Exploring Heuristic and Optimum Branching Algorithms for Image Phylogeny

Zanoni Dias, Siome Goldenstein, and Anderson Rocha

Institute of Computing, University of Campinas
Av. Albert Einstein, 1251
Campinas, São Paulo, Brazil, 13083-970
{zanoni, siome, anderson}@ic.unicamp.br

Abstract

Currently, multimedia objects can be easily created, stored, (re)-transmitted, and edited for good or bad. In this sense, there has been an increasing interest in finding the structure of temporal evolution within a set of documents and how documents are related to one another overtime. This process, also known in the literature as Multimedia Phylogeny, aims at finding the phylogeny tree(s) that best explains the creation process of a set of near-duplicate documents (e.g., images/videos) and their ancestry relationships. Solutions to this problem have direct applications in forensics, security, copyright enforcement, news tracking services and other areas. In this paper, we explore one heuristic and one optimum branching algorithm for reconstructing the evolutionary tree associated with a set of image documents. This can be useful for aiding experts to track the source of child pornography image broadcasting or the chain of image distribution in time, for instance. We compare the algorithms with the state-of-the-art solution considering 350,000 test cases and discuss advantages and disadvantages of each one in a real scenario.

Keywords: Image Phylogeny Tree, Image Phylogeny, Image Dependencies, Image's Ancestry Relationships, Near-Duplicate Detection and Recognition, Near-Duplicates Kinship

1. Introduction

Nowadays, popular images and videos spread out at a rapid pace on the web through blogs, news sites, and social media, described by the popular expression *to go viral*. It is straightforward to find exact duplicates among available media, but many types of media objects, such as images and videos, can suffer small modifications during the redistribution that can change them without interfering on their semantic meaning. Some example include A/D or D/A conversions, (de)-coding, noise due to transmission, and small editing/corrections as brightness adjustments, and cropping. These are called near-duplicates, and are part of an active research area [1, 2, 3, 4, 5].

While most of these changes are natural and not necessarily harmful, sometimes the distribution itself may cause copyright infringement or even be a criminal action [6]. In some situations, the spreading pattern of an image or video can help companies to understand demographics and effectiveness of an ad campaign or a product.

These scenarios motivated the advent of a new research subfield called Multimedia Phylogeny [7, 8, 9, 10, 11], with the objective of investigating the history and evolutionary process of digital objects.

Multimedia Phylogeny goes beyond the detection of near-duplicate objects which only seeks to determine if a set of objects are near-duplicates of one another. In Multimedia Phylogeny, we are interested in finding the structure of modifications of a set of multimedia objects, including their causal and ancestry relationships, source of modifications (root or patient zero) of a set of related documents, reconstructing the order and transformations that originally created the near-duplicate set [7, 8, 9, 10, 11]. Figure 1 illustrates one case for images. The underlying question in Multimedia Phylogeny is how we find the relationships between each pair of images to point out which document generated another one across time and which transformation parameters were used.

Solutions to this problem have applications in different areas, such as the ones listed bellow:

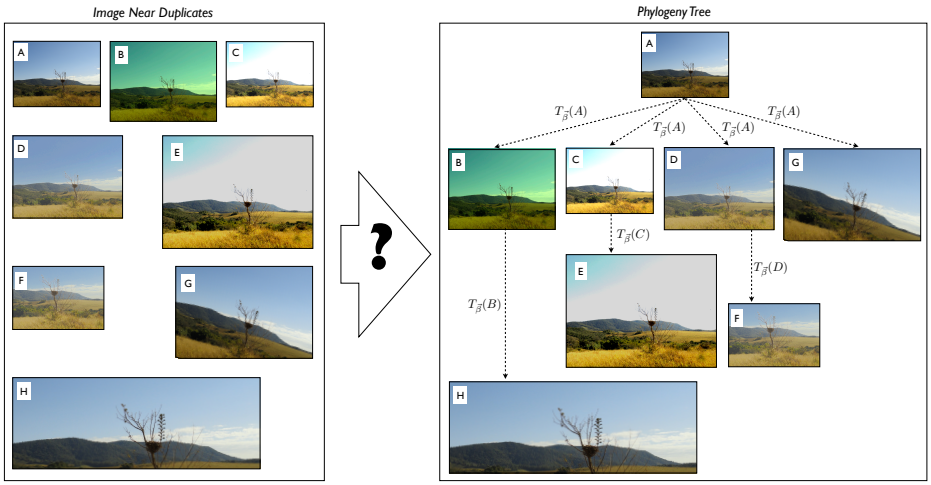


Figure 1: An example of set of near duplicate images and the underlying tree structure pointing out the directions of content modification across time and the underlying parameters for such modifications. The *fundamental question* of Multimedia Phylogeny (question mark in the figure) is how to find the transformation parameters for each pair of images and how to compare them to correctly find the ancestry relationships.

- **Security:** the modification graph of a set of documents might provide information of suspects' behavior, and point out the directions of online content distribution.
- **Forensics:** better results might be achieved if the forensic analysis is performed in the original document instead of on a near-duplicate [6, 12]. In addition, forensic experts might focus their attention on individuals associated with the redistribution of content closer to the root of the tree as they are more likely to be the ones who created the content at first place.
- **Copyright enforcement:** traitor tracing without the requirement of active source control solutions such as the ones based on either watermarking or fingerprinting approaches.
- **News tracking services:** Multimedia Phylogeny might be a valuable tool for mining applications. The near-duplicate relationships can feed news tracking services with key elements for determining the opinion formation process across time and space [7, 8].
- **Information Retrieval:** content-based retrieval systems might be developed to allow the display of similar photographs coming from different photographers without any metadata analysis

Current approaches state that there are two steps necessary to reconstruct the phylogeny tree. The first one consists of calculating a dissimilarity matrix whose entries compare every two documents (e.g., images) in a set. For the second step, this matrix is then used to reconstruct the tree. In this sense, De Rosa et al. [8] make the decision at every pair of images on what should be the right direction, while Dias et al. [9, 11] have a global method to construct a minimum spanning tree.

Following the same methodology discussed in previous works, in this paper we explore one heuristic and one optimum branching algorithm for reconstructing the evolutionary tree associated with a set of image documents. The heuristic solution is inspired on Prim's classic minimum spanning tree algorithm [13] while the optimum branching algorithm is based on the classical Chu-Liu, Bock and Edmonds algorithm [14, 15, 16]. We compare both methods to the state-of-the-art Oriented Kruskal algorithm [11] using the same dissimilarity matrices for image near-duplicate sets and discuss the pros and cons of each approach.

2. Related Work

In the past decade, we have seen an increasing progress on the development of efficient and effective systems to identify the cohabiting versions of a given document in the wild [4, 17, 18]. However, only recently there were first attempts to go beyond the detection of near duplicates, focused on identifying the structure of relationships within a set of near-duplicates.

During its lifetime, a multimedia object might undergo different processing stages whereby each processing operator might alter the underlying features of the object's content in a characteristic and detectable manner. Therefore, establishing relationships between pairs

of digital objects by analyzing their content is challenging due to the wide variety and diversity of processing operators that can be applied in the process. However, it is still possible due to the telltales left behind by the operators. In this regard, Multimedia Phylogeny aims at collecting these image telltales to study the evolutionary processes to which multimedia objects are subject [11].

Kennedy et al. [7] were the first to deal with the problem of parent-child relationships for pairs of images. Their work proposed a technique based on a visual migration map (VMM) for the detection of plausible parent-child relationships within a set of images. The VMM represents an approximation of the history of the images. However, the authors did not discuss how to find possible parameters for the family of transformations that lead a parent image to its resulting offspring.

Different from the VMM approach proposed in [7], De Rosa et al. [8] aimed at detecting the image dependencies within a set of images by considering the images' mutual information. This is expressed in terms of the sum of the mutual information between the content-based components and the content-independent components between them.

Dias et al. [9] introduced and formally defined the problem of Image Phylogeny Tree reconstruction. The authors proposed to find the structure of transformations and their parameters that generated a given set of near-duplicate images. This work has been expanded upon in [11], which presented an initial solution for dealing with more than one tree. In [10], the authors showed the applicability of phylogeny tree algorithms to the context of videos and introduced a first solution to reconstruct the evolutionary tree of a set of near-duplicate videos.

Kender et al. [19] explored content-based relationships for video clips downloaded from YouTube and related to the same event. The authors illustrated the construction of a content-dependency graph, whose structure is justified from a "genetic" standpoint by representing how videos evolve in terms of mutations, crossover and other operators.

Lastly, determining the temporal signatures associated with images has also been investigated by Mao et al. [20] and by Fan and Queiroz [21]. The first discussed an information theoretic approach for device temporal forensics while the latter discussed the identification compression history associated with a bitmap image. However, none of these approaches aimed at finding the structure of modifications within a set of documents.

3. Review on Multimedia Phylogeny

In this section, we present a formal definition of phylogeny tree and discuss how we can build it from a set of image transformation estimations and a tree building algorithm.

3.1. Image Phylogeny Tree

An *Image Phylogeny Tree* shows the structure of transformations and the evolution of a set of near-duplicate images. Usually, the tree reconstruction algorithm builds upon an initial set of near-duplicates and a dissimilarity function d that yields small values for ordered pairs that are likely father and son on the tree, and large values for ordered pairs that are unlikely so.

Let $T_{\vec{\beta}}$ be an image transformation from a family $\mathcal{T}$. We can define a dissimilarity function between two images $\mathcal{I}_A$ and $\mathcal{I}_B$ as the minimum

$$d_{\mathcal{I}_A,\mathcal{I}_B} = \left|\mathcal{I}_B - T_{\vec{\beta}}(\mathcal{I}_A)\right|_{\text{point-wise comparison method } \mathcal{L},} \quad (1)$$

for all possible values of $\vec{\beta}$ that parameterize $\mathcal{T}$. Equation 1 measures the amount of residual between the best transformation of $\mathcal{I}_A$ to $\mathcal{I}_B$, according to the family of operations $\mathcal{T}$, and $\mathcal{I}_B$ itself. The comparison is made possible using any point-wise comparison method $\mathcal{L}$ in the end.

With a set of n near-duplicate images, the first task for creating an image phylogeny tree is to calculate the dissimilarity between every pair of such images. In this case, it is necessary to consider a good set of possible image transformations, $\mathcal{T}$, from which one image can generate an offspring [8, 11, 22, 23]. The family of image transformations $\mathcal{T}$ that we consider in this paper is the same as in [11]: resampling, cropping, affine warping (including rotation, translation, and off-diagonal correction), brightness and contrast changes, and lossy compression using the standard lossy JPEG algorithm.

3.2. Creating the Dissimilarity Matrix

There are two core steps in the process of reconstructing an image phylogeny tree from a set of near-duplicate images: the dissimilarity function and the tree-building algorithm.

Following the methodology proposed in [11], for each possible pair of images $\mathcal{I}_A$ and $\mathcal{I}_B$ in a set of images, the first task consists of estimating the possible values of $\vec{\beta}$ that parameterizes $\mathcal{T}$ and best approximates image $\mathcal{I}_A$ onto $\mathcal{I}_B$ through the dissimilarity function $|\mathcal{I}_B - T_{\vec{\beta}}(\mathcal{I}_A)|_{\mathcal{L}}$ (Eq. 1). To achieve this, we follow the steps listed bellow:

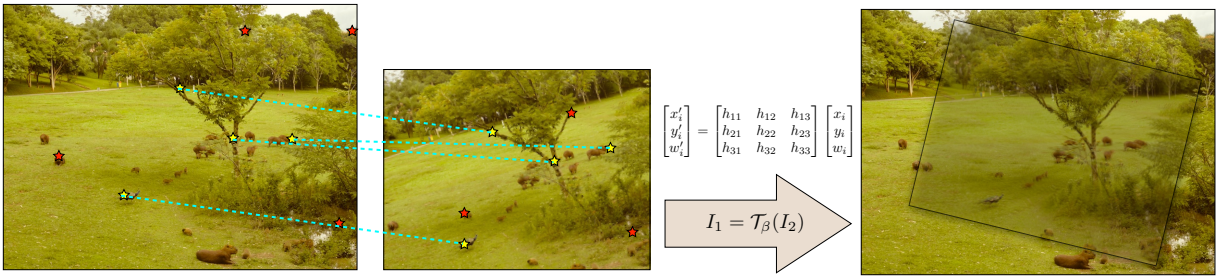


Image I_1 Image I_2 Mapping $I_2 \mapsto I_1$

Figure 2: To calculate image dissimilarities between a pair of images I_1 and I_2, we find robust points of interest in both images. For the good matches (yellow stars), we calculate an homography matrix representing the necessary parameters to transform one image to another's domain. Once we perform the mapping, we can compare both images pixel wise within the region of interest they overlap.

1. calculate the corresponding points between images $\mathcal{I}_A$ and $\mathcal{I}_B$ using the Speeded-Up Robust Features (SURF) algorithm [24];
2. robustly estimate the affine warping transformation parameters for image $\mathcal{I}_A$ with respect to $\mathcal{I}_B$ taking the corresponding points into consideration and using RANSAC algorithm [25];
3. calculate the mean and variance of each $\mathcal{I}_B$'s color channel and normalize image $\mathcal{I}_A$'s color channels using such measures;
4. compress the result of Steps 2 and 3 according to $\mathcal{I}_B$'s quantization table;

The first step gives candidate points to estimate the resampling and cropping operations using points robustly filtered using RANSAC (Step 2) as illustrated in Figure 2. With such points, we can find the homography matrix

$$H_{ab} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2}$$

which maps points from image I_1 to image I_2. In other words, given a pixel $p_a = [x_a, y_a, w_a]$ in image I_1, it will be mapped to pixel $p'_b = [x_b, y_b, w_b]$ through the operation

$$p'_b = H_{ab} p_a \tag{3}$$

Appendix B shows more details about calculating the homography matrix using robust interest points as found in Step 2. The appendix also shows one simple algorithm for finding such homography matrix.

In Step 3, we perform pixel intensity normalization of image $\mathcal{I}_A$ according to the $\mathcal{I}_B$ color channels' mean and variance. Step 4 compresses image $\mathcal{I}_A$ according to $\mathcal{I}_B$'s quantization table. Finally, we uncompress both of them and calculate their point-wise dissimilarity using the standard *Minimum Squared Error* (MSE) on the domain of the target image. Figure 2 depicts the process of mapping one image to another image's domain.

Once a dissimilarity matrix is created, different tree building algorithms can be used to recover the ancestry relationships for a given set of images. In this paper, we explore two of these algorithms and contrast them to state-of-the-art Oriented Kruskal algorithm [11].

4. New Methods for Reconstructing an Image Phylogeny Tree

This paper's main contribution relies on exploring two methods for reconstructing an image phylogeny tree from a set of near-duplicate images. Both approaches operate upon a set of n near-duplicate images. To build the dissimilarity matrix M, we use the same setup proposed in [11].

4.1. Method #1 – Best Prim Algorithm

As the authors in [11] extended upon the classical Kruskal minimum spanning tree algorithm [26] for finding the phylogeny tree associated with a set of near-duplicate images, one might wonder what happens if we use a similar algorithm such as the one proposed by Prim [13].

Therefore, for the first alternative, we explore a heuristic for building the tree based on the classical Prim minimum spanning tree algorithm [13] which we call Best Prim. The algorithm initially receives a dissimilarity matrix M that captures the dissimilarities between each pair of n possible near-duplicates. For each possible root out of n, it reconstructs the tree rooted at the chosen node and calculates the cost for building such a tree using an auxiliary method called Oriented Prim. After evaluating all possible roots, the algorithm

chooses the one with the lowest reconstruction cost as the final image phylogeny tree.

Figure 4 depicts the execution of the algorithm for a toy example with $n = 6$ image duplicates for the dissimilarity matrix depicted in Figure 3. In the first iteration, Best Prim finds the tree $[1, 1, 6, 2, 4, 4]$ whose cost (sum of dissimilarities) is 118 (31 + 23 + 15 + 27 + 22).

M	1	2	3	4	5	6
1	-	31	57	37	45	49
2	31	-	33	23	29	32
3	51	41	-	42	37	38
4	16	36	28	-	15	27
5	35	18	54	30	-	54
6	12	40	22	60	19	-

Figure 3: An example of a dissimilarity matrix used by the Best Prim reconstruction algorithm.

In this tree representation, each element e_i in $[\cdot]$ identifies the parent of a node with `id=i`. For instance, the tree $[1, 1, 6, 2, 4, 4]$ represents a tree with six vertices such that vertex 1 is the root of the tree and also the parent of vertex 2, which in turn, is the parent of vertex 4. Vertex 6 is the parent of vertex 3. Finally, vertex 4 is the parent of vertices 5 and 6. This tree is represented in Figure 4(a).

In the second iteration, Best Prim finds the tree $[4, 2, 6, 2, 4, 4]$, whose cost is 103 (23 + 15 + 16 + 27 + 22), and better than the previous one. The process continues until the algorithm finds the last possible tree rooted at vertex 6, $[6, 5, 6, 2, 6, 6]$, whose cost is 94 (12 + 19 + 18 + 22 + 23), better than any of the previous trees and, therefore, it is the resulting image phylogeny tree (IPT).

In summary, Best-Prim algorithm builds n different trees using the Oriented Prim algorithm as a subroutine with each of the n vertices as a root. Then it chooses the lowest cost tree, that is, the one with the lowest dissimilarity sum across the edges of the tree. Its running time is $O(n^3)$ (since the algorithm performs n calls to Oriented Prim which runs in $O(n^2)$). These steps are summarized in Algorithm 1.

4.1.1. Oriented Prim Definition

The Oriented Prim algorithm requires a dissimilarity matrix M built upon a set of n near-duplicate images and a node in which the reconstructed tree will be rooted (`root`). Lines 1–5 of Algorithm 2 initialize the `tree`, `dist` and `edge` vectors which store the tree to be reconstructed, the minimum distance of a node to the current partial reconstructed tree and the edge with minimum distance to the current partially reconstructed tree,

Algorithm 1 Best Prim

Input: number of near-duplicate images n, and an $n \times n$ dissimilarity matrix M

```
1: minCost ← ∞
2: for i ∈ [1..n] do
3:     [tree, cost] ← OrientedPrim(M, n, i)
4:     if (cost < minCost) then
5:         minCost ← cost
6:         minTree ← tree
7:     end if
8: end for
9: return minTree
```

respectively. Line 6 sets the tree's root, Line 7 initializes the total tree's cost and Line 8 initializes the number of edges already included in the partially reconstructed tree.

At each iteration of the main loop (Lines 9–26), a new vertex is included in the tree through a new edge. The inner block (Lines 10–16) seeks the partially reconstructed tree's closest vertex. Lines 17–19 insert the new vertex in the tree. The last block (Lines 20–25) updates the `dist` and `edges` vectors taking into account the newest vertex inserted in the tree. Finally, Line 27 returns the reconstructed tree and its associated cost, which consists of the sum of dissimilarities across the tree's edges.

Algorithm 2 Oriented Prim

Input: number of near-duplicate images n, an $n \times n$ dissimilarity matrix M, and the root of the tree to be reconstructed *root*

```
 1: for i ∈ [1..n] do                                   ▷ Initialization
 2:     tree[i] ← 0
 3:     dist[i] ← M[root, i]
 4:     edge[i] ← root
 5: end for
 6: tree[root] ← root      ▷ Tree begins with one vertex and no edges
 7: cost ← 0
 8: n_edges ← 0
 9: for each n_edges < n − 1 do          ▷ While the tree is not complete
10:     min ← ∞                           ▷ Seeks the tree's closest edge
11:     for i ∈ [1..n] do
12:         if (dist[i] < min and tree[i] = 0) then
13:             new ← i
14:             min ← dist[i]
15:         end if
16:     end for
17:     tree[new] ← edge[new]  ▷ Puts the newest vertex on the tree
18:     n_edges ← n_edges + 1
19:     cost ← cost + min
20:     for i ∈ [1..n] do     ▷ Updates the distance and edges vectors
21:         if (tree[i] = 0 and dist[i] > M[new, i]) then
22:             dist[i] ← M[new, i]
23:             edge[i] ← new
24:         end if
25:     end for
26: end for each
27: return (tree, cost)
```

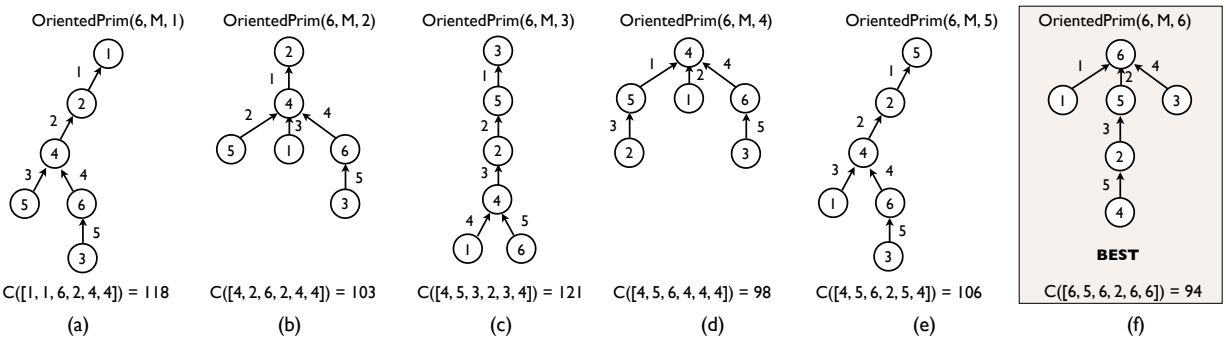


Figure 4: Simulation of the Best-Prim algorithm to construct an Image Phylogeny Tree from the Dissimilarity Matrix depicted in Figure 3. C([·]) represents the cost associated (sum of edges) with the tree in [·].

4.2. Method #2 – Chu-Liu, Bock and Edmonds Algorithm

In the literature, the Optimal Branching (OB) problem deals with the construction of minimum spanning trees on directed graphs with known roots. Solutions to this problem were proposed independently by Chu and Liu [14], Edmonds [15], and Bock [16].

The version proposed by Edmonds [15], is recursive and receives as input a graph G with n nodes and weights associated with the edges as well as a especial vertex r, considered the root of the branching. In practice, however, r is not known, which means there are two options: we can either run the algorithm independently n times and pick up the tree with the lowest total weigh cost (similar to Best Prim, Section 4.1) or we can devise an additional dummy node with a connection to every node in the tree and start the algorithm from such dummy node. We shall give more details about this strategy at the end of Section 4.2.1.

4.2.1. Chu-Liu, Bock and Edmonds Algorithm Definition

In the literature, there are a number of black-box implementations of the optimal-branching algorithm available. For the purpose of this paper, we consider a C++-implementation of Chu-Liu, Bock and Edmonds optimum branching algorithm as described by Tarjan [27][1].

The direct implementation of the algorithm relies on a conceptual recursive description. Let $G_i = (V_i, E_i)$ be an input graph in some recursive step of the algorithm and r to be the root of the tree.

Figure 5 depicts the execution of the algorithm for a toy example with $n = 6$ image near duplicates. The edge weight denotes the dissimilarities between the nodes (images).

The first step consists of assuming one node as a possible root, similar to Oriented Prim. Suppose, for instance, that node $r = 1$ is such node. Then, for each node v in the graph other than r, we select the edge arriving at v with the lowest cost. By doing so, we end up with the solid-line edges in Figure 5(a). If there is no circuit in the structure found, we return such branching. If there is a circuit, we deal with it by creating a dummy node, say v_{C_i} representing it. For each edge connecting a node x outside the circuit and a node y in the circuit, we update its weight by taking into consideration the edge weight $w(x, y)$ plus the lowest edge weight in the circuit minus the edge weight of the edge arriving at y.

For instance, in Figure 5 the edge $(x, y) = (2, 4)$ with $w(2, 4) = 8$ is updated to edge $(2, v_{C_i})$ with weight $w(2, v_{C_i}) = 8+3-4$ because 3 is the lowest cost in the circuit and 4 is the cost of the edge arriving at $y = 4$. The same happens with the edge $(x, y) = (3, 5)$ which becomes the edge $(3, v_{C_i})$ with weight $w(3, v_{C_i}) = 11+3-6$. We keep the edges connecting nodes outside the circuit. However, for edges between a node y in the circuit and another x outside the circuit, we add an edge between the dummy node v_{C_i} and x with the weight of $w(y, x)$. For instance, in Figure 5, the edge $(y, x) = (6, 2)$ connects a node within the circuit and a node outside the circuit. Therefore, we create the edge $(v_{C_i}, x) = (v_{C_i}, 2)$ with weight $w(v_{C_i}, 2) = w(6, 2) = 7$.

After processing all nodes within and outside the circuit, we have the updated graph depicted in Figure 5(b). Then, the algorithm recursively finds the optimal branching A_{i+1} in this updated graph. This results in the selection of the edges marked in solid lines in Figure 5(b), namely the edges (1,2), (2,v_{C_i}), and (2,3).

Next, the algorithm needs to cope with circuit it-

[1] `http://sourceforge.net/projects/edmonds-alg/` and detailed in [27]

self. This is achieved by first discarding the edge in the current optimum branching connecting a node x to the dummy node C. In this case, $(x, v_{C_i}) = (2, 3)$ and updates such edge with the correct one in the original graph $(x, y) = (2, 4)$. In addition, the algorithm updates all edges in the optimum branching within the circuit that do not arrive in y, i.e., edges (4,5) and (5,6).

Finally, the algorithm just needs to update possible edges from nodes within the circuit to outside that are part of the optimum branching. In this example, there are none. The algorithm ends up with the graph in the Figure 5(c) with the solid lines highlighting the edges part of the optimum branching. We give algorithmic details of such step-by-step procedure in Algorithm 3.

In terms of complexity, given a graph $G_i = (V_i, E_i)$, we can obtain A'_i in time $O(V_i + E_i)$. If we are given an optimum branching A_{i+1}, we can obtain A''_i in time $O(V_i)$. Therefore, the algorithm's complexity is given by the recurrence $T(V_i, E_i) = T(V_{i+1}, E_{i+1}) + O(V_i + E_i)$. As in each recursive call the number of vertices is reduced in, at least, one unit, ($|V_{i+1}| < |V_i|$), resulting in $T(V, E) = O(V(V+E))$. In our case, as $E = \Theta(V^2)$ (complete graph), we have that $T(V, E) = O(V^3)$. However, it is worth noting that there exists a faster implementation provided by [27], for dense graphs, which runs in $O(V^2)$.

Finally, note that we can use this algorithm to obtain an optimum branching for a graph $G = (V, E)$ the root of the tree is unknown In this case, we have at least two ways of achieving this objective. The first one, used for Best Prim Algorithm in Section 4.1, we can run the algorithm $|V_i|$ times, each time with a different node as a possible root, and choose the one with the minimum cost. The second one, consists of inserting a dummy node v_0 as such root and make the correct changes to allow us to eliminate such node later without any loss (ensuring the edges between such node and any other will never be selected as part of the optimum branching). In this case, it is enough to make $V_0 = V \cup \{v_0\}$ and $E_0 = E \cup \{(v_0, v)\}$ for each vertex $v \in V$ with weight $w(v_0, v) = K$ where K is a sufficiently large number (e.g., the absolute sum of the weights of all edges in the graph) and run the discussed algorithm using as input $G_0 = (V_0, E_0)$ and v_0 as the root of the optimum branching. Then, when we obtain an optimum branching A_0 from the algorithm, we just need to do $A = A_0 \backslash (v_0, v)$, where (v_0, v) is the only edge involving the vertex v_0 and A is the final optimum branching for the graph G.

5. Experiments and Methodology

We follow the methodology introduced by Dias et. al. [11] for the validation of the algorithms in this paper.

5.1. Evaluation Metrics

As in [11], we look at four different quantitative metrics (*Root*, *Edges*, *Leaves*, and *Ancestry*) to evaluate a reconstructed tree in scenarios where the Ground Truth is available.

Root: $R(\text{IPT}_1, \text{IPT}_2) = \begin{cases} 1, & \text{If } \texttt{Root(IPT}_1\texttt{)} = \texttt{Root(IPT}_2\texttt{)} \\ 0, & \text{Otherwise} \end{cases}$

Edges: $E(\text{IPT}_1, \text{IPT}_2) = \frac{|E_1 \cap E_2|}{n-1}$

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

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

To describe the above evaluation metrics, we use the illustrative example of Figure 6 of an original tree and its reconstructed version. An original image phylogeny tree, IPT_1, and its reconstructed version, IPT_2 are given.

Considering the *Root* metric, the reconstructed tree has the correct root node. According to the *Edges* metric, the reconstructed tree has $E = \frac{|\{(1,6),(6,3)\}|}{5} = \frac{2}{5} = 40\%$ correct edges. In addition, the reconstructed tree has $L = \frac{|\{4\}|}{|\{1,2,4,5\}|} = \frac{1}{4} = 25\%$ of correct leaves. Finally, the reconstructed tree has $A = \frac{7}{10} = 70\%$ correct ancestors, where common ancestry are $\{(5 \rightarrow 3), (2 \rightarrow 3), (1 \rightarrow 6), (1 \rightarrow 3), (4 \rightarrow 6), (4 \rightarrow 3), (6 \rightarrow 3)\}$. This metric measures how many times an ancestor x of a node y is correctly found in the tree. We denote an ancestry by $(y \rightarrow x)$. A node x is an ancestor of y in the tree if there exists a path from y to x in the tree. For instance, $(5 \rightarrow 3)$ is a correct ancestry pair since node $x = 3$ is the ancestor of node $y = 5$ in Figure 6's trees.

5.2. Experimental Protocol

We compare the two algorithms Best Prim and Chu-Liu, Bock and Edmonds to the Oriented Kruskal [11] on a controlled environment with full trees and with missing links. In both experiments, we assess the quality of the results using the metrics devised in Section 5.1.

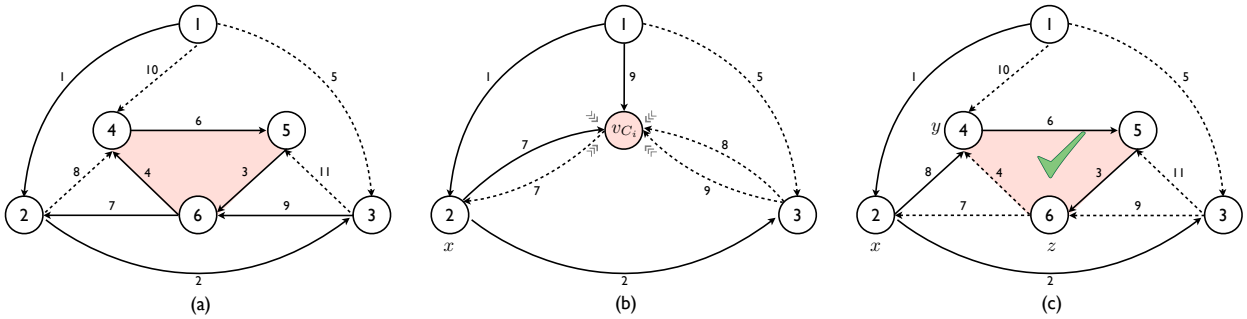
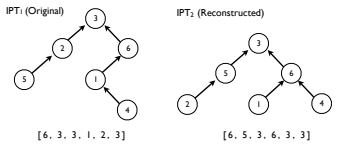


Figure 5: Simulation of the Chu-Liu, Bock and Edmonds algorithm to construct an Image Phylogeny Tree from a given Dissimilarity Matrix.



Trees (node points to its parent)	
$\text{IPT}_1 = [6, 3, 3, 1, 2, 3]$	$\text{IPT}_2 = [6, 5, 3, 6, 3, 3]$
Root	
$\text{Root}(\text{IPT}_1) = 3$	$\text{Root}(\text{IPT}_2) = 3$
Edges (oriented edges of children to parents)	
$E_1 = \{(1,6),(2,3),(4,1),(5,2),(6,3)\}$	
$E_2 = \{(1,6),(2,5),(4,6),(5,3),(6,3)\}$	
Leaves	
$L_1 = \{5,4\}$	$L_2 = \{1,2,4\}$
Ancestry	
$A_1 = \{(5 \to 2),(5 \to 3),(2 \to 3),(4 \to 1),(4 \to 6),(4 \to 3)$ $(1 \to 6),(1 \to 3),(6 \to 3)\}$	
$A_2 = \{(2 \to 5),(2 \to 3),(5 \to 3),(1 \to 6),(1 \to 3),(4 \to 6)$ $(4 \to 3),(6 \to 3)\}$	

Figure 6: On the left, an example of an original tree encompassing an original image and its transformations to create near-duplicates. On the right, the reconstructed phylogeny tree of transformations and near-duplications. On the bottom, a table detailing the data structures necessary to run and evaluate the phylogeny algorithms.

1. **Controlled Environment with Complete Trees**. In this experiment, the ground truth of the image's relationships is known a priori. In addition, if we are evaluating the relationship of n near-duplicates, there are no missing links between two given nodes.
2. **Controlled Environment with Missing Links**. Similar to the previous experiment except now we can have trees with missing links: a parent node $\mathcal{I}_A$ have a child $\mathcal{I}_B$ and a grand-child $\mathcal{I}_C$ but the link (genealogical connection) from $\mathcal{I}_A$ to $\mathcal{I}_C$ is missing, that is $\mathcal{I}_B$ is not present in the set of available images. With this analysis, we want to verify if the introduced algorithms preserve the ancestry information with missing pieces of the evolution.

5.3. Data Sets

For the controlled scenario, we selected five possible transformations an image can undergo in order to generate a near-duplicate: JPEG re-compression with a different quality factor, resizing with different scales for horizontal and vertical axis, contrast adjustment, brightness adjustment, and gamma correction. The parameter ranges for such transformations are detailed in Appendix A.

To create the data set, we selected 50 images from the *Uncompressed Color Image Database* (UCID) data set [28][2] which contains a wide variety of images with 512×384-pixel resolution without compression artifacts. For reproducibility, we used the images with $\texttt{id} = i \times 25$ where $i \in [1..50]$.

For each image, we created trees with 10, 20, 30, 40, and 50 nodes (near-duplicates) from 50 different tree topologies. A tree with 50 nodes represents an original image with its forty-nine near-duplicates. For each topology, we created ten different set of parameters also randomly selected from the predefined parameter ranges. The final data set has 25,000 test cases for each tree size. As we select five possible tree sizes, the final data set has 125,000 test cases (50 images, 50 topologies, 10 sets of parameters, 5 different tree sizes).

For the experiments with missing links, we selected all trees with 50 nodes (25,000) and randomly created subsets of them with varying size ([5..45] with increments of 5) preserving the original root. Therefore, the data set for the experiment with missing links has

[2] http://www.staff.lboro.ac.uk/~cogs/datasets/UCID/ucid.html

Algorithm 3 Chu-Liu, Bock, and Edmonds' Algorithm

Input: $G_i = (V_i, E_i)$ and r, the root of the tree

$A'_i \leftarrow \emptyset$
for each vertex $v \in V_i$ other than r **do**
    $A'_i \leftarrow A'_i \cup$ edge $\{(u, v)\}$ with the lowest weight in E_i
end for each
if (A'_i does not contain a circuit) **then**
    $A_i \leftarrow A'_i$ ▷ Found an optimal branching
else
    $C_i \leftarrow$ **Find** the circuit $(v_1, v_2, \ldots, v_k) \in A'_i$.
    $V_{i+1} = V_i \backslash C_i \cup \{v_{C_i}\}$ ▷ v_{C_i} is a dummy node representing the shrinking of the circuit $C_i \subset A'_i$
    $E_{i+1} \leftarrow \emptyset$ ▷ Building E_{i+1}
    $w_{C_i} \leftarrow$ Find the lightest edge's weight $\in C_i$
    for each edge between a vertex $x \in V_i \backslash C_i$ and a vertex $y \in C_i$ **do**
        $w(x, v_{C_i}) \leftarrow w(x, y) - w(p(y), y) + w_{C_i}$ ▷ Updating the edge's weight. $p(y)$ is a vertex $\in C_i$ (i.e., $(p(y), y)$ is an edge in the circuit C_i)
        $E_{i+1} \leftarrow E_{i+1} \cup \{(x, v_{C_i})\}$ ▷ Adding the edge to E_{i+1}
    end for each
    for each edge between a vertex $x \in C_i$ and a vertex $y \in V_i \backslash C_i$ **do**
        $w(v_{C_i}, y) \leftarrow w(x, y)$ ▷ Update the edge's weight
        $E_{i+1} \leftarrow E_{i+1} \cup \{(v_{C_i}, y)\}$
    end for each
    for each edge between a vertex $x \in V_i \backslash C_i$ and a vertex $y \in V_i \backslash C_i$ $(x \neq y)$ **do**
        $E_{i+1} \leftarrow E_{i+1} \cup \{(x, y)\}$ ▷ Add the edge (x, y) to E_{i+1} and preserve its weight as in E_i
    end for each
    $G_{i+1} \leftarrow (V_{i+1}, E_{i+1})$
    $A_{i+1} \leftarrow$ **Recursively** find the optimal branching of G_{i+1}
    Let x be the only node such that the edge $(x, v_{C_i}) \in A_{i+1}$
    Let y be the node $\in C_i$ such that $\exists$ an edge $(x, y) \in E_i$ corresponding to the edge $(x, v_{C_i}) \in E_{i+1}$
    $A''_i \leftarrow A_{i+1} \backslash \{(x, v_{C_i})\}$
    $A''_i \leftarrow A''_i \cup \{(x, y)\}$
    $A''_i \leftarrow A''_i \cup \{(w, z)$, such that $(w, z) \in C_i \; \forall \; z \neq y$ ▷ Adding all edges in the circuit other than the one arriving at the node $y\}$
    for each edge $(v_{C_i}, t) \in A''_i$ **do** ▷ Replacing the dummy node v_{C_i} back with the ones originally in C_i with their respective edges
        $A''_i \leftarrow A''_i \backslash \{(v_{C_i}, t)\}$
        $A''_i \leftarrow A''_i \cup \{(z, t)\}$, such that $(z, t) \in E_i$ is the edge corresponding to the edge $(v_{C_i}, t) \in E_{i+1}$
    end for each
    $A_i \leftarrow A''_i$
end if
return A_i as the optimal branching in G_i

225,000 test cases (25,000 trees × 9 missing links setups) plus the test cases comprising complete trees with 50 nodes (25,000). Then, we reconstruct the tree with the remaining nodes and evaluate the reconstruction process using the same metrics we designed in Section 5.1.

All the test cases for complete trees and for missing links are exactly the same as the ones discussed in [11]. All the necessary information to obtain the data sets will be available online[3].

5.4. Controlled Environment with Complete Trees

In this scenario, we tested all algorithms considering a total of 125,000 test cases with complete trees.

Figure 7 depicts the summary results for the two algorithms discussed in this paper in comparison to Oriented Kruskal [11]. In the chart, the x-axis denotes the number of nodes in the tested tree while the y-axis denotes the percentage of correct reconstruction (correct score) according to the four metrics used (Root, Edges, Leaves, and Ancestry).

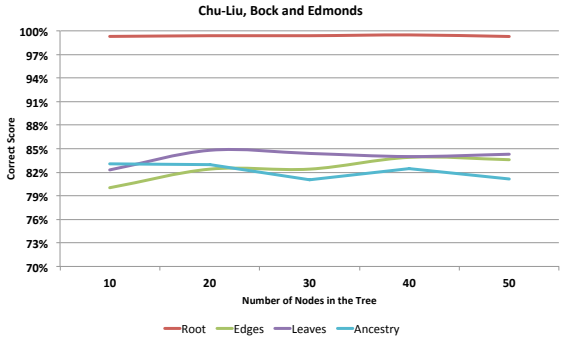
For instance, for trees with 10 nodes, the Oriented Kruskal (OK) correctly finds the root in 98.6% percent of the cases while Best Prim (BP) finds it in 90.9% of the cases. The Chu-Liu, Bock and Edmonds (CLBE) algorithm outperforms both approaches and finds the correct root in 99.3% of the cases. As we consider 125,000 test cases in the validation, CLBE outperforms OK for finding the correct root in 875 cases.

To directly compare the three algorithms, we define their error Δ_{Error} with respect to each metric. For instance, the $\Delta_{RootError}$ of CLBE with respect to OK for the root metric is given by

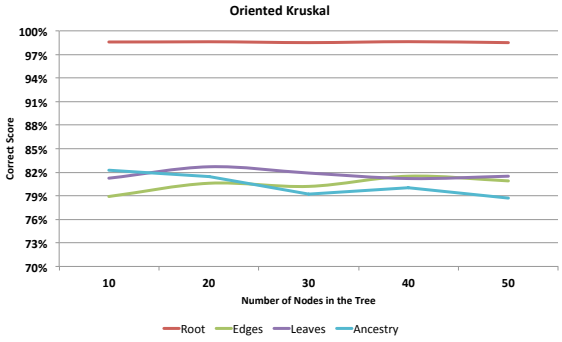
$$\Delta_{RootError} = \left(\frac{1 - CLBE_{Root}}{1 - OK_{Root}} \right) - 1. \qquad (4)$$

According to this definition, Table 1 compares the two best performing algorithms: Chu-Liu, Bock and Edmonds (CLBE) and Oriented Kruskal (OK). From the table, we see how CLBE algorithm clearly outperforms OK regardless the number of nodes in the tree and independent of the considered metric. Running a McNemar

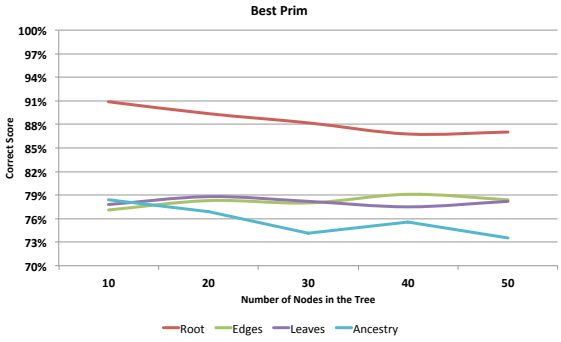
[3] http://www.ic.unicamp.br/~rocha/pub/communications.html



(a) Chu-Liu, Bock and Edmonds algorithm.



(b) Oriented Kruskal.



(c) Best Prim.

Figure 7: Summary results for Chu-Liu, Bock and Edmonds, Oriented Kruskal and Best-Prim algorithms for image phylogeny tree reconstruction considering the scenario with complete trees (no missing links).

test for the root metric (binary variable) and a Wilcoxon signed-rank test for the other three metrics, we have that all differences are statistically significant at 99% confidence level. All differences with respect to Best Prim are also significant.

On average, CLBE algorithm reduces the error in about 10% for edges, leaves and ancestry metrics while reducing the error for finding the root in about 50%. In a forensic scenario, such reduction of error in ancestry and root is an important achievement since, for instance, it might lead to a better graph representing suspect's behavior (good ancestors representation) and also point out the root of a potential online attack with more certainty (better root finding).

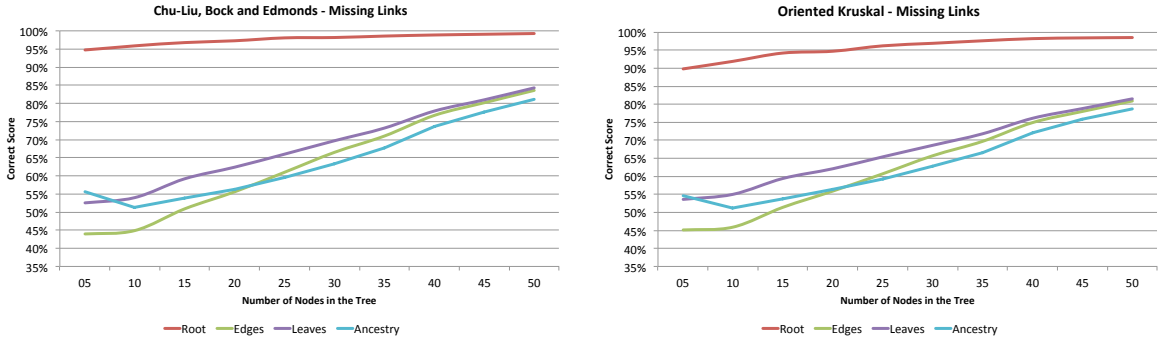
5.5. Controlled Environment with Missing Links

In this section, we evaluate the effectiveness of the algorithms with respect to the reconstruction of the phylogeny tree when some of the links are missing. In other words, we intend to find the ancestor connections between one image and a descendant even if some of the connections between them are missing. For example, in Figure 1, suppose image C is missing and we still want to be able to find the relationship between image A and E (an indirect descendant of A).

Size	$\Delta_{RootError}$	$\Delta_{EdgesError}$	$\Delta_{LeavesError}$	$\Delta_{AncestryError}$
10	-51.60%	-5.21%	-5.85%	-4.52%
20	-56.22%	-9.28%	-12.14%	-8.11%
30	-59.46%	-11.11%	-13.81%	-8.65%
40	-63.24%	-12.97%	-14.89%	-12.50%
50	-52.83%	-14.14%	-15.14%	-11.27%

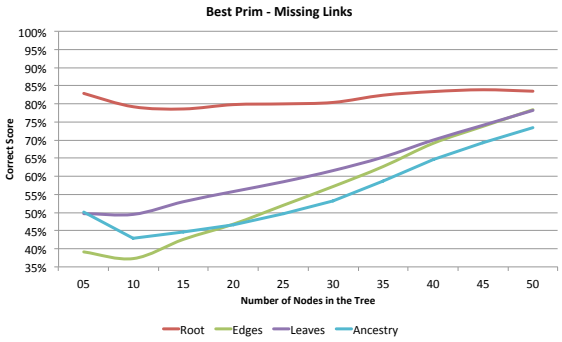
Table 1: Reduction in error of Chu-Liu, Bock and Edmonds algorithm with respect to Oriented Kruskal [11] considering the scenario with complete trees.

Here, as we discussed in Section 5.3, we have a data set with missing links comprising 250,000 test cases (25,000 trees $\times$ 9 missing links setups). For each tree, we randomly select some nodes to remove (preserving the original root) and generate a tree with some missing connections. Then, we reconstruct the tree with the remaining nodes and evaluate the reconstruction process using the metrics we designed in Section 5.1 to compare trees.



(a) Chu-Liu, Bock and Edmonds.

(b) Oriented Kruskal.



(c) Best Prim.

Figure 8: Summary results for Chu-Liu, Bock and Edmonds, Oriented Kruskal and Best-Prim algorithms for image phylogeny tree reconstruction considering the scenario with missing links.

Figure 8 depicts the summary results for the two algorithms discussed in this paper compared to OK [11] in the scenario with missing links.

In the chart, the x-axis denotes the number of nodes remaining in the tested tree after removing $(50 - x)$ nodes. For instance, $x = 5$ means we are evaluating a tree with 45 missing nodes in the image phylogeny tree, while $x = 50$ is the complete tree. The y-axis denotes the percentage of correct reconstruction (correct score) according to the four metrics.

Consider, for instance, the case for trees with 20 missing links ($x = 30$ in the chart). In this case, the BP algorithm correctly finds the root in 80.4% of the cases while OK finds it in approximately 96.9% of the cases. The CLBE algorithm, on the other hand, outperforms both approaches finding the correct root in 98.2% of the cases, a reduction in error of $\Delta_{RootError} = -41.9\%$ with respect to OK or $\Delta_{RootError} = -90.8\%$ with respect to Best Prim.

Even more important for a forensic scenario, the CLBE algorithm outperforms BP and OK in the other metrics as well for varying number of missing nodes in the phylogeny tree. Therefore CLBE algorithm can be considered a promising method for image phylogeny reconstruction with respect to the state-of-the-art OK or even the extension we discuss herein for the Prim minimum spanning tree algorithm.

Table 2 compares the two best performing algorithms: CLBE and OK, with the former algorithm outperforming the latter in all cases (except the ones in red). Running a McNemar test for the root metric (binary variable) and a Wilcoxon signed-rank test for the other three metrics, we have that all differences are statistically significant at 99% confidence level in favor of CLBE, except the ones highlighted in red which show either a statistically significant difference in favor of OK or no statistical difference.

We can observe that, regardless the number of missing links, CLBE outperforms OK for finding the root and the ancestors. On average, CLBE algorithm reduces the error in about 5% for ancestry, while reducing the error for finding the root in about 40%. The contrary happens for edges and leaves when we have a scenario with a high number of missing links (sizes 5, 10, 15, for

instance), with OK performing best for these particular cases.

Size	$\Delta_{RootError}$	$\Delta_{EdgesError}$	$\Delta_{LeavesError}$	$\Delta_{AncestryError}$
05	-48.0%	2.2%	2.4%	-2.0%
10	-49.4%	1.8%	2.2%	-0.2%
15	-44.8%	1.0%	0.5%	-0.2%
20	-49.1%	0.7%	-0.8%	0.0%
25	-50.0%	-0.8%	-1.7%	-0.7%
30	-41.9%	-2.3%	-3.5%	-1.6%
35	-41.7%	-4.3%	-5.0%	-3.6%
40	-38.9%	-7.2%	-7.5%	-5.7%
45	-43.8%	-10.0%	-10.4%	-7.9%
50	-53.3%	-14.1%	-15.1%	-11.3%

Table 2: Reduction in error of Chu-Liu, Bock and Edmonds Algorithm with respect to Oriented Kruskal [11] considering a scenario with missing links.

6. Conclusions and Future Work

In this paper, we discussed two algorithms: a heuristic directed oriented graph extension over the classic Prim minimum spanning tree algorithm (Best-Prim) and one optimum branching algorithm for reconstructing the evolution tree associated with a set of images (Chu-Liu, Bock and Edmonds). To our knowledge, this is the first time such approaches are discussed for this problem with such a complete validation. In addition, this is the first time an exact algorithm (Chu-Liu, Bock and Edmonds) is explored for the image phylogeny problem showing promising results when compared with existing heuristic-based solutions in the related phylogeny literature. We compared both algorithms to the state-of-the-art Oriented Kruskal recently proposed by Dias et al. [11].

While Best Prim seems a natural extension to the Oriented Kruskal algorithm, we have found that it is not a competitive solution for the image phylogeny problem. On the other hand, the heuristic-based Oriented Kruskal is outperformed by the optimum branching Chu-Liu, Bock and Edmonds algorithm. This algorithm is consistently more effective for finding the roots and the ancestry relationships either considering complete tree scenarios or with missing links in the phylogeny tree. Therefore it presents an interesting feature for forensic purposes in which we are interested such as finding the suspects responsible for breaking copyright laws (roots of the trees) or the chain of suspects involved in an illegal activity (ancestry relationships), among others.

In practice, the Chu-Liu, Bock and Edmonds superiority over Oriented Kruskal algorithm can be directly observed in different applications. For instance, in a forensic scenario in which the task is to trace back the original document of a chain of possible modifications, Chu-Liu, Bock and Edmonds algorithm is more suited since it performs better at finding the root of the near-duplicate's tree. In addition, for copyright enforcement, Chu-Liu, Bock and Edmonds algorithm might be also more interesting to find all the individuals in the tree that effectively changed the document since, for this task, it outperforms Oriented Kruskal when finding the ancestry information. The same conclusion is valid for news tracking services in which it is important to find the connections in the tree and not only its correct root.

Although this paper is about two possible solutions for image phylogeny reconstruction algorithms, we are currently exploring alternatives for the dissimilarity function, and methods to combine multiple dissimilarities. We are also interested in the extension of the current methods to deal with forests (multiple concurrent trees).

Finally, we believe that efforts like the ones presented by [7, 8, 9, 10, 11], and the solutions discussed herein will move the Multimedia Phylogeny approaches toward meeting the strong standards of the Daubert trilogy [29] which establishes a high bar for acceptance of forensic evidence (analog and digital) in US courts and, possibly, in other countries.

Acknowledgment

This work was partially supported by São Paulo Research Foundation – FAPESP (Grant 2010/05647-4), National Counsel of Technological and Scientific Development – CNPq (Grants 307018/2010-5, 304352/2012-8, 306730/2012-0, and 477692/2012-5), Microsoft, and the European Union through the Rewind project. The Rewind project acknowledges the financial support of the Future and Emerging Technologies (FET) program within the Seventh Framework Program for Research of the European Commission (under FET-Open grant 268478).

References

[1] Y. Maret, Efficient Duplicate Detection Based on Image Analysis, Phd thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland (2007).

[2] E. Valle, Local-descriptor matching for image identification systems, Phd thesis, Universit de Cergy-Pontoise, Cergy-Pontoise, France (2008).

[3] E. Valle, M. Cord, S. Philipp-Foliguet, High-dimensional descriptor indexing for large multimedia databases, in: Intl. Conference on Information and Knowledge Management, 2008, pp. 739–748.

[4] A. Joly, O. Buisson, C. Frélicot, Content-Based Copy Retrieval Using Distortion-Based Probabilistic Similarity Search, IEEE Transactions on Multimedia (TMM) 9 (2) (2007) 293–306.
[5] H. sik Kim, H.-W. Chang, J. Lee, D. Lee, BASIL: Effective Near-Duplicate Image Detection using Gene Sequence Alignment, in: European Conference on Information Retrieval (ECIR), Springer, 2010, pp. 229–240.
[6] A. Rocha, W. Scheirer, T. E. Boult, S. Goldenstein, Vision of the unseen: Current trends and challenges in digital image and video forensics, ACM Computing Surveys (CSUR) 43 (4) (2011) 26:1–26:42.
[7] L. Kennedy, S.-F. Chang, Internet Image Archaeology: Automatically Tracing the Manipulation History of Photographs on the Web, in: Intl. Conference on Multimedia, ACM, 2008, pp. 349–358.
[8] A. D. Rosa, F. Uccheddua, A. Costanzo, A. Piva, M. Barni, Exploring Image Dependencies: a New Challenge in Image Forensics, in: Media Forensics and Security II, SPIE, 2010, pp. X1–X12.
[9] Z. Dias, A. Rocha, S. Goldenstein, First Steps Toward Image Phylogeny, in: Intl. Workshop of Information Forensics and Security (WIFS), IEEE, 2010, pp. 1–6.
[10] Z. Dias, A. Rocha, S. Goldenstein, Video phylogeny: Recovering near-duplicate video relationships, in: Intl. Workshop of Information Forensics and Security (WIFS), IEEE, 2011, pp. 1–6.
[11] Z. Dias, A. Rocha, S. Goldenstein, Image phylogeny by minimal spanning trees, IEEE Transactions on Information Forensics and Security (T.IFS) 7 (2) (2012) 774–788.
[12] S. Goldenstein, A. Rocha, High-Profile Forensic Analysis of Images, in: Intl. Conf. on Crime Detection and Prevention, 2009, pp. 1–6.
[13] R. C. Prim, Shortest connection networks and some generalizations, Bell Systems Technical Journal 36 (1957) 1389–1401.
[14] Y. J. Chu, T. H. Liu, On the Shortest Arborescence of a Directed Graph., Science Sinica 14 (1965) 1396–1400.
[15] J. Edmonds, Optimum Branchings., J. Research of the National Bureau of Standards 71B (1967) 233–240.
[16] F. Bock, An algorithm to construct a minimum directed spanning tree in a directed network, Developments in Operations Research (1971) 29–44.
[17] H. Shen, J. Liu, Z. Huang, C.-W. Ngo, W. Wang, Near-duplicate video retrieval: Current research and future trends, ACM Computing Surveys (CSUR)To appear.
[18] C. Xiao, W. Wang, X. Lin, J. X. Yu, G. Wang, Efficient similarity joins for near-duplicate detection, ACM Transactions on Database Systems (TODS) 36 (3) (2011) 15:1–15:41.
[19] J.Kender, M. Hill, A. Natsev, J. Smith, L. Xie, Video genetics: A case study from youtube, in: ACM Intl. Conference on Multimedia (ACMMM), 2010, pp. 1253–1258.
[20] J. Mao, O. Bulan, G. Sharma, S. Datta, Device temporal forensics: An information theoretic approach, in: IEEE Intl. Conference on Image Processing (ICIP), Cairo, Egypt, 2009, pp. 1501–1504.
[21] Z. Fan, R. de Queiroz, Identification of bitmap compression history: Jpeg detection and quantizer estimation, IEEE Transactions on Image Processing (T.IP) 12 (2) (2003) 230–235.
[22] L. Brown, A Survey of Image Registration Techniques, ACM Computing Surveys (CSUR) 24 (1992) 325–376.
[23] J. Pluim, J. Maintz, M. Viergever, Mutual Information Based Registration of Medical Images: A Survey, IEEE Transactions on Medical Imaging (T.MI) 22 (2003) 986–1004.
[24] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding 110 (3) (2008) 346–359.
[25] M. Fischler, R. Bolles, Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography., in: Communications of the ACM, Vol. 24(6), 1981, pp. 381–395.
[26] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. of the American Mathematical Society 7 (1) (1956) 48–50.
[27] R. E. Tarjan, Finding Optimum Branchings, Networks 7 (1977) 309–312.
[28] G. Schaefer, M. Stich, UCID – An Uncompressed Colour Image Database, in: SPIE Storage and Retrieval Methods and Applications for Multimedia, 2004, pp. 472–480, http://www.staff.lboro.ac.uk/~cogs/datasets/UCID/ucid.html.
[29] D. E. Shelton, Forensic Science in Court - Challenges in the 21st Century, Rowman & Littlefield Publishers, 2011.
[30] R. Hartley, A. Zisserman, Multiple View Geomerty in Computer Vision, Cambridge University Press, 2003.

Appendix A. Operation Ranges for the Controlled Experiments

Table A.3 shows the transformations and their operational ranges for the controlled data set used in this paper. All transformations can be combined in any form to create a near-duplicate.

Table A.3: Transformations and their operational ranges for creating the controlled data set.

Transformation	**Operational Range**
Geometry	
(1) Resampling (Up/Down)	
– Global scaling	[90%, 110%]
(2) Rigid Transformation	
– Global scaling	[90%, 110%]
– Rotation	$[-5°, 5°]$
(3) Generic Affine Transformation	
– Scaling by axis	[90%, 110%]
– Rotation	$[-5°, 5°]$
– Off-diagonal correction	[0.95, 1.05]
Cropping	
(4) Cropping	[0%, 5%]
Color	
(5) Brightness Adjustment	[−10%, 10%]
(6) Contrast Adjustment	[−10%, 10%]
(7) Gamma Correction	[0.9, 1.1]
Compression	
(8) Re-compression	[50%, 100%]

Appendix B. Estimating the Homography Matrix

There are numerous algorithms for calculating the homography matrix necessary for mapping pixels of one image to another image's domain. The Direct Linear Transform (DLT) [30] algorithm is a simple algorithm

used to solve for the homography matrix H given a sufficient set of point correspondences.

Assuming we are working in homogeneous coordinates, and that we are given a set of corresponding pixels for two images I_1 and I_2 as illustrated in Figure 2, the relationship between two corresponding pixels **x** and **x'** can be re-written as:

$$c\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = H\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{B.1}$$

where c is any non-zero constant, $(\begin{smallmatrix} u & v & 1 \end{smallmatrix})^T$ represents x', $(\begin{smallmatrix} x & y & 1 \end{smallmatrix})^T$ represents **x** and $H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}$.

Dividing the first and second rows of Eq. B.1 by the third and the third rows, respectively, we end up with the following two equations:

$$-h_1x - h_2y - h_3 + (h_7x + h_8y + h_9)u = 0 \tag{B.2}$$
$$-h_4x - h_5y - h_6 + (h_7x + h_8y + h_9)u = 0. \tag{B.3}$$

We can rewrite Eqs B.2 and B.3 in matrix form as:

$$A_i\mathbf{h} = \mathbf{0} \tag{B.4}$$

where

$$A_i = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & ux & uy & u \\ 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{pmatrix} \tag{B.5}$$

and

$$h = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^T \tag{B.6}$$

Given that each point correspondence in one image provides two equations, we need four correspondences for solving H. The restriction is that no three points can be collinear. Four 2×9 A_i matrices (one per point correspondence) can be stacked on top of one another to get a single 8×9 matrix A.

According to [30], in many cases we may be able to use more than four correspondences to ensure a more robust solution and that is what we do in this paper.