International Conference on Image Processing Theory, Tools and Applications

Author[1], Author[1] and Author[2]
[1] Affiliation 1
e-mail: author-1@ieee.org, author-2@ieee.org
[2] Affiliation 2
e-mail: author-3@ieee.org

***Abstract*— Nowadays it is extremely easy to tamper images and share them thanks to social media. Identifying the tamper history is mandatory to be able to trust any of those images. During this internship, our main goal is to compute the image phylogeny tree of an image set, where images are created from successive JPEG compressions. Every step, a child image is created from the compression of a parent image. In this paper we present our method based on a binary decision between two images and explain the results obtained with this method.**

***Keywords*— Image processing theory, Image processing tools, Image processing applications, Template, Typesetting.**

I. Introduction

II. Method

In this section, we present a new approach to build an image phylogeny tree from a set of near-duplicate images. Unlike state of the art methods [] that mainly focus on complex algorithm that try their best to approximate the tree from a dissimilarity matrix, we try to take a binary decision between two images : "Can this image be an ancestor of that other one?". It allows us to focus more on the images themselves and their characteristics and have a simple tree building algorithm. Figure 1 shows the diagram of our method.

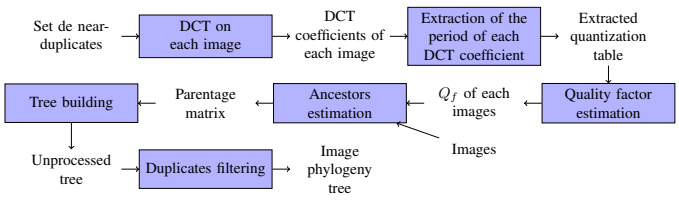


Fig. 1: Diagram of our method

In this section, we will first present the theorems we based designed and based our method on, then we explain how we extract the period of each DCT coefficient to extract a quantization table. We also explain how we compute the quality factor of an image and estimate the ancestors. We then present our tree building algorithm that will eventually output an image phylogeny tree.

A. Theorems

B. Quality factor estimation

We work with images that underwent at least one JPEG compression. These images, traveling through social networks, may have had their format changed, and converted to PNG for instance. The quality factor of the last JPEG compression, mandatory for our method, is then lost. We propose to estimate the quality factor of images that did not undergo any lossless compression other than JPEG.

1) Quantization table extraction: When a signal is quantized, its values gather around multiples of the quantization step. The gap between these values, the period, is what we are going to compute for the first 35 DCT coefficients of the image (in zigzag order). We only use the first 35 because further coefficient are often too quantized and have null values. Not only are they useless, but we also may badly estimate their period, and hinder the table extraction process. To get these periods, we compute the autocorrelation of a signal, as shown Fig. 2. We only keep the first few peaks above a threshold. It allows us to filter out the noise and not have missing peaks because of the threshold, as we would have Fig. 2 with peaks alternating above and below the threshold. We repeat this process for every DCT coefficient selected, it outputs a partial quantization table, named $\widehat{q}(u,v)$. Next we compute Q_f from $\widehat{q}(u,v)$.

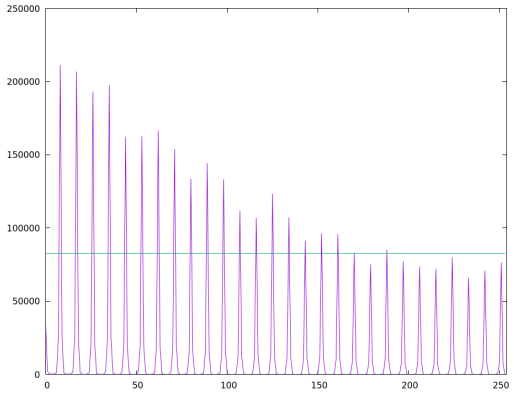


Fig. 2: Autocorrelation of the DC coefficient and the threshold in green

P

2) Quality factor estimation: We have two choices to compute the quality factor from $\widehat{q}(u,v)$. We can either check against every single quantization table from $Q_f = 1$ to $Q_f = 100$ which one is closest or use equations 1 and 2 from the JPEG standard.

$$\begin{aligned} if \quad Q_f < 50 \quad Q_s = 5000/Q_f \\ Q_s = 200 - (Q_f \times 2) \quad otherwise \end{aligned} \tag{1}$$

$$\begin{aligned} q(u,v) = \frac{(base(u,v) \times Q_s) - 50}{100} \\ with\ 1 \leq q(u,v) \leq 255 \end{aligned} \tag{2}$$

The main advantage of checking against every table is that it outputs a Q_f, it is the n the index of the minimal distance between $q_n(u,v)$ and $\widehat{q}(u,v)$, $n \in \{1..100\}$. It however is computation heavy and lacks accuracy.

Using equations 1 and 2, even if it is light and fast, has one main issue : it cannot compute Q_f. It is possible to get Q_s from equation 2, but it is not possible to get Q_f from Q_s in equation 1 without knowing whether Q_s is greater than 50.

We chose to use both methods. We get a first estimation of Q_f by finding which table from $q_n(u,v)$ is most similar to $\widehat{q}(u,v)$, we call this step primary estimation. With this primary estimation of Q_f, we refine our estimation by using equations 1 and 2. Thus, a Q_f* is computed for each coefficient in the partial quantization table $\widehat{q}(u,v)$. The actual Q_f is the mean of all Q_f*. This step is called secondary estimation. We should add that even if after the primary estimation, Q_f is incorrectly estimated above 50, the difference between both cases in equation 1 is small for values of Q_f close to 50. The estimatied Q_f is not always exactly the real Q_f anyway, we address this problem in the parent estimation.

C. Ancestors estimation

The quality factor used to compress the image is mandatory to estimate the ancestors. Not only is it a very effective marker to filter out images that cannot be ancestors, the quality factor also allows us to normalize images to better compare them.

We assume that JPEG compression is a deterministic operation. That is, for the same implementation the same input will always output the same result. This implies that all compressions of the same image will be identical, for any number of compression the image underwent beforehand. As exlained before, we limit ourselves to JPEG compression, it means that to obtain a child image from its parent, the parent was JPEG compressed. The parent is one compression away from its child.

From this, we can accurately estimate the parent :

Let I_m and I_n be two images and Q_{f_m} and Q_{f_n} their quality factors, and $C(I,Q)$ a compression operator, with I an image and Q a quality factor.

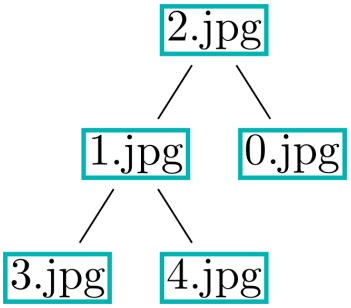
if $I_m = C(I_n, Q_{f_m})$, then I_n is a direct ansestor of I_m.

As we stated above, our Q_f is not always the right Q_f, but can be too big or too small. We solve that testing Q_f and its neighbors as compression candidates until a ancestor is found. The running time of our algorithm is closely related with the accuracy of the estimation of our Q_f's.

This method, even if it yields excellent results, can only find the direct parent and not further ancestors. It is indeed not easy to take a binary decision when the bounds of the values changes so much with the input data.

D. Tree building

The ancestors estimation process output a binary matrix of size $n \times n$, n being the number of images in the set. This matrix is called a parentage matrix. A 1 value at index (i,j) in this matrix means that image I_i is an ancestor of image I_j. A 0 value tells us they are not related. Figure 3 shows an example of a tree with its parentage matrix.



(a) Phylogeny tree

-	I_0	I_1	I_2	I_3	I_4
I_0	-	0	0	0	0
I_1	0	-	0	1	1
I_2	1	1	-	1	1
I_3	0	0	0	-	0
I_4	0	0	0	0	-

(b) Parentage matrix

Fig. 3: A phylogeny tree and its parentage matrix.

From this example, we can notice several things. A column with only 0 values does not have any ancestor, it is the root of the tree, which, by definition, does not have ancestors. A line with only 0 values is the parent of noone : it is a leaf. We can generalize and say that the less a column has 0 values, the less ancestors it has, the closest it is to the root.

Algorithm 1 presents our tree building algorithm.

Each iteration, an image is selected as the root (lines 1 and 11) and named $root$. $root$ is removed (line 7) from the ancestors of the other images. If these other images do not have any ancestor (line 8), it means that $root$ was the direct parent of the image being processed. This image is added as a child of $root$ (line 9). Line 5 prevents images from being a potential root twice.

Data: M a n×n parentage matrix
Result: the root of the tree

```
nextRoot ← row with min sum of elements;
treeRoot ← nextRoot;
forall rows row of M do
    root ← nextRoot;
    mark root as done;
    for i ← 0 to n do
        row[i] ← 0;
        if sum of elements of row == 0 then
            add i as child of root;
        end
        if row has the smallest sum of elements and is
         not marked as done then
            nextRoot ← i;
        end
    end
end
return treeRoot
```

Algorithm 1: Tree building algorithm.

This algorithm runs in $O(n^2)$. It has two nested loops and if the sums are computed once at the beginning and updated every time a parent is removed, there is no extra loop increasing complexity.

III. Experiments and results

In this section, we presents the results we obtained with our method. We start by describing how we generated our datasets and then explain our results and how and why they were obtained.

A. Datasets generation

We created three datasets, one with trees with 15 images, another one with 25 images and a last one with 50 images. It will allows us to see how things like the distance to the root or the number of compressions affect our tree estimation method.

From a seed images (never lossy compressed), a first image is compressed with $85 < Q_f < 99$, it is the root image. This image is added to the images pool. While the number of images in the pool is smaller than the desired number, an image is randomly chosen in the pool, compressed with $Q_{f_{parent}} - 15 < Q_{f_{child}} < Q_{f_{parent}} - 1$ and added to the pool. The quality factor cannot be smaller than 30, further than that, the image is both too deteriorated and does not contain meaningful information and not a realistic use case on social networks.

We used the six base images of BOWS 2 [1], where each image was used to create 15 trees for each tree size, a total of 90 trees per dataset, or 8100 images.

B. Experimental results

We use seven metrics to measure our results, three metrics measuring the accuracy of the estimation of the quality factor, the four other are given by Dias et al. [2] and available table III-B.

Root	$R(IPT_1, IPT_2) = \begin{cases} 1 & if \texttt{Root}(\texttt{IPT}_1) = \texttt{Root}(\texttt{IPT}_2) \\ 0 & Otherwise \end{cases}$
Edges	$E(IPT_1, IPT_2) = \frac{\lvert E_1 \cap E_2 \rvert}{n-1}$
Leaves	$L(IPT_1, IPT_2) = \frac{\lvert L_1 \cap L_2 \rvert}{\lvert L_1 \cup L_2 \rvert}$
Ancestry	$A(IPT_1, IPT_2) = \frac{\lvert A_1 \cap A_2 \rvert}{\lvert A_1 \cup A_2 \rvert}$

$Root$ is trivial and returns 1 if the roots of both trees are identical, $Edge$ measures the ratio of nodes which have the correct direct parent, $Leaves$ is the ratio of correct leaves and $Ancestry$ is the ratio of correct ancestors up to the root.

The three metrics used to measure the accuracy of the estimation of the quality factor is the estimation mean error, the mean error on all the Q_f's on every image of the dataset. The mean overestimation and mean underestimation, which should be symetrical, will tell us whether our algorithm overestimates the Q_f's, and when.

Table 4 shows the results we got using our methods on the three datasets.

Metric \ Dataset	**15 images**	**25 images**	**50 images**
Mean estimation error of Q_f	0.52	0.64	0.83
Mean overestimation of Q_f	1.61	1.86	1.94
Mean underestimation of Q_f	1.07	2.12	6.68
roots	95.83	88.88	84.72
edges	99.70	99.24	98.97
leaves	99.59	99.15	98.74
ancestry	99.44	96.88	96.96

Fig. 4: Results table, the first three metrics measure the accuracy of the estimation of Q_f, the other which measure the quality of the tree reconstruction are given in percentage.

We can see that the smaller the dataset, the better the results. We should add that the best score for the three first metrics is 0, when all Q_f were correctly estimated, and the best score for the other is 100, when the two trees are identical.

The mean errors, even if they are below 1 for the three datasets, and are very good, get worse when the number of images increases, in particular the overestimation is pretty high for 50 images.

So high an overestimation is due to too much noise on the DCT coefficients for images compressed a lot of times. We cannot properly estimated the period of such coefficients and therefore cannot properly estimate Q_f. When no peak is

Metric \ Dataset	15 images	25 images	50 images
roots	69.60	33.33	49.01
edges	88.86	92.40	95.07
leaves	91.30	93.00	94.72
ancestry	78.06	82.22	88.21

Fig. 5: Results with one image missing in every tree.

detected, our algorithm actually assumes that the image was not quantized, or compressed with a Q_f very close to 100, since a signal not quantized will not show any peaks. A high number of compression is most likely to happen for 50 images, hence the worse results.

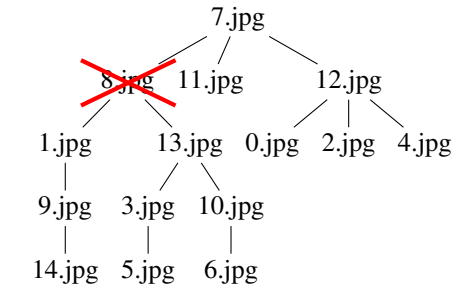
We assumed that JPEG compression was a deterministic operation. However, it appears not to be the case for small Q_f's (¡ 35) with a high number of compressions. It means that for some images, no parent can be found. We wanted to have a simple tree reconstruction algorithm, with only binary data, no other information is available during the reconstruction process, thus, our algorithm cannot distinguish the root, which has no parent, and a problematic image, with no parents detected. The root is then chosen as the image with the smallest index. Again, this is most likely to occur on big datasets.

An other problems, which does not affect the root, but rather the rest of the tree, is block convergence, and image convergence, as shown by Lai and Bohme [3]. After a given number of compressions, an image can become identical to its parent, it then becomes impossible to tell which is which. This is caused by images with $Q_f = 30$, the lower bound of Q_f during our generation. If an image with $Q_f = 30$ has a child, this child will also have $Q_f = 30$, and so recursively.

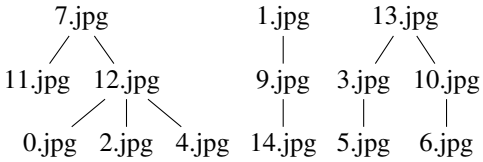
The results given above are obtained on the whole dataset, where all images are available. Our method is very effective to estimate the phylogeny tree when all images are present, when there is always a direct parent. Since we cannot estimate other ancestors, we wanted to try our methods on the same datatets, with one image missing to see if the results were still good. Table 5 shows the results.

We can see that the root metric suffers the most from the image missing. As explained above, our tree reconstruction algorithm cannot differentiate an image with no parents (the root) and an image with no parents detected. The root of the tree is in this case almost randomly chosen. Figure 6 shows how a tree with a missing node results in an incorrect tree estimation. We actually have correct subtrees, with the right parents, but we do not have the information on how to links those tree together.

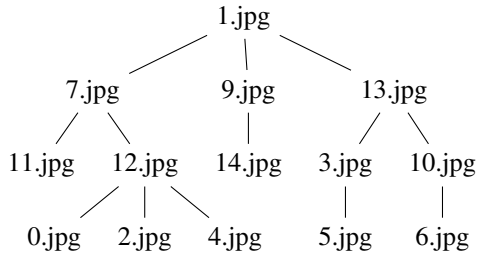
Even though our method was designed for grayscale images, we wanted to see if the luminance channel of a color image contained enough information for our method to estimate the tree. None of our algorithm were modified and we only gave color images instead of grayscale images as input. Table 7



(a) Original tree with node 8 removed.



(b) Three subtrees that we do not know how to link.



(c) Output of tree reconstruction algorithm

Fig. 6: Exemple of a tree with a missing node and a reconstruction attempt.

shows our results.

Métrique \ Dataset	15 images	25 images	50 images
Mean estimation error of Q_f	1.15	1.29	1.42
Mean overestimation of Q_f	1.85	2.70	2.64
Mean underestimation of Q_f	2.97	3.79	4.94
roots	93.94	81.82	87.88
edges	99.35	98.61	99.38
leaves	99.62	98.66	99.78
ancestry	98.79	94.13	98.82

Fig. 7: Results table with color images.

The results are not very different from grayscale images, slightly worse, but not significantly so. The estimation of Q_f is not as good, it can be because of the rounding when going from RGB to YUV. We did not use any of the chrominance channel, the luminance channel looks to contain enough information and allows us to reliably estimate the tree.