximal degree is bounded, the total computation is in $O(|V_P|^2 * |V_D|^2 * h_P * h_D)$ time complexity.

4.4.2. Optimized algorithm

It is easy to see that the above algorithm computes the alignment of subparts of P and D more than one time. Using dynamic programming and an adapted order of navigation in the tree, we can avoid useless computation.

Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two undirected weighted trees.

We denote by $\mathcal{F}(P, a, b), a, b \in V_P$ the set of rooted trees P^r, $r \in V_P$, such that b is an ancestor of a in P^r. We denote by $anc(P, a, b)$, $a, b \in V_P$ the set of vertices $x \in V_P$ such that x is an ancestor of a in at least one rooted tree in $\mathcal{F}(P, a, b)$. We denote by $\mathcal{C}(P, a, b)$, $a, b \in V_P$ the set of of vertices $x \in V_P$ such that x is a child of a in at least one rooted tree in $\mathcal{F}(P, a, b)$.

It is easy to see that for computing $\eta(P^p(i, i_a), D^d(j, j_a))$ and $\eta(\mathcal{F}(P^p, i), \mathcal{F}(D^d, j))$, we need to know $\eta(P^p(i_c, i), D^d(j_c, j))$ for all $i_c \in \mathcal{C}(P, i, p)$, $j_c \in \mathcal{C}(D, j, d)$. We can start by computing $\eta(P^p(i, i_a), D^d(j, j_a))$ and $\eta(\mathcal{F}(P^p, i), \mathcal{F}(D^d, j))$, for all i (respectively, j) being a leaf of P^p (respectively, D^d), which have no child, by definition, and continue iteratively with all vertices which have all their children already computed.

An adapted order of navigation for a tree $T = (V, E, \omega)$ can be obtained by the following algorithm.

Algorithm 2: Order of navigation computation algorithm (computeOrder)

```
Data: unrooted tree $T$
Result: list of couples of vertices $L$
begin
    list of couples of vertices $L \leftarrow \emptyset$
    LIFO queue of vertices $Q \leftarrow \emptyset$
    foreach $v \in V$ do
        if $deg(v) = 1$ then
            push $v$ in $Q$
    while $Q \neq \emptyset$ do
        pop $v$ of $Q$
        foreach $w \in \mathcal{N}(v)$ do
            if $(v, w) \notin L$ then
                if $\forall x \in \mathcal{N}(v) \setminus \{w\}, (x, v) \in L$ then
                    push $w$ in $Q$
                    add $(v, w)$ at the end of $L$
end
```

Complexity. Let $T = (V, E, \omega)$ be an unrooted tree. Each vertex of degree 1 (one) will be put in the queue, then, for each edge, each of its vertices will be put twice in the queue. As $|V| = |E| + 1$, the complexity of computeOrder is in $O(|E|)$.

Final algorithm. We can compute Homeomorphic Alignment for unrooted trees with improved complexity, using this order of navigation.

Algorithm 3: Homeomorphic Alignment Distance for Unrooted Trees

```
Data: pattern rooted tree $P$,
data rooted tree $D$
Result: $\eta(P, D)$
begin
    list of couples of vertices $L_P \leftarrow computeOrder(P)$
    list of couples of vertices $L_D \leftarrow computeOrder(D)$
    foreach $(p, p') \in L_P$ do
        foreach $A \subseteq \mathcal{F}(P^{p'}, p)$ do
            Compute $\eta(A, \emptyset)$; // Prop.3
        foreach $p_a \in anc(P, p, p') \setminus \{p\}$ do
            Compute $\eta(P^{p_a}(p, p_a), \emptyset)$; // Prop.3
    foreach $(d, d') \in L_D$ do
        foreach $B \subseteq \mathcal{F}(D^{d'}, d)$ do
            Compute $\eta(\emptyset, B)$; // Prop.  2
        foreach $d_a \in anc(D, d, d') \setminus \{d\}$ do
            Compute $\eta(\emptyset, D^{d_a}(d, d_a))$; // Prop.3
    foreach $(p, p') \in L_P$ do
        foreach $(d, d') \in L_D$ do
            foreach $A \subseteq \mathcal{F}(P^{p'}, p)$ do
                foreach $B \subseteq \mathcal{F}(D^{d'}, d)$ do
                    Compute $\eta(A, B)$; // Prop.5
            foreach $p_a \in anc(P, p, p') \setminus \{p\}$ do
                foreach $d_a \in anc(D, d, d') \setminus \{d\}$ do
                    Compute $\eta(P^{p_a}(p, p_a), D^{d_a}(d, d_a))$; // Prop.4
    Compute $\eta(P, D)$; // Prop.2 and Prop.6
end
```

Complexity. Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. The initialization of L_P and L_D is in $O(|V_P| * d_P + |V_D| * d_T)$ time complexity.

Observe that, for $p \in V_P$,

$$\sum_{p' \in \mathcal{N}(p)} |anc(P, p, p') \setminus \{p\}| = |V_P| - 1 \ .$$

As a result,

$$\sum_{(p,p') \in L_P} |anc(P, p, p') \setminus \{p\}| = (|V_P| - 1) \times |V_P| \ .$$

Combining the time complexities of computing the reformulations (given in the Sec.4.3.4) with the loops of the algorithm, and taking into account the above observation, we obtain that the total computation time of this algorithm is in $O(S^2 * (N * 2^{3*N} + S^2 * N))$ complexity, with N and S defined as in Sec.4.3.4.

If the maximal degree is bounded, the total computation is in $O(S^4)$ time complexity.

5. Cut operation

For the purpose of removing spurious branches without any cost, we propose to integrate the cut operation in our alignment.

In [30], Wang et al. propose a new operation allowing to consider only a part of a tree. Let $G = (V, A, \omega)$ be a weighted tree. *Cutting* G at an arc $a \in A$, means removing a, thus dividing G into two subtrees G_1 and G_2. The *cut operation* consists of cutting G at an arc $a \in A$, then considering only one of the two subtrees. Let K a subset of A. We use $Cut(G, K, v)$ to denote the subtree of G containing v and resulting from cutting G at all arcs in K. In the case of a rooted tree, we consider that the root r_G of G cannot be removed by the cut operation, and then we use the notation $Cut(G, K) = Cut(G, K, r_G)$. In the case of a rooted forest, we consider that the root of each rooted tree composing the rooted forest cannot be removed by the cut operation, and then we use the same notation than above: $Cut(G, K)$.

See Fig.9 for some examples of cut operation.

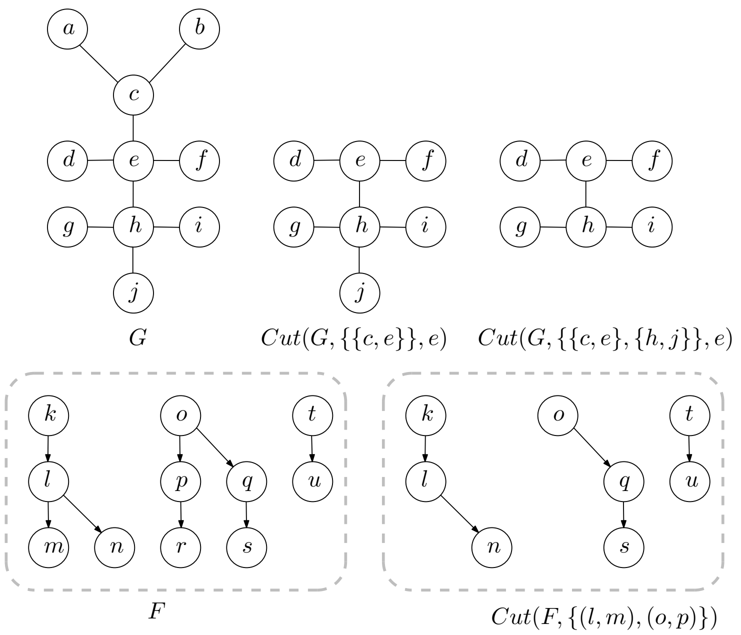
G
$Cut(G, \{\{c, e\}\}, e)$
$Cut(G, \{\{c, e\}, \{h, j\}\}, e)$
F
$Cut(F, \{(l, m), (o, p)\})$

Figure 9: Examples of cut operations. Top row: on an unrooted tree. Bottom row: on a rooted forest.

Our main problem can be stated as follows: Given a weighted tree $P = (V_P, A_P, \omega_P)$ (the pattern tree) and a weighted tree $G_D = (V_D, A_D, \omega_D)$ (the data tree), find $\eta_{cut}(P, D) = min_{K \subseteq A_D, v \in V_D}\{\eta(P, Cut(D, K, v)\}$ and the associated homeomorphic alignment. In the case of rooted trees and rooted forests, $\eta_{cut}(P, D) = min_{K \subseteq A_D}\{\eta(P, Cut(D, K))\}$.

5.1. Integration of cut operation in our algorithm

It can be seen that the cut operation can be integrated in the homeomorphic alignment by replacing the deletion of complete subtrees by their cut.

In order to obtain the algorithms for the computation of η_{cut}, we just need to replace:

- η by η_{cut} in the reformulations and algorithms of the Subsec. 4.3 and 4.4
- Prop. 3 by the following:

  Proposition 7. *Let $i \in V_P \setminus \{p\}, j \in V_D \setminus \{d\}, i_a \in anc(i), j_a \in anc(j)$,*

$$\begin{aligned}
&\eta_{cut}(\emptyset, \emptyset) = 0 \\
&\eta_{cut}(P(i, i_a), \emptyset) = \eta_{cut}(\mathcal{F}(P, i), \emptyset) + \gamma(\omega_P(i_a, i), 0) \\
&\eta_{cut}(\mathcal{F}(P, i_a), \emptyset) = \sum_{i' \in \mathcal{C}(i_a)} \eta_{cut}(P(i', i_a), \emptyset) \\
&\eta_{cut}(\emptyset, D(j, j_a)) = 0 \\
&\eta_{cut}(\emptyset, \mathcal{F}(D, j_a)) = 0.
\end{aligned}$$

It is interesting to remark that the complexity is not modified.

6. Experimentation

6.1. Usage of homeomorphic alignment

There are several ways to use the homeomorphic alignment:

- If we have no a priori knowledge both on pattern tree P and on data tree D, we need to use the homeomorphic alignment on the two unrooted trees. In this case, the complexity is in $O(|V_P|^2 * |V_D|^2)$.
- If we want to be sure that a specific vertex v in the data tree is aligned with a specific vertex w in the data tree (for example if we are sure than the vertex of the head in the data is the one with the highest z-coordinate), we can use the homeomorphic alignment between P^v and D^w. In this case, the complexity is in $O(|V_P| * |V_D| * h_P * h_D)$, where h_P (respectively, h_D) represents the height of P (respectively, D).
- If we want to be sure that a specific vertex v in the model tree is aligned (i.e. there is no merging on v), we can use the homeomorphic alignment between P^v and D, by successively computing the homeomorphic alignment between P^v and D^w, for all $w \in V_D$, and using optimizations as in 4.4.2. In this case, the complexity is in $O(|V_P| * h_P * |V_D|^2)$.

- If we want to be sure that a specific vertex v in the data tree is aligned (i.e. there is no merging or cut on v), we can use the homeomorphic alignment between P and D^v, using the same method as above. In this case, the complexity is in $O(|V_P|^2 * |V_D| * h_D)$.

In our application, consisting to find the initial pose of a subject, we can at least assume that the torso of the subject will be aligned. Then, we can use the matching with $O(|V_P| * h_P * |V_D|^2)$ for the initialization. For the tracking, if a part of the subject is static, we can use the last alignment of this part for obtain a matching in $O(|V_P| * |V_D| * h_P * h_D)$.

6.2. Results

Our model tree contains seven vertices, representing head, torso, crotch, the two hands and the two feet. Experimentally, the data tree obtained from the skeleton of the visual hull has a degree bounded by 4, and its number of vertices is between seven and twenty, with a Gaussian probability repartition centered on ten.

All the results have been obtained on a computer with a processor Intel(R) Core(TM) 2 Quad Q8200 at 2.33 GHz and 3 Go of RAM, with Linux, Ubuntu 9.04. The algorithms have been implemented in C++.

6.2.1. Speed

Protocol. To find the average computation time of involved algorithms, we have randomly generated 32 pattern trees, and for each one, 32 data trees, yielding 1024 pairs of trees. Each pattern tree has seven vertices, one of which has a degree equal to 4. Each data tree has at least one 4-degree vertex. The results for both alignment and homeomorphic alignment algorithms, for the different kinds of trees, are shown in figure 10.

Discussion. For our purpose of initializing the pose of a subject, we can see that in the average case ($|V_d| = 12$), the homeomorphic alignment can be computed very quickly (frequency ≥ 100), even in the case of unrooted trees. However, we prefer to use the homeomorphic alignment with a rooted pattern tree, because of its better speed, without loss of precision.

The main observation we can do, in regard of the results, is the weak difference of performances between alignment and homeomorphic alignment in practice, even if there is a significant difference of complexity between alignment and homeomorphic alignment. It is due to small size of the trees involved here, and to the fact that, even if the degree of the trees is bounded, and by this way, can be considered as a constant, the computation of the alignment (or homeomorphic alignment) of subforests take a large part of the computation time. This computation part being similar for both algorithms, the rest (where the complexity differs) is less significant in regard of the speed.

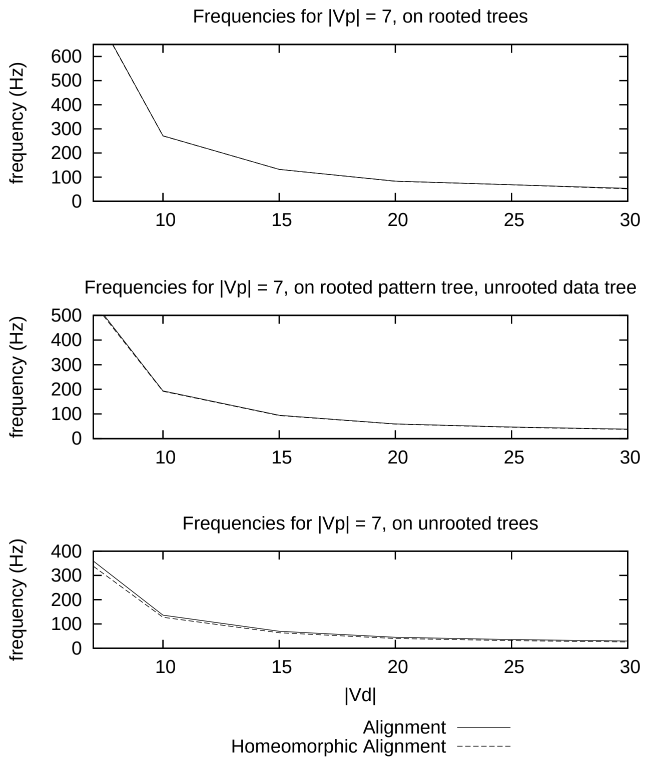

Figure 10: Frequencies of alignment and homeomorphic alignment for variable sizes of data tree, for the different rooting cases.

6.2.2. Accuracy

Protocol. Each experiment consists of the following:

1. We randomly generate a pattern tree P.
2. We generate a data tree D from P in two steps:

   weight variation: we randomly alter the edge weights by a given amount of variation.

   structural noise: we randomly add new vertices by three ways, corresponding to the different types of noise: splitting an existing vertex, and linking the two parts by a 0-weighted edge (*splitted vertices*), adding a new 2-degree vertex by the split of an edge (*useless 2-degree vertices*), and adding a new 1-degree vertex, linked by an randomly weighted edge (*spurious branches* and *ghost limbs*).
3. We compute the alignment and the homeomorphic alignment between P and D.
4. The precision is given by the percentage of good matchings, that is the percentage of pattern tree vertices that match with their equivalent in the

data tree (which is known, according to the above protocol to generate the data tree).

The results, obtained by averaging precisions over 1000 experiments, are shown on Fig. 11.

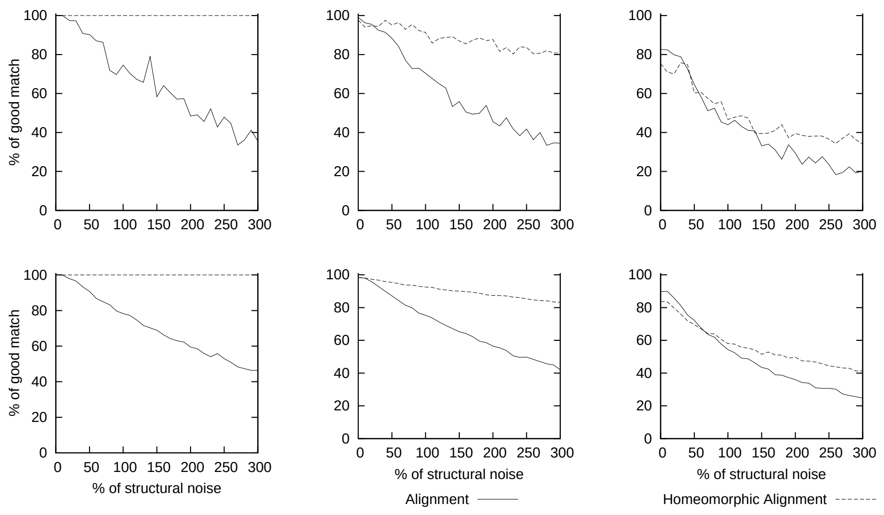

Figure 11: Precision with different percents of weight variation, for different sizes of pattern tree. First row: pattern tree with 10 vertices. Second row: pattern tree with 20 vertices. Columns, from left to right: 0%, 10% and 50% of weight variation.

Discussion. The first important point which can observed is that homeomorphic alignment give always perfect matchings when only structural noise occurs. In the other cases, the homeomorphic alignment globally gives a more accurate matching than the alignment, specially in regard of the structural noise. However, in case of high weight variation and low structural noise, the results of alignment are better than those of homeomorphic alignment. It is due to a special case of subtree and weight variation, which affects the accuracy of homeomorphic alignment but not the alignment (see Fig.12 for an example).

In the case of our application, we empirically observe weight variation between 0 and 33 percent, and structural noise between 0 and 200 percent. For the majority of these cases, homeomorphic alignment give sensibly better results.

6.2.3. Pose Initialization

We have checked the alignment on several types of visual hulls, with different resolutions (64^3 and 128^3), with or without spurious branches and ghosts limbs. Some of these results are shown in Fig. 13.

The ghost limb can be matched by the algorithm only if its position on the skeleton is the same as another limb, and if it has approximately the same length. In the other cases, it is successfully removed.

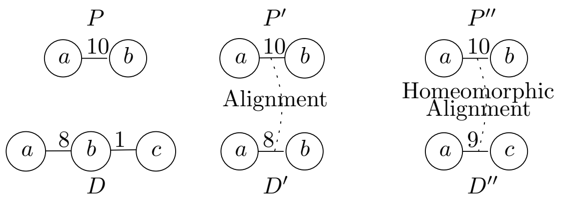

Figure 12: Special case where alignment is better than homeomorphic alignment. First column: P is the pattern tree, D is obtained from P by a weight variation of 20% on $\{a, b\}$, and by adding a new 1-degree vertex c, linked to b by an edge weighted by 1. Second column: optimal alignment. $P' = P$ and $D' = Cut(D, \{\{b, c\}\}, b)$. Accuracy is equal to 100%. Last column: homeomorphic alignment. P and D are already homeomorphic. $P'' = P$ and $D'' = \mathcal{MK}(D)$. Accuracy is equal to 50%.

The spurious branches do not disturb the good alignment of the model on the data.

The only case of bad alignment has been obtained on a very low quality visual hull, where the length of legs was shorter than the length of arms. This case can be solved by rooting the model tree on the head, and the data tree on the vertex with highest z-coordinate.

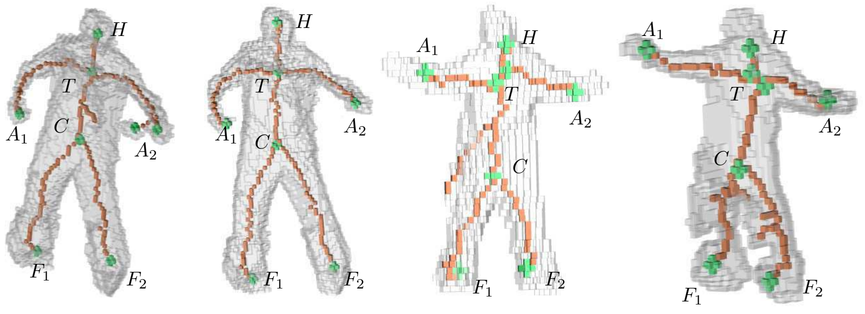

Figure 13: Examples of results obtained on 3D shapes with different resolutions and noises: green voxels represent the points matching with pattern tree.

7. Conclusion

In this paper, we have introduced a new type of alignment between weighted trees, the homeomorphic alignment, taking into account the topology and avoiding the noise induced by spurious branches, splitted and useless 2-degree vertices. We have also developed several robust algorithms to compute it with a good complexity, which enable its application in real time for motion capture purpose.

In future works, we will take into account more useful information on the model, such as spatial coordinates of data vertices, and include them in our algorithm, for a better robustness. Finally, using this alignment, we will propose a new fast method of pose initialization for motion capture applications.

References

[1] T. Moeslund, A. Hilton, V. Krüger, A survey of advances in vision-based human motion capture and analysis, Computer Vision and Image Understanding 104 (2-3) (2006) 90–126.

[2] T. Moeslund, E. Granum, A survey of computer vision-based human motion capture, Computer Vision and Image Understanding 81 (3) (2001) 231–268.

[3] A. Laurentini, The visual hull concept for silhouette-based image understanding, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (2) (1994) 150–162.

[4] K. Cheung, S. Baker, T. Kanade, Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 1 (2003) 77.

[5] B. Michoud, E. Guillou, H. Briceno, S. Bouakaz, Real-Time Marker-free Motion Capture from multiple cameras, in: IEEE 11th International Conference on Computer Vision, 2007. ICCV 2007., 2007, pp. 1–7.

[6] L.F. Bastos and J. Tavares, Matching of objects nodal points improvement using optimization, Inverse Problems in Science and Engineering, 14 (5) (2006) 529–541.

[7] F.P.M. Oliveira and J.M.R.S. Tavares, Algorithm of dynamic programming for optimization of the global matching between two contours defined by ordered points, CMES: Computer Modeling in Engineering & Sciences, 31 (1) (2008) 1–11.

[8] F.P.M. Oliveira, J.M.R.S. Tavares, and C. Source, Matching contours in images through the use of curvature, distance to centroid and global optimization with order-preserving constraint, CMES: Computer Modeling in Engineering & Sciences, 43 (1) (2009) 91–110.

[9] C. Chu, O. Jenkins, M. Mataric, Markerless kinematic model and motion capture from volume sequences, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 2 (2003) 475.

[10] C. Menier, E. Boyer, B. Raffin, 3d skeleton-based body pose recovery, in: 3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), IEEE Computer Society, Washington, DC, USA, 2006, pp. 389–396.

[11] G. Brostow, I. Essa, D. Steedly, V. Kwatra, Novel Skeletal Representation for Articulated Creatures, in: Computer Vision - ECCV 2004, Vol. 3023 of LNCS, Springer, 2004, pp. 66–78.

[12] A. Torsello, E. Hancock, A skeletal measure of 2D shape similarity, Computer Vision and Image Understanding 95 (1) (2004) 1–29.

[13] X. Bai, L. Latecki, Path similarity skeleton graph matching, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (7) (2008) 1282–1292.

[14] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, S. W. Zucker, Shock graphs and shape matching, in: International Conference on Computer Vision, 1998, pp. 222–229.

[15] H. Sundar, D. Silver, N. Gagvani, S. Dickinson, Skeleton based shape matching and retrieval, Shape Modeling International (2003) 130 – 139.

[16] H. Whitney, A set of topological invariants for graphs, American Journal of Mathematics, 55 (1) (1933) 231–235.

[17] M. Pelillo, K. Siddiqi, S. Zucker, Many-to-many matching of attributed trees using association graphs and game dynamics, Lecture notes in computer science (2001) 583–593.

[18] A. Torsello, D. Hidovic-Rowe, M. Pelillo, Polynomial-time metrics for attributed trees, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (7) (2005) 1087–1099.

[19] R. Pinter, O. Rokhlenko, D. Tsur, M. Ziv-Ukelson, Approximate labelled subtree homeomorphism, Journal of Discrete Algorithms 6 (3) (2008) 480–496.

[20] J. Wang, K. Zhang, Finding similar consensus between trees: an algorithm and a distance hierarchy, Pattern Recognition 34 (1) (2001) 127–137.

[21] K. Tai, The Tree-to-Tree Correction Problem, Journal of the ACM 26 (3) (1979) 422–433.

[22] T. Jiang, L. Wang, K. Zhang, Alignment of trees - an alternative to tree edit, in: CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Springer-Verlag, London, UK, 1994, pp. 75–86.

[23] J. Jansson, A. Lingas, A fast algorithm for optimal alignment between similar ordered trees, LNCS 2089 (2001) 232–240.

[24] E. Tanaka, K. Tanaka, The tree-to-tree editing problem., International Journal of Pattern Recognition and Artificial Intelligence 2 (2) (1988) 221–240.

[25] G. Valiente, An efficient bottom-up distance between trees, International Symposium on String Processing and Information Retrieval 0 (2001) 0212.

[26] K. Zhang, Algorithms for the constrained editing distance between ordered labeled trees and related problems, Pattern Recognition 28 (3) (1995) 463–474.

[27] S. Selkow, The Tree-to-Tree Editing Problem, Information Processing Letters 6 (6) (1977) 184–186.

[28] W. Yang, Identifying Syntactic differences Between Two Programs, Software - Practice and Experience 21 (7) (1991) 739–755.

[29] J. Wang, K. Zhang, C. Chang, Identifying approximately common substructures in trees based on a restricted edit distance, Information Sciences: an International Journal 121 (3-4) (1999) 367–386.

[30] J. Wang, K. Zhang, G. Chang, D. Shasha, Finding approximate patterns in undirected acyclic graphs, Pattern Recognition 35 (2) (2002) 473 – 483.