

# La phylogénie des images dans les réseaux sociaux

---



## Mémoire de fin d'étude

Master *Sciences et Technologies*,  
Mention *Informatique*,  
Parcours IMAGINA

**Auteur**

Noé Le Philippe

**Superviseurs**

William Puech  
Christophe Fiorio

**Lieu de stage**

Équipe ICAR - LIRMM UMR 5506 - CNRS, Université de Montpellier

## Résumé

Il est aujourd’hui extrêmement facile de manipuler des images et de les partager à l'aide des réseaux sociaux. Il est important d'identifier l'historique des transformations appliquées aux images afin de leur accorder un minimum crédibilité. Dans le cadre de ce stage, l'objectif principal de nos recherches consiste à calculer l'arbre de phylogénie d'un ensemble d'images où les images sont issues de différentes compressions JPEG successives. À chaque étape, une image enfant est donc créée à partir de la compression d'une image parent. Dans ce rapport nous présentons dans l'état de l'art les différentes méthodes de reconstruction d'un arbre phylogénétique et traitons de la détection et de l'identification des compressions JPEG multiples. Nous présentons également notre méthode qui s'articule autour d'un point central : la décision binaire entre deux images et nous détaillons les résultats obtenus grâce à cette méthode.

**Mots clés.** Phylogénie des images, JPEG, Compressions multiples, Arbre de phylogénie

---

## 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 master's thesis we present in a state of the art several methods for computing the image phylogeny tree and take an overview of the different methods for detecting and identifying multiple JPEG compressions. We also introduce our method based on a binary decision between two images and explain the results obtained with this method.

**Mots clés.** Image phylogeny, JPEG, Multiple compressions, Phylogeny tree

# Table des matières

<b>Table des matières</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Near-duplicate images (NDI) . . . . .	2
1.2 Arbre phylogénétique (Image Phylogeny Tree - IPT) . . . . .	2
1.3 Pourquoi se restreindre à la compression comme transformation? . . . . .	4
<b>2 État de l'art</b>	<b>5</b>
2.1 Étude de l'arbre phylogénétique . . . . .	5
2.2 Analyse des recompressions JPEG . . . . .	8
2.3 Distances entre distributions . . . . .	14
<b>3 Méthode de calcul de phylogénie à l'aide des caractéristiques des coefficients DCT d'une bitmap</b>	<b>17</b>
3.1 Principe et théorème . . . . .	17
3.2 Estimation du facteur de qualité d'une image JPEG . . . . .	18
3.3 Estimation du parent . . . . .	21
3.4 Reconstruction de l'arbre . . . . .	22
3.5 Filtrage des doublons . . . . .	23
<b>4 Résultats</b>	<b>25</b>
4.1 Génération des données . . . . .	25
4.2 Résultats expérimentaux . . . . .	25
<b>5 Perspectives et conclusion</b>	<b>32</b>
<b>Bibliographie</b>	<b>33</b>

## Introduction

La phylogénie, en sciences naturelles, est définie comme l'étude des relations de parenté entre êtres vivants. Et c'est exactement de cela qu'il s'agit dans le cas des images, l'étude des relations de parenté entre images. [18]

À l'ère du numérique et des réseaux sociaux, il n'a jamais été aussi simple de partager des idées et du contenu. À chaque partage cependant, l'information peut être modifiée plus ou moins fortement. Les images, puisque c'est là notre sujet d'étude, peuvent avoir subi un certain nombre de transformations et de modifications avant d'être publiées sur les réseaux sociaux. C'est dans ce contexte que nous allons intervenir afin de tenter de reconstituer la phylogénie d'une image. Il peut être difficile de différentier une image modifiée de l'originale, et de savoir laquelle est l'originale alors que cette détection est cruciale, surtout dans un monde où l'information peut être facilement falsifiée par tout un chacun. Les applications sont multiples et variées, et ne se cantonnent pas à la détection et la discrimination d'images altérées. Il est également possible de se servir de la phylogénie des images pour optimiser l'espace de stockage en ne gardant que l'image originale et l'historique des modifications ou encore suivre la diffusion et l'évolution des idées sur les réseaux sociaux.

Dans ce chapitre nous commencerons par définir ce que sont les near-duplicate images, puis nous détaillerons la notion d'arbre phylogénétique et enfin nous expliquerons pourquoi nous nous sommes restreints à la compression JPEG dans le cadre de cette étude. Nous ferons un état de l'art dans le chapitre 2 où nous présenterons les différentes approches pour calculer l'arbre de phylogénie. Nous aborderons également la problématique de la détection des recompressions et l'estimation du nombre de recompressions. Nous présentons ensuite notre méthode dans le chapitre 3, où détaillerons en quoi consiste, et comment est prise une décision binaire entre deux images, et comment l'arbre est ensuite reconstruit. Les résultats obtenus grâce à cette méthode seront ensuite présentés et expliqués chapitre 4 puis nous finirons par conclure chapitre 5.

## 1.1 Near-duplicate images (NDI)

Nous travaillons sur un ensemble d'images, toutes similaires visuellement, et au milieu de cet ensemble d'images, nous devons décider quelle image est le parent de quelle autre, ou autrement dit, quelles images sont des **near-duplicates**. JOLY et al. [13] définissent la notion de near-duplicate comme suit :  $I_{n+1} = T(I_n)$ ,  $T \in \mathcal{T}$  où  $I_n$  est l'image parent,  $I_{n+1}$  est l'image à la génération suivante  $n+1$ , l'image enfant et  $\mathcal{T}$  est un ensemble de transformations autorisées,  $I_{n+1}$  et  $I_n$  sont alors des NDI. Dans le cas général,  $\mathcal{T} = \{resampling, cropping, affine warping, color changing, lossy compression\}$ . La figure 1.1 montre un exemple de near-duplicates. Le *resampling*, ou rééchantillonage en français revient à changer le nombre de pixels de l'image, le *crop* est illustré figure 1.1b, *affine warping* englobe tout ce qui peut être translation ou rotation, *color changing* concerne tout ce qui va changer la couleur, comme le changement de contraste par exemple, ou le passage au noir et blanc (figure 1.1a), et enfin *lossy compression* est la compression avec pertes, une dégradation de l'image pour réduire son poids. Notons l'utilisation du terme *transformations autorisées*. Ce terme est double, d'une part, il place une limite arbitraire dans la force de la transformation, par exemple une image croppée à plus de 10% pourra ne pas être considérée comme un near-duplicate, et d'autre part, il permet de restreindre l'espace des transformations possibles. Ces transformations peuvent évidemment se composer, et une image enfant peut être le résultat de plusieurs transformations.

**Note.** Nous utiliserons les notions *relation parent-enfant* et *relation de parenté de manière interchangeable* dans le reste de ce rapport, mais c'est bien d'une relation parent-enfant qu'il s'agit.



FIGURE 1.1 – Exemple de near-duplicates

## 1.2 Arbre phylogénétique (Image Phylogeny Tree - IPT)

L'arbre phylogénétique (IPT) est l'arbre représentant les relations de parenté entre les différentes images. Il est extrait d'un ensemble de NDI, et constitue l'objectif final de l'application. La figure 1.2 illustre la construction de l'IPT à partir d'un ensemble de NDI. Le passage d'une génération à l'autre, autrement dit d'un noeud à son fils, se fait à travers la transformation  $I_{n+1} = T(I_n)$ , ainsi, une image  $I_{n+1}$  et son parent  $I_n$  sont des NDI, alors qu'une image  $I_{n+1,i}$  et sa soeur  $I_{n+1,j}$ ,  $i \neq j$  ne le sont pas (figure 1.3).

La reconstruction de l'arbre se concentre autour de deux problèmes principaux. Le premier est l'identification de la racine ( $n = 0$ ), et le second est l'estimation du reste de l'arbre, et en

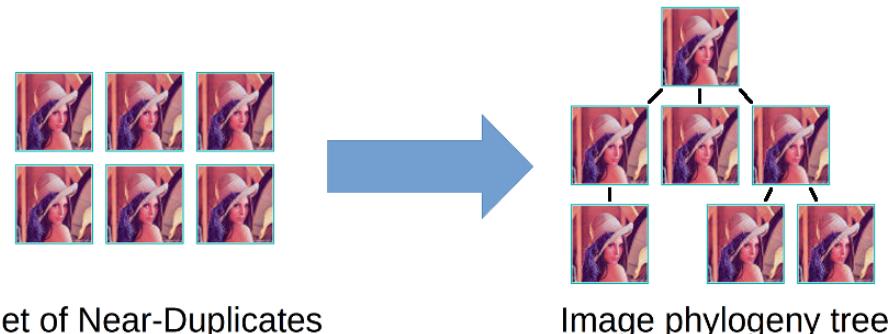


FIGURE 1.2 – Passage d'un ensemble de NDI à un arbre phylogénétique

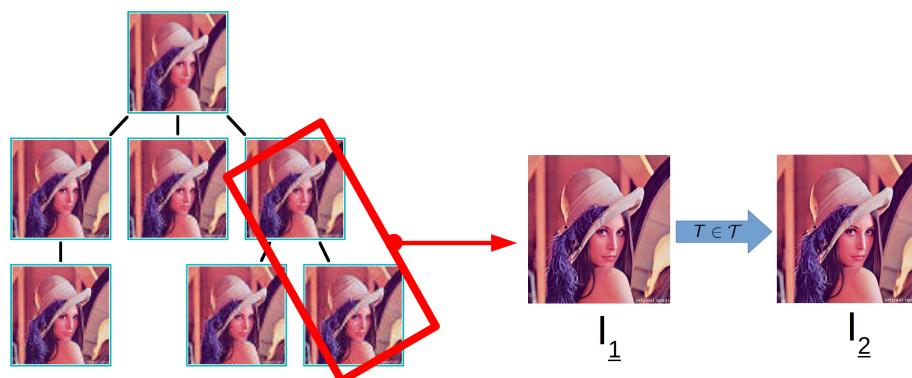


FIGURE 1.3 – Passage d'une image parent à l'image enfant

particulier positionner précisément chacune des images dans leur génération respective (valeur de  $n$ ). Il est en effet critique d'identifier correctement la racine. Prenons par exemple un des cas d'utilisation de l'IPT, la détection d'altération d'images. L'idée est que pour un ensemble d'images, plus la génération est proche de la racine, c'est à dire plus  $n$  est petit, moins l'image a subi de transformations, et donc moins elle est altérée. Avec comme cas particulier  $n = 0$ , l'image est alors la génération 0 : la racine. Notons que si la racine est mal identifiée, une image pourra être à tort marquée comme altérée. Il n'est pas toujours garanti que la totalité des images de l'arbre original soit présente, l'identification précise de la génération peut alors se révéler complexe. Certaines transformations peuvent être mineures, et difficile à détecter, dans ce cas deux images de deux générations proches  $n$  et  $n + k$ , avec  $k$  petit ( $k = 1, 2$  ou  $3$  par exemple), peuvent être identifiées à tort comme appartenant à la même génération. En conclusion, il est clair que l'arbre ne sera qu'une estimation, et sauf dans des cas idéaux, ne sera pas l'arbre original.

### 1.3 Pourquoi se restreindre à la compression comme transformation ?

Dans le cadre du stage, nous proposons de réduire l'espace des transformations à  $\mathcal{T} = \{\text{lossy compression}\}$  avec JPEG comme algorithme de compression. L'étude et la détection des recompressions JPEG est en effet un sujet suffisamment vaste et complexe pour que cela soit la seule transformation étudiée dans le cadre de ce stage de recherche. C'est de plus un sujet qui à notre connaissance n'a pas été traité dans le cadre de la phylogénie des images, et qui est largement étudié dans le domaine du forensics (criminalistique en français). L'étude et la détection des recompressions est en effet un sujet vaste, la figure 1.4 détaille l'étendue du problème. La compression JPEG sera détaillée dans le chapitre 2 en section 2.2. Aussi la seule caractéristique dont nous parlerons dans cette section est le facteur de qualité  $Q$ .  $Q \in [1..100]$  et plus  $Q$  est grand, plus l'image sera de bonne qualité, mais moins elle sera compressée. Nous sommes donc en présence de trois cas pour les recompressions successives : le premier est le cas où tous les  $Q$  successifs sont égaux, dans le second cas, l'image est plus dégradée à chaque recompression, et dans le troisième, nous n'avons aucune idée sur la manière dont l'image a été recompressée par rapport à son parent. Elle peut avoir été recompressée avec la même qualité, ou une meilleure qualité, ou encore une moins bonne qualité.

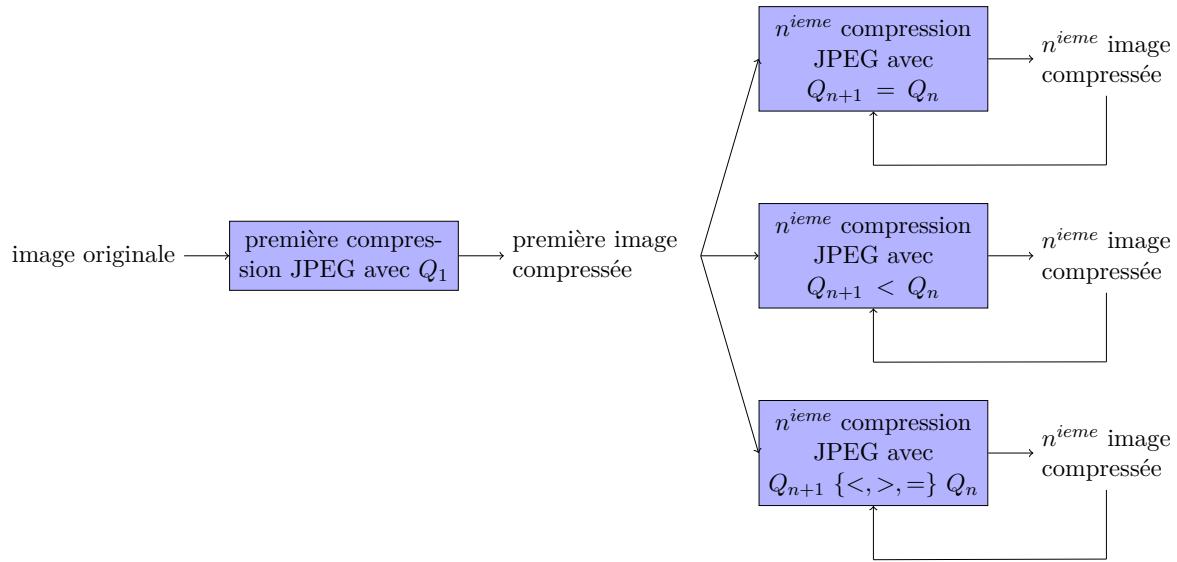


FIGURE 1.4 – Exemple des différents scénarios de recompression d'images

Nous avons choisi de nous concentrer particulièrement sur le deuxième cas, lorsque le facteur de qualité du parent est strictement supérieur à celui de l'enfant.

## État de l'art

Ce chapitre tentera de faire un état de l'art des différentes techniques de calcul d'un arbre phylogénétique, puis traitera des différentes méthodes permettant d'extraire des informations de plusieurs compressions successives, et enfin présentera quelques calculs de distances.

### 2.1 Étude de l'arbre phylogénétique

#### La Visual Migration Map (VMM)

L'article de KENNEDY et CHANG est à notre connaissance le premier article concernant vraiment notre sujet [15]. Les auteurs proposent une approche permettant de détecter automatiquement la manière dont une image a été éditée ou manipulée, et d'en extraire des relations parent-enfant entre les images. Ils vont construire à partir de l'estimation de ces transformations une Visual Migration Map (VMM) (voir figure 2.1) qui est en fait un arbre de phylogénie.

KENNEDY et CHANG partent du principe que les transformations sont directionnelles, c'est à dire qu'il n'est possible d'aller que d'une image moins transformée vers une image plus transformée. Ainsi, ils vont tenter d'estimer la direction de chaque transformation entre deux images  $I_1$  et  $I_2$  ( sachant que  $\mathcal{T} = \{scaling, cropping, grayscale, overlay, insertion\}$ ). Trois scénarios sont alors possibles : si toutes les transformations sont dans le même sens, l'image fille est alors celle vers qui pointent les transformations, si les transformations sont dans des sens contraires, les images sont sûrement des soeurs, elles n'ont en tous cas pas de relation parent-enfant, et enfin si aucune transformation n'a été détectée, c'est que soit les images sont identiques, soit qu'elles ne sont pas des near-duplicates. La figure 2.2 illustre ce principe par un exemple.

Un graphe est ensuite construit à partir des couples d'images pour lesquels une relation parent-enfant a été détectée. À noter qu'une relation parent-enfant ne veut pas forcément dire que c'est le parent direct mais plutôt un ancêtre. Ainsi, un noeud du graphe (une image) peut avoir plusieurs noeuds parents, pour finalement obtenir l'arbre désiré, seuls les chemins les plus longs sont conservés, comme on peut le voir figure 2.3.

#### Image Phylogeny Tree (IPT)

Tout comme l'approche présentée précédemment, DIAS et al. [8][9] proposent une approche basée sur le contenu de l'image. Elle consiste à mapper une image sur le domaine d'une autre, pour

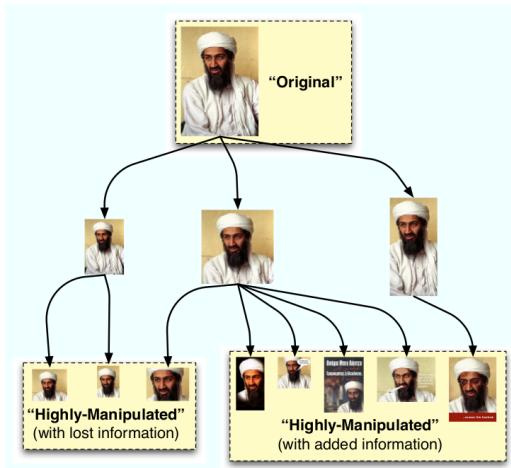


FIGURE 2.1 – Exemple de VMM, issu de [15].

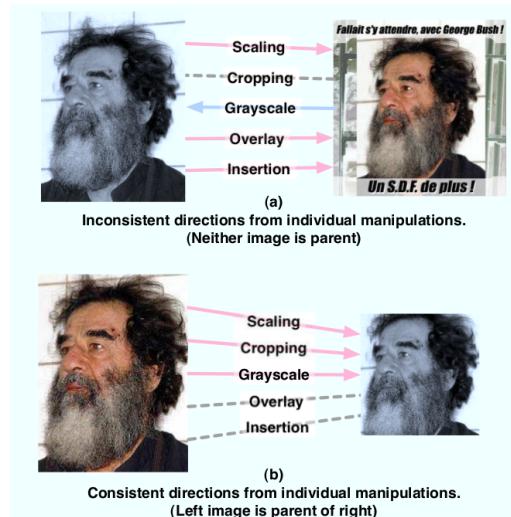


FIGURE 2.2 – Exemple de direction des transformations, issu de [15].

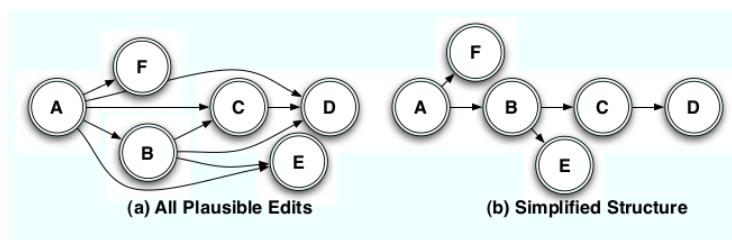


FIGURE 2.3 – Exemple de simplification de graphe, issu de [15].

pouvoir les comparer, et ensuite estimer le coût de cette opération. L'hypothèse est que si deux images sont dépendantes, alors il est possible d'obtenir l'une en appliquant une transformation à l'autre.

Les images sont comparées à l'aide d'une fonction de dissimilarité, ou *dissimilarity function*  $d(.,.)$  qui renvoie des petites valeurs lorsque les images sont proches (elles ont subi des transformations similaires). L'équation 2.1 détaille cette fonction.  $I_m$  et  $I_n$  sont les deux images qui vont être comparées, et  $T_{\vec{\beta}} \in \mathcal{T}$  est la transformation en train d'être estimée, où  $\vec{\beta}$  est l'ensemble des valeurs que peut prendre en paramètre la transformation. Dans le cas de la compression JPEG par exemple  $\vec{\beta} = \{1..100\}$ . La transformation la plus faible est gardée comme résultat, c'est la transformation la plus probable qu'a pu subir l'image. À noter que  $d(.,.)$  est asymétrique, c'est à dire que  $d(I_m, I_n) \neq d(I_n, I_m)$ , ce qui est parfaitement logique, en plus d'être nécessaire, sachant que les transformation sont directionnelles, comme expliqué précédemment.

$$d(I_m, I_n) = \min_{T_{\vec{\beta}}} |I_n - T_{\vec{\beta}}(I_m)|_{\text{comparison method}} \quad (2.1)$$

Nous voyons donc bien que la problématique principale de cette méthode est de trouver une bonne méthode de comparaison. Les auteurs procèdent de la manière suivante :

1. Trouver des points caractéristiques (SURF [1]),
2. Filtrer les points et estimer les paramètres de transformation affines tels que les translations, rotations et rééchantillonages avec RANSAC [11],
3. Calculer la moyenne et la variance de chaque canal couleur de  $I_n$  pour normaliser les couleurs de  $I_m$ ,
4. Compresser les résultats des deux étapes précédentes avec la table de quantification de  $I_n$

La dissimilarité entre les deux images est enfin obtenue en utilisant la *minimum mean squared error* (MMSE) entre les deux images dans le domaine spatial. (*comparison method* de l'équation 2.1)

Ces quatre étapes servent à rendre les images comparables, en mappant l'une dans le domaine de l'autre, pour avoir des résultats pertinents.

La distance  $d(.,.)$  est donc appliquée à tous les couples d'images de l'ensemble, pour créer une matrice de dissimilarité, ou *dissimilarity matrix*, une matrice de taille  $n \times n$  qui sera ensuite donnée comme entrée à leur algorithme de reconstruction d'arbre, Oriented Kruskal. Cet algorithme est expliqué de manière exhaustive dans [9]. Dans le chapitre 3, nous proposons une approche différente, notre reconstruction sera donc également différente.

En plus de proposer une approche pour reconstituer l'arbre de phylogénie, DIAS et al. proposent une approche pour comparer deux arbres, et donc évaluer notre arbre reconstruit s'il est comparé

avec la vérité terrain. Il consiste en quatre métriques :

$$\text{Root} \quad R(IPT_1, IPT_2) = \begin{cases} 1 & \text{if } \text{Root}(IPT_1) = \text{Root}(IPT_2) \\ 0 & \text{Otherwise} \end{cases}$$

$$\text{Edges} \quad E(IPT_1, IPT_2) = \frac{|E_1 \cap E_2|}{n-1}$$

$$\text{Leaves} \quad L(IPT_1, IPT_2) = \frac{|L_1 \cap L_2|}{|L_1 \cup L_2|}$$

$$\text{Ancestry} \quad A(IPT_1, IPT_2) = \frac{|A_1 \cap A_2|}{|A_1 \cup A_2|}$$

**Root** est triviale, et renvoie si les racines sont identiques. **Edges** mesure le ratio de noeuds ayant le bon parent, **Leaves** est le ratio de feuilles correctes, et enfin **Ancestry** est le ratio d'ancêtres corrects jusqu'à la racine.

Ces métriques serviront à évaluer nos résultats dans la suite du stage.

## 2.2 Analyse des recompressions JPEG

Notre but n'est pas vraiment de détecter si une image a été compressée plusieurs fois, nous sommes en effet quasi-certains que nos images auront été recompressées, puisque nous travaillons avec un ensemble de NDI où  $\mathcal{T} = \{\text{lossy compression}\}$ . L'important est plutôt de savoir combien de fois, et à partir de quelle image  $I_n$  a été compressée, et donc pouvoir en déduire sa distance avec la racine. La majorité des articles dans le domaine du forensics ne se concentrent que sur une image, pour en extraire le maximum d'informations possible. Ce n'est pas exactement notre cas, puisque les informations pertinentes pour nous ne sont pas dans l'image directement, mais plutôt dans ses relations avec les autres. Il nous a cependant paru important de traiter l'aspect forensics du problème, et se renseigner sur les différentes techniques, qui bien que créées pour un autre cas d'utilisation, peuvent, sinon s'adapter, au moins nous donner des pistes de recherche.

### La compression JPEG

Une rapide introduction sur la compression JPEG et son fonctionnement s'impose. Nous ne parlerons cependant que du mode de compression avec perte, en laissant de coté son mode de compression sans perte, peu intéressant en plus de n'être que rarement utilisé. Pour de plus amples détails, le lecteur se dirigera vers [19].

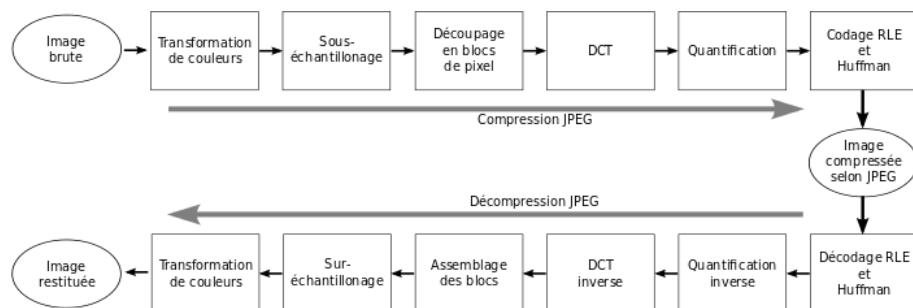


FIGURE 2.4 – Étapes de compression et décompression, issues de [14].

La figure 2.4 liste toutes les étapes permettant de passer d'une image non compressée à une image compressée ainsi que le processus inverse. Les étapes sont sommairement :

- L'image est convertie dans l'espace YUV,
- Les canaux U et V sont sous-échantillonés,
- Chaque canal est découpé en blocs de  $8 \times 8$  pixels,
- Une DCT est appliquée à chacun de ces blocs pour avoir une matrice de  $8 \times 8$  coefficients,
- Chaque coefficient est quantifié selon une table de quantification (voir figure 2.5) correspondant au facteur de qualité  $Q \in \{1, 2, 3, \dots, 100\}$  et arrondi à l'entier le plus proche,
- Le tout est ensuite compressé à l'aide d'un codage entropique

C'est surtout l'étape de quantification qui va réduire la quantité d'information, et donc permettre de réduire la taille du fichier. Cette quantification, si elle est trop agressive va faire apparaître des artefacts de bloc, des blocs  $8 \times 8$  visibles (voir figure 2.6). C'est ce qui caractérise le JPEG, et qui permet de détecter un certain nombres de choses, comme l'altération d'images [3], les doubles compressions [2] ou encore dans notre cas les relations de parenté.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

FIGURE 2.5 – Exemple de table de quantification, issu de [14].

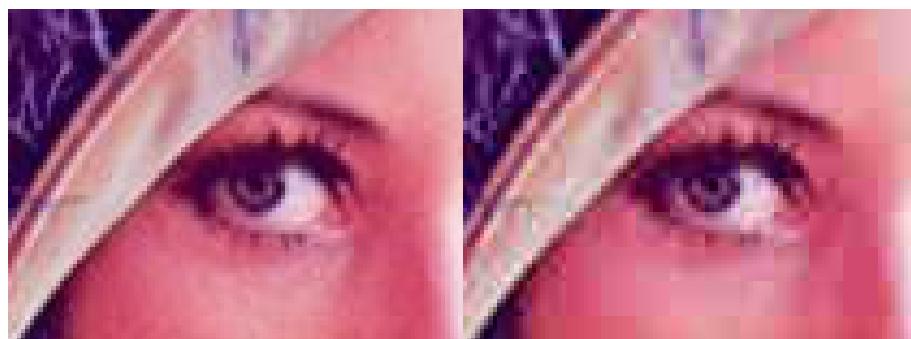


FIGURE 2.6 – Artefacts de blocs, à gauche avec  $Q=90$ , à droite avec  $Q=20$

## Estimation de la matrice de compression primaire dans les images JPEG doublement compressées

LUKÁŠ et FRIDRICH [17] proposent dans leur approche non seulement de détecter si une image est doublement compressée, mais également d'estimer les paramètres de la première compression. Pour bien comprendre la suite de leurs travaux, il convient d'expliquer plus en détail l'étape de quantification de JPEG.

La quantification se fait de la manière suivante :

$$F^*(u, v) = \left\lfloor \frac{F(u, v) + \left\lfloor \frac{Q(u, v)}{2} \right\rfloor}{Q(u, v)} \right\rfloor, \quad (2.2)$$

où  $F^*(u, v)$  est la valeur quantifiée,  $F(u, v)$  est la valeur aux indices  $u$  et  $v$  dans la matrice  $8 \times 8$  de coefficients DCT et  $Q(u, v)$  est la valeur dans la table de quantification (figure 2.5) à ces mêmes indices.

Le calcul inverse est :

$$F'(u, v) = F^*(u, v) * Q(u, v). \quad (2.3)$$

où  $F'(u, v)$  est la valeur issue de la quantification inverse aux indices  $u$  et  $v$  dans la matrice  $8 \times 8$  de coefficients DCT,  $F^*(u, v)$  est la valeur quantifiée et  $Q(u, v)$  est la valeur dans la table de quantification (figure 2.5) à ces mêmes indices.

Nous voyons bien à partir des équations 2.2 et 2.3 que le résultat après décompression sera un multiple de  $Q(u, v)$  et que toutes les valeurs n'étant pas des multiples seront absentes.

La double compression est le fait de compresser deux fois une image, et donc de lui faire subir deux fois toutes les étapes de compression et de décompression, avec tous les arrondis et troncage que cela peut comporter. On notera, pour la double compression,  $Q_1$  comme la matrice primaire, c'est à dire la table de quantification ayant servi à faire la première compression (inconnue donc), et  $Q_2$  comme la matrice secondaire, ou la table de quantification actuelle, disponible dans l'en-tête du fichier JPEG. Les auteurs limitent la double compression aux cas où  $Q_1 \neq Q_2$ .

Lors de la première compression, les coefficients sont quantifiés avec  $Q_1$ , ce qui veut dire que les coefficients sont des multiples de  $Q_1$ . Cependant, lorsque l'image est repassée du domaine fréquentiel au domaine spatial grâce à la DCT inverse, un certain nombre d'arrondis et de troncages se produisent pour rester dans l'intervalle  $[0..255]$ . Lors de la seconde compression, les coefficients DCT sont calculés à partir de ces valeurs inexactes, et donc ne seront pas multiples de  $Q_1$ , mais se concentreront autour. Ces nouveaux coefficients DCT seront ensuite quantifiés pour former la nouvelle image, doublement compressée.

Les auteurs ayant laissé de côté le cas où  $Q_1 = Q_2$ , nous sommes en présence de deux cas :  $Q_1 > Q_2$  et  $Q_1 < Q_2$ . Chacun des cas a des artefacts distincts et reconnaissables. La figure 2.7 illustre un exemple de distributions ayant subi des quantifications, la figure 2.7a n'a subi qu'une seule quantification, la figure 2.7b correspond au cas où  $Q_1 > Q_2$  et la figure 2.7c correspond au deuxième cas. Un autre cas est délaissé, c'est celui où  $Q_1$  est facteur  $Q_2$ , ce qui est assez logique, ils auront les mêmes multiples, et il sera donc impossible de détecter quoi que ce soit, du moins en utilisant la méthode proposée ici.

L'estimation de la première table de quantification se limite aux basses fréquences, les hautes fréquences étant plus fortement quantifiées (souvent jusqu'à 0), elles contiennent moins d'informations et donc rendent leur estimation difficile.

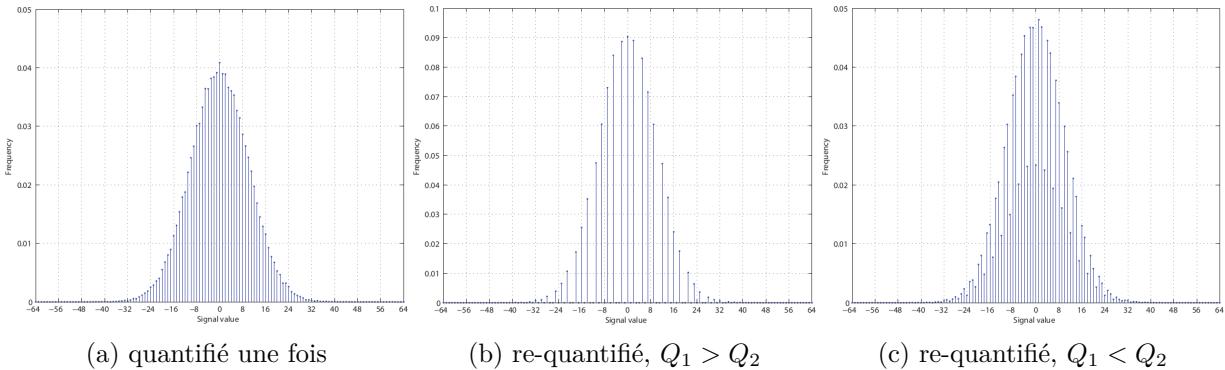


FIGURE 2.7 – Exemple de quantifications sur l’histogramme d’une distribution normale avec des classes de largeur 1, issu de [10].

Leur technique consiste à se munir d’une série de candidats pour la matrice primaire, notés  $\{Q_{1,1}, \dots, Q_{1,n}\}$ . Ces tables de quantification viennent des tables utilisées dans les appareils photos, les téléphones portables ou les différentes implémentations de la compression JPEG. Parmi ces  $\{Q_{1,1}, \dots, Q_{1,n}\}$ , le meilleur candidat, et donc la table la plus probable sera la table de quantification pour laquelle la différence entre l’histogramme de l’image et l’histogramme de l’image compressée avec  $Q_{1,k}$  sera la plus faible.

L’identification de la matrice primaire peut être utile pour l’identification d’une relation parent-enfant entre deux images. Une image est potentiellement le parent d’une autre si sa table de quantification est la même que la matrice primaire d’une image recompressée avec une matrice secondaire. Cette méthode, bien que fort intéressante dans le cas de la phylogénie, perd en efficacité au-delà de la première recompression, et l’identification des différentes matrices peut devenir difficile à cause du bruit engendré par les multiples recompressions.

### Détection des doubles compressions JPEG avec la même matrice de quantification

Comme expliqué section 1.2, nous tentons dans un premier temps d’identifier la génération  $n$  d’une image  $I_n$ . Une tâche qui peut s’avérer difficile si les transformations sont minimes. En sachant que deux images  $I_m$  et  $I_n$  sont extrêmement similaires, sommes-nous en présence d’une double compression avec la même matrice de quantification, où l’une est le parent de l’autre, ou sommes-nous dans le cas où les images sont soeurs, et générées à partir d’une table de quantification identique ?

L’article de HUANG et al. [12] traite précisément de ce problème. Les auteurs constatent que les différences entre deux images issues de compressions successives diminuent lorsque le nombre de compressions augmente (voir figure 2.8), ce qui est en accord avec l’article de LAI et BOHME [16]. Leur méthode, même si elle permet de détecter jusqu’à trois ou quatre compressions, est la plus efficace lorsque qu’il n’y a eu qu’une seule recompression.

Les auteurs vont s’appuyer sur trois signes distinctifs laissés par la compression JPEG : les erreurs de quantification, les erreurs de troncage et les erreurs d’arrondis. Leur méthode de détection est basée sur la perturbation aléatoire des coefficients DCT de l’image.

À partir d’une image  $I$  compressée un nombre inconnu de fois, ils vont calculer  $D$  comme le

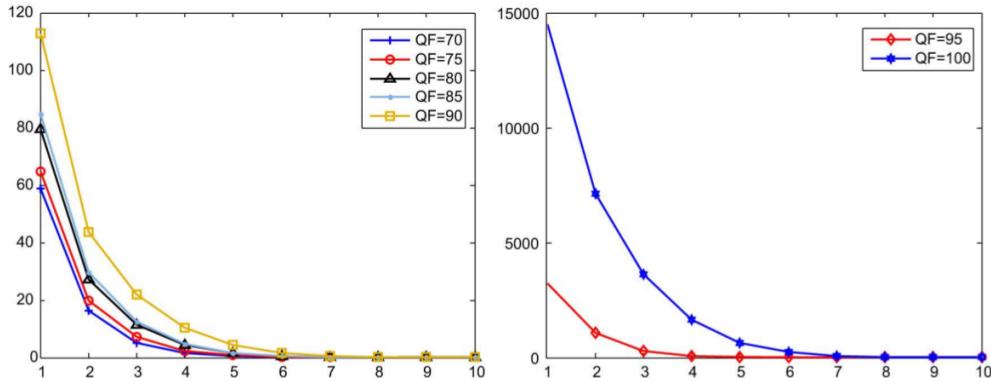


FIGURE 2.8 – Diminution du nombre de valeurs différentes au fil des recompressions, issu de [12].

nombre de coefficients DCT différents entre  $I$  et  $I$  recompressée avec sa table de quantification, cette nouvelle image est notée  $I'$

Ils vont ensuite perturber (ajouter ou soustraire 1) aléatoirement un nombre  $n/r$  de coefficients DCT de  $I'$  où  $n$  est le nombre de coefficients modifiés par coefficient différent de zéro et  $r$  est un ratio choisi de manière expérimentale.

Cette image perturbée  $I'$  est ensuite recompressée avec sa table de quantification pour donner  $I''$ . Le nombre  $Dm$  est calculé comme le nombre de coefficients DCT différents entre  $I'$  et  $I''$ . Cette étape est répétée  $k$  fois, à chaque fois avec la nouvelle image obtenue. On obtient alors  $\bar{Dm}$ , la moyenne de tous les  $Dm$ .

Une image  $I$  est doublement compressée si  $\bar{Dm} \geq D$ . En effet, si c'est la première recompression,  $D$  est grand (voir figure 2.8). C'est là qu'intervient le choix du ratio  $r$ .

Ce ratio est calculé de manière expérimentale, c'est à dire que grâce à une vérité terrain, et pour un certain ensemble d'images, les auteurs ont extrait le ratio donnant le meilleur résultat. Un des inconvénients de ce ratio est qu'il peut être sous-optimal pour un autre ensemble d'images, qui aura des caractéristiques différentes.

Un ratio mal choisi perturbera trop l'image et la modifiera plus que sa première compression ou alors ne la perturbera pas suffisamment et l'image serait alors détectée à tort comme simplement compressée.

Cette méthode offre un moyen de détecter la présence de double compressions avec la même matrice de quantification, un cas probable dans le cadre des réseaux sociaux. Même si elle a été pensée pour la détection dans le cas de la première compression, cette méthode pourrait être étendue à la détection de la la  $n^{i\text{ème}}$  et la  $n+1^{\text{ème}}$  compression. Contrairement aux auteurs, qui travaillent sur une image dans le vide, nous travaillons sur un ensemble d'images, et avons donc plus d'informations à notre disposition.

### Convergence des blocs lors de compressions successives

L'intérêt de la méthode proposée par CARNEIN et al. [6] est l'estimation du nombre de compressions JPEG qu'a pu subir l'image, une estimation qui va au-delà de deux ou trois compressions successives [12][17], il est en effet possible d'aller jusqu'à plusieurs centaines de compressions. Cette approche se base sur la notion de blocs cycliques.

Lors de compressions successives, une phénomène appelé *convergence de bloc* peut être observé. Un bloc converge lorsque la valeur des pixels du bloc à la compression  $t$  est égale à la valeurs des pixels à la compression  $t + 1$ , ce bloc est alors appelé stable [16]. Certains blocs cependant échappent à cet effet et exhibe un phénomène de cycle. C'est à dire que la valeur des pixels à la compression  $t$  est égale à la valeur des pixels à la compression  $t + n, n > 1$ . Le deuxième type de bloc à échapper à ce phénomène est les blocs plats, ayant tous les pixels de la même valeur, ils ne peuvent pas être cycliques puisque tous les coefficients DCT sauf le premier sont nuls.

Ces blocs sont nommés blocs compteurs. Ainsi, si un bloc de l'image a un cycle de longueur  $l = 9$ , il sera possible de savoir, modulo  $l$ , combien de compressions a subi l'image. Et avec plus de blocs, ayant chacun des  $l$  différents, il est possible d'aller bien plus loin dans le nombre de compressions. Prenons l'exemple de trois blocs pour lesquels  $l_1 = 2, l_2 = 5$  et  $l_3 = 9$ . Le nombre maximum de compressions successives qu'il sera possible d'estimer est le *plus petit commun multiple* des ces longueurs, autrement dit,  $\text{ppcm}(2, 5, 9) = 90$ , soit 90 compressions successives, et pour l'exemple illustré par la figure 2.9,  $\text{ppcm}(6, 7, 10, 12) = 420$ .

Les auteurs proposent deux approches pour les blocs cycliques. La première consiste à détecter les blocs cycliques à l'aide d'une recherche exhaustive sur l'ensemble des blocs et un certain nombre de compressions puis sélectionner les blocs intéressants. Cette méthode, bien qu'étant la plus simple, et ne modifiant pas l'image, peut cependant rencontrer des problèmes lorsqu'elle comporte un grand nombre de blocs plats, il peut en effet être difficile de trouver des blocs ayant des cycles de longueur intéressante. La deuxième approche proposée est donc d'insérer des blocs artificiellement créés dans l'image (voir figure 2.9).

Cette méthode nécessite de prendre un certain nombre de précautions et requiert une protection (padding) autour des blocs insérés. L'algorithme de sur-échantillonage est en effet parfois amené à utiliser les valeurs des pixels des blocs adjacents pour un meilleur rendu et provoquer un effet de débordement (spill over) des valeurs [5], ce qui modifie le bloc artificiellement créé (dans le vide) et donc modifie son cycle, rendant nulle son information.



FIGURE 2.9 – Exemple d'insertion de blocs, issu de [6].

Si l'insertion de blocs artificiels dans l'image est mise de coté, technique qui ne serait pas possible dans notre cas, le principal inconvénient de cette méthode est qu'elle se restreint à des images ayant toutes le même facteur de qualité, et en particulier  $Q = 100$ , pour lequel les cycles sont les plus longs, et donc donnent le plus d'informations. C'est un cas spécifique pour notre cadre d'utilisation : les réseaux sociaux, où les facteurs de qualité de compression varient en fonction de l'application utilisée. Cette méthode est en effet plus orientée vers le tatouage et le traçage d'images.

### 2.3 Distances entre distributions

Dans l'approche que nous proposons de développer lors de ce stage, il nous sera nécessaire de calculer des distances entre des distributions. C'est la raison pour laquelle nous avons décidé passer en revue un certain nombre de distance dans l'état de l'art.

Commençons par définir la différence entre une distance et une divergence. Une distance, contrairement à une divergence, est symétrique, c'est à dire que  $distance(I_m, I_n) = distance(I_n, I_m)$ , et elle vérifie l'inégalité triangulaire. Une distance est parfaite pour mesurer la proximité entre deux objets. Elle est cependant dans l'incapacité de nous donner des indications sur une direction.

Nous travaillons avec une relation asymétrique (relation parent-enfant), une divergence semble donc être mieux adaptée pour estimer la direction de cette relation (voir chapitre 3).

Il peut cependant être envisagé d'utiliser une distance pour estimer une direction si le calcul de la distance est fait après avoir appliqué une opération asymétrique. Comme expliqué chapitre 2 section 2.1, DIAS et al. mappent une image dans le domaine d'une autre, ce qui est une opération asymétrique, et donc qui permettrait d'utiliser une distance. C'est pourquoi nous traiterons aussi bien les distances que les divergences.

Il existe un grand nombre de mesure de distances [7], aussi, nous ne parlerons que des distances entre distributions les plus utilisées. Nous n'entrerons cependant pas dans les détails théoriques de ces calculs de distance, qui nécessiteraient une étude bibliographique à eux seuls, nous utiliserons ces distances comme des boîtes noires et choisirons de manière expérimentale la plus adaptée. Nous allons donc donner pour chaque distance son nom et sa formule.

Dans la suite de cette section,  $P$  et  $Q$  seront des densités de probabilité.

### Les distances

La distance euclidienne :

$$D_{euc}(P, Q) = \sqrt{\sum_{i=1}^d |P_i - Q_i|^2} \quad (2.4)$$

La distance de Bhattacharyya :

$$D_B(P, Q) = -\ln \sum_{i=1}^d \sqrt{P_i Q_i} \quad (2.5)$$

L'index de Jaccard est donné comme :

$$S_{Jac}(P, Q) = \frac{\sum_{i=1}^d P_i Q_i}{\sum_{i=1}^d P_i^2 + \sum_{i=1}^d Q_i^2 - \sum_{i=1}^d P_i Q_i} \quad (2.6)$$

Il correspond à la taille de l'intersection entre deux distributions divisé par la taille de leur union. La distance de Jaccard est :

$$D_{Jac}(P, Q) = 1 - S_{Jac}(P, Q) = \frac{\sum_{i=1}^d (P_i - Q_i)^2}{\sum_{i=1}^d P_i^2 + \sum_{i=1}^d Q_i^2 - \sum_{i=1}^d P_i Q_i} \quad (2.7)$$

### Les divergences

La divergence de Kullback–Leibler :

$$D_{KL}(P||Q) = \sum_{i=1}^d P_i \log \frac{P_i}{Q_i} \quad (2.8)$$

La K divergence :

$$D_{kdiv}(P||Q) = \sum_{i=1}^d P_i \log \frac{2P_i}{P_i + Q_i} \quad (2.9)$$

Au travers de cet état de l'art, nous avons pu voir différentes méthodes de calcul de l'arbre phylogénétique et avons vu que la difficulté se trouve dans l'estimation des relations de parenté

entre les images. Ce que nous avons tenté de résoudre en nous intéressant à l'analyse des compressions multiples, analyse qui est loin d'être triviale et qui souvent se limite aux premières recompressions. Nous avons également vu plusieurs calculs de distances qui serviront eux aussi à estimer les relations de parenté entre images.

## Méthode de calcul de phylogénie à l'aide des caractéristiques des coefficients DCT d'une bitmap

Nous tentons de reconstruire l'arbre de phylogénie d'un ensemble de near-duplicates. Contrairement aux approches de l'état de l'art qui utilisent des algorithmes de construction d'arbre complexes pour reconstruire l'arbre à partir d'une dissimilarity matrix, nous allons prendre une décision binaire entre deux images, à savoir "Cette image est-elle le parent de cette autre image ?". Cela nous permet de nous concentrer sur les images et leurs caractéristiques et avoir un algorithme de construction d'arbre simple. La figure 3.1 présente notre méthode.

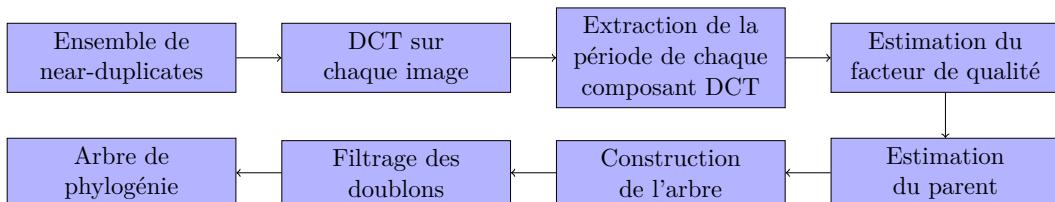


FIGURE 3.1 – Schéma de notre méthode

### 3.1 Principe et théorème

Pour tout couple d'images de notre ensemble, nous tentons donc de prendre une décision binaire, qui va nous permettre d'extraire une *matrice de parenté*. Rappelons que nous tentons de savoir si une image est l'ancêtre d'une autre, et pas seulement son parent direct. Nous mettons en place un framework pour tenter de résoudre ce problème :

**Définition.** Un *marqueur* est une caractéristique locale ou globale extraite de l'image qui indique qu'une certaine opération a été effectuée. Ce marqueur va se transmettre aux enfants de l'image, et serviront potentiellement à prouver qu'une image n'est pas le parent d'une autre.

Par exemple une image en noir et blanc ne peut pas être le parent d'une image couleur car le marqueur "couleur" est absent de l'image en noir et blanc.

**Définition.** Soit  $f(I_m, I_n)$  une fonction qui pour tout couple d'images  $(I_m, I_n)$  détecte à chaque fois qu'il est présent un marqueur indiquant qu'il n'y a pas de relation de parenté entre  $I_m$  et

$I_n$ . Cette fonction est appelée fonction de négation. Notons que ce type de fonction est plus ou moins abstrait et que c'est ici que se situe la difficulté.

**Théorème.** Pour tout couple d'images  $(I_m, I_n)$  d'un ensemble de near-duplicates, s'il n'existe pas de marqueur prouvant que  $I_m$  n'est pas le parent de  $I_n$ , alors il y a une relation parent-enfant entre  $I_m$  et  $I_n$ ,  $I_m \rightarrow I_n$ .

*Démonstration.* Si  $f(I_m, I_n)$  ne parvient pas à trouver de marqueur prouvant que  $I_m$  n'est pas de parent de  $I_n$  alors c'est que ce marqueur n'existe pas et donc que  $I_m$  est le parent de  $I_n$ , avec  $m < n$ .  $\square$

Ce framework a été imaginé lors de l'étude bibliographique, nous allons ensuite présenter notre méthode, et voir comment il a été utilisé et adapté.

## 3.2 Estimation du facteur de qualité d'une image JPEG

Nous travaillons sur des images ayant subi une compression JPEG. Ces images, au fil de leurs voyages sur les réseaux sociaux, on pu changer de format et avoir été convertie au format PNG par exemple. L'information du facteur de qualité utilisé pour compresser l'image, indispensable à nos calculs, a été perdue. Nous proposons donc d'estimer le facteur de qualité d'une bitmap n'ayant pas subi d'autre compression avec perte que le JPEG.

### Estimation de la table de quantification

Nous avons besoin des coefficients DCT de l'image pour pouvoir estimer la table de quantification. Deux choix s'offrent à nous : calculer une DCT à partir d'une bitmap, ou compresser une bitmap au format JPEG avec  $Q_f = 100$  et lire directement les coefficients DCT. Le deuxième choix s'avère être le plus simple à mettre en place et c'est celui que nous avons utilisé.

Comme expliqué section 2.2, les valeurs des coefficients DCT vont se rassembler autour des multiples de  $q$ , qui est le nombre utilisé pour la quantification. Chaque coefficient DCT forme donc un signal périodique, de période  $q$ . C'est cette période que nous allons tenter d'estimer. L'autocorrélation est la technique traditionnellement utilisée pour extraire une période.

Chaque implémentation de la compression JPEG possède sa propre table de quantification, un exemple de table de quantification est donnée figure 2.5. La table de quantification  $base(.,.)$  est donnée pour  $Q_f = 50$ , pour obtenir les tables de quantification pour les différents  $Q_f$ , les formules 3.1 et 3.2 sont utilisées. Cela permet, si c'est une table standard qui est utilisée, de ne pas avoir à stocker la table dans l'en-tête du fichier JPEG et de pouvoir la calculer à partir de  $Q_f$ .

$$\text{si } Q_f < 50 \quad Q_s = 5000/Q_f \quad \text{sinon} \quad Q_s = 200 - (Q_f \times 2) \quad (3.1)$$

$$q(u, v) = \frac{(base(u, v) \times Q_s) - 50}{100} \quad \text{en bornant } q(u, v) \text{ par 1 et 255} \quad (3.2)$$

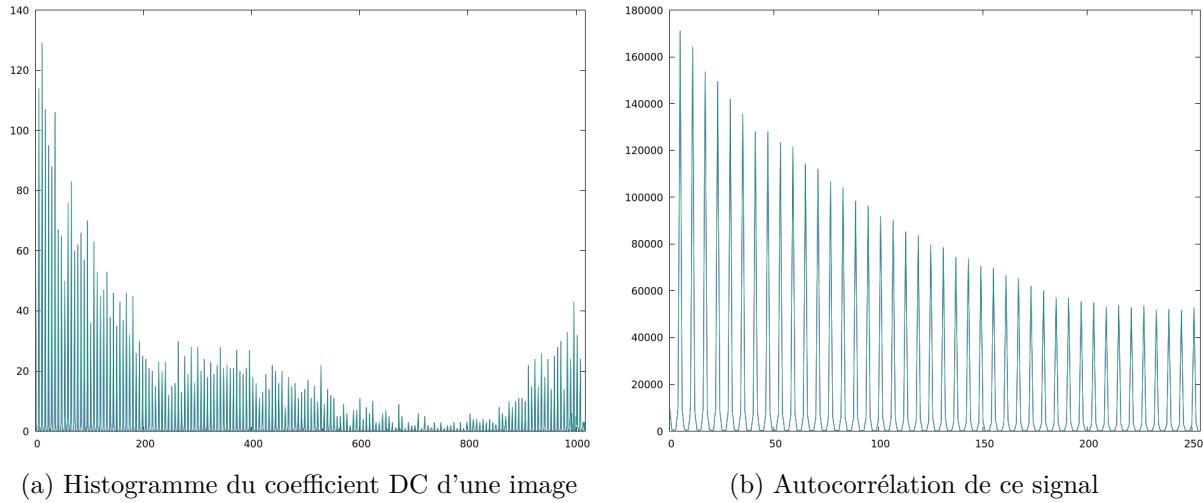


FIGURE 3.2 – Exemple de signal et son autocorrélation

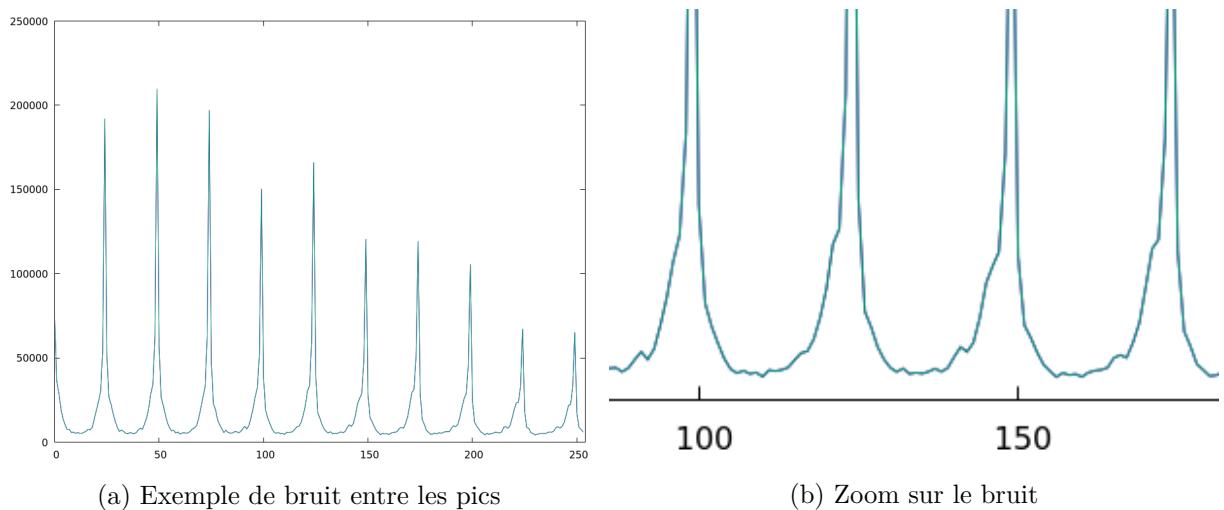


FIGURE 3.3 – Présence de bruit entre les pics

### L'autocorrélation et la détection de pics

L'autocorrélation est la multiplication du signal avec lui-même à différents points du temps. Ainsi, l'autocorrélation d'un signal périodique aura des valeurs maximales (des pics) lorsque le décalage temporel  $\tau$  est un multiple de la période, et donc que le signal est le plus similaire avec lui-même.

Nous savons que les valeurs du pas de quantification sont bornées par 1 et 255, nous allons donc calculer l'autocorrélation entre ces deux bornes. Un exemple de signal et son autocorrélation est disponible figures 3.2a et 3.2b.

La période va ensuite être le delta entre deux pics consécutifs. La détection d'un pic est triviale lorsque le signal est propre et ne comporte aucun bruit, une valeur est un pic si la valeur précédente et la valeur suivante sont strictement inférieures. Nous ne sommes hélas pas dans le

cas idéal, comme illustré figure 3.3, où de faux pics peuvent être détectés si on prend le signal en l'état.

La méthode la plus simple, efficace, peu coûteuse en temps de calcul et facile à mettre en place est de ne garder que les valeurs supérieures à un seuil. Notre seuil est calculé comme cinq fois la valeur moyenne de l'autocorrélation. Un exemple est disponible figure 3.4

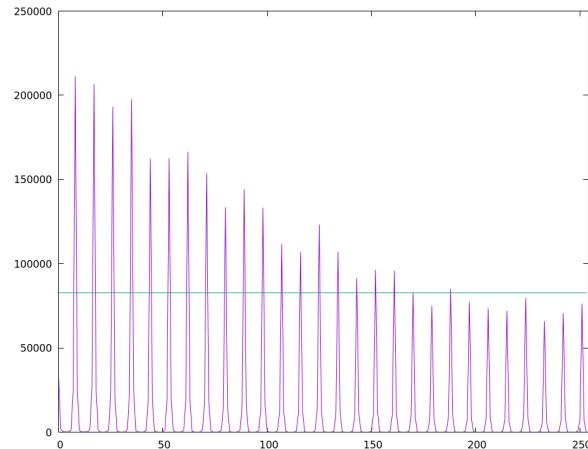


FIGURE 3.4 – Exemple d'autocorrélation avec un seuil (en vert)

Seuls les quelques premiers pics sont conservés, nous sommes en effet certains qu'ils seront au dessus du seuil, et donc qu'il n'y aura pas de pic manquant (comme les pics de la droite de la figure 3.4 qui alternent entre au dessus et en dessous du seuil) qui viendrait fausser le calcul de la période.

Il est important de noter que cette technique donne de très bons résultats, mais seulement dans notre cas d'étude, c'est à dire que le  $Q_f$  d'une image est strictement inférieur à celui de son parent. Dans le cas où le  $Q_f$  peut être supérieur, comme expliqué section 2.2, des doubles pics peuvent apparaître et rendre compliqué le calcul de la période. Un exemple est disponible figure 3.5.

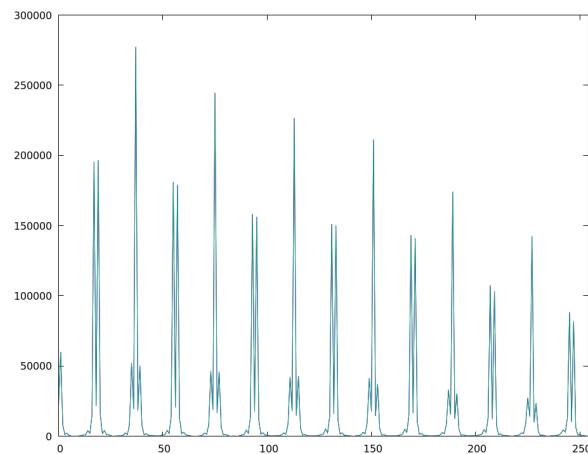


FIGURE 3.5 – Exemple d'autocorrélation avec des doubles pics

Cette opération est donc répétée pour les 35 premiers coefficients DCT de l'image pour obtenir  $\hat{q}(u, v)$  la table de quantification estimée. Nous n'estimons que les 35 premiers coefficients DCT de l'image, la quantification des derniers coefficients est trop importante et il y a trop de valeurs à zéro, et donc pas assez d'informations pour estimer de manière précise la période.

### Calcul de $Q_f$ à partir de $\hat{q}(u, v)$

Il va maintenant falloir calculer  $Q_f$  à partir de  $\hat{q}(u, v)$ , la table de quantification précédemment estimée. Deux choix s'offrent à nous : regarder parmi toutes les tables  $q_n(u, v)$ ,  $n \in \{1..100\}$  laquelle est la plus similaire (avec un calcul de distance par exemple), ou utiliser les équations 3.1 et 3.2.

L'avantage de la première méthode est qu'elle nous donne directement  $Q_f$ , c'est la table pour laquelle la distance entre  $q_n(u, v)$  et  $\hat{q}(u, v)$ ,  $n \in \{1..100\}$  est la plus faible. Le principal désavantage de cette technique est qu'elle est coûteuse en temps de calcul, en plus de manquer de précision.

L'autre méthode, même si elle ne demande presque aucun calcul, a un défaut majeur, elle ne permet pas d'obtenir  $Q_f$ . On peut en effet obtenir  $Q_s$  à partir de l'équation 3.2, mais il n'est pas possible d'obtenir  $Q_f$  à partir de  $Q_s$  sans savoir si  $Q_f$  est supérieur à 50.

Nous avons choisi de conjuguer les deux méthodes. Premièrement nous obtenons une estimation grossière de  $Q_f$  en regardant parmi toutes les tables laquelle est la plus similaire, puis avec cette estimation de  $Q_f$  nous calculons, manière plus précise le vrai  $Q_f$  à l'aide des équations 3.1 et 3.2. Le  $Q_f$  final est donc la moyenne des  $Q_f$  obtenus pour chaque coefficient DCT.

## 3.3 Estimation du parent

Le facteur de qualité utilisé pour compresser l'image nous est indispensable pour estimer le parent. En plus d'être un marqueur très efficace pour filtrer les images ne pouvant pas être le parent, le facteur de qualité nous sert à normaliser les images pour pouvoir les comparer.

Nous partons du principe que la compression JPEG est une opération déterministe, elle donnera, pour une même implémentation, toujours le même résultat. Cela implique que toutes les compressions d'une certaine image avec le même facteur de qualité seront strictement identiques, et ce quelque soit le nombre de compressions qu'à pu subir l'image au préalable.

Nous nous sommes limités aux compressions JPEG comme expliqué précédemment, nous savons donc que pour passer d'une image parent à son image enfant, l'image parent aura été recompressée. L'image parent est donc à une compression de l'image enfant.

Nous allons utiliser ces précédents principes pour identifier avec précision le parent :

Soit  $I_m$  et  $I_n$  deux images et  $Q_{f_m}$  et  $Q_{f_n}$  leurs facteurs de qualité respectifs, et  $C(I, Q)$  l'opérateur de compression, avec  $I$  l'image et  $Q$  le facteur de qualité.

Si  $I_m = C(I_n, Q_{f_m})$ , alors  $I_n$  est le parent direct de  $I_m$ .

C'est en pratique presque aussi simple qu'en théorie, on voit bien que tout dépend du facteur de qualité, or nous estimons ce facteur de qualité, et il peut arriver qu'il ne soit pas exact, mais un peu trop grand ou un peu trop petit. Nous n'allons donc pas nous contenter de comparer les images compressées avec le  $Q_f$  estimé, nous allons également regarder les  $Q_f$  alentours jusqu'à

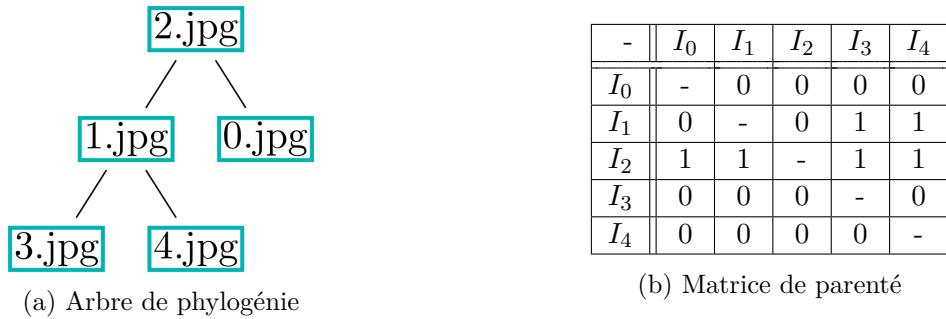


FIGURE 3.6 – Une arbre de phylogénie et sa matrice de parenté.

ce qu'un parent soit détecté. La précision de l'estimation du parent est donc directement corrélée avec le temps de calcul de notre algorithme.

Cette méthode, même si elle donne d'excellents résultats pour trouver le parent direct, ne permet pas de trouver les grands parents ou les ancêtres plus éloignés. Nous tentons en effet de prendre une décision binaire : "Cette image est-elle le parent de cette autre image ?". Une décision binaire est difficile lorsque l'on travaille sur des valeurs dont les bornes dépendent de l'input. Il est alors difficile d'utiliser des seuils pour correctement filtrer les valeurs. L'alternative logique est de ne pas utiliser de seuil, et comparer les valeurs entre elles.

On peut raisonnablement penser que l'ancêtre le plus probable est l'image ayant la plus petite valeur (issue d'une métrique quelconque) parmi toutes les autres valeurs. Le problème est que l'on perd alors le principe de décision binaire entre deux images.

### 3.4 Reconstruction de l'arbre

La phase d'estimation des parent nous a permis de créer une matrice binaire de taille  $n \times n$ ,  $n$  étant le nombre d'images dans le set. Cette matrice est appelée *matrice de parenté*. Une valeur à 1 à l'indice  $(i, j)$  de cette matrice indiquera que l'image  $I_i$  est un des ancêtres de  $I_j$ . Une valeur à 0 indiquera l'absence de parenté. Un exemple d'arbre et de matrice de parenté est disponible figure 3.6. On peut noter plusieurs choses intéressantes de cet exemple. Une colonne où toutes les valeurs sont à 0 indiquera que l'image n'a aucun parent, il s'agit donc de la racine, qui par définition n'a pas de parent. Une ligne où toutes les valeurs sont à 0 signifie que l'image n'est le parent de personne, c'est à dire qu'il s'agit d'une feuille. On peut ainsi généraliser et dire que plus une colonne contient de valeurs à 0, moins cette image a d'ancêtres, et donc qu'elle est proche de la racine.

Nous donnons l'algorithme de reconstruction d'arbre à titre indicatif algorithme 1.

À chaque itération, une image est sélectionnée comme la racine (lignes 1 et 11) et nommée *root*, elle est retirée (ligne 7) des ancêtres des autres images si c'est un ancêtre. Si ces autres images n'ont plus d'ancêtre (ligne 8), c'est que *root* était le parent direct de l'image en train d'être traitée, cette image est donc ajoutée comme enfant de *root* (ligne 9). La ligne 5 permet de ne traiter qu'une fois chaque image comme racine potentielle.

**Data:**  $M$  a  $n \times n$  parentage matrix

**Result:** the root of the tree

```

1 nextRoot  $\leftarrow$  row with min sum of elements;
2 treeRoot  $\leftarrow$  nextRoot;
3 forall rows row of M do
4   | root  $\leftarrow$  nextRoot;
5   | mark root as done;
6   | for i  $\leftarrow$  0 to n do
7     |   | row[i]  $\leftarrow$  0;
8     |   | if sum of elements of row == 0 then
9       |     | add i as child of root;
10      |     | end
11      |     | if row has the smallest sum of elements and is not marked as done then
12        |       | nextRoot  $\leftarrow$  i;
13        |       | end
14    |   | end
15 end
16 return treeRoot
```

**Algorithm 1:** Construction de l'arbre

Cet algorithme a une complexité de  $O(n^2)$ . Il y a deux boucles imbriquées, et si les sommes sont calculées une seule fois au début et mises à jour à chaque fois qu'un parent est enlevé, il n'y a pas de boucle supplémentaire augmentant la complexité.

### 3.5 Filtrage des doublons

Nous avons expliqué précédemment que des images avec le même historique de compression étaient identiques. Le fait qu'elles soient identiques peut fausser les métriques introduites par DIAS et al. En effet, prenons le cas de trois images  $I_m$ ,  $I_n$  et  $I_k$ , où  $I_m$  et  $I_n$  sont identiques, et  $I_k$  est l'enfant de  $I_m$ . Ces deux images étant identiques,  $I_k$  peut aussi bien être l'enfant de  $I_m$  que de  $I_n$ , sans que cela ne change rien aux caractéristiques de  $I_k$ . Mais lors de la reconstruction de l'arbre, si  $I_k$  est marquée comme enfant de  $I_n$ , les métriques de DIAS et al. marqueront  $I_k$  comme ayant le mauvais parent, et donc donneront un mauvais score à la reconstruction, alors qu'elle est fondamentalement juste. Le processus de filtrage des doublons est illustré figure 3.7.

Ce filtrage est appliqué aussi bien à l'arbre original qu'à l'arbre reconstruit, pour qu'ils soient comparables.

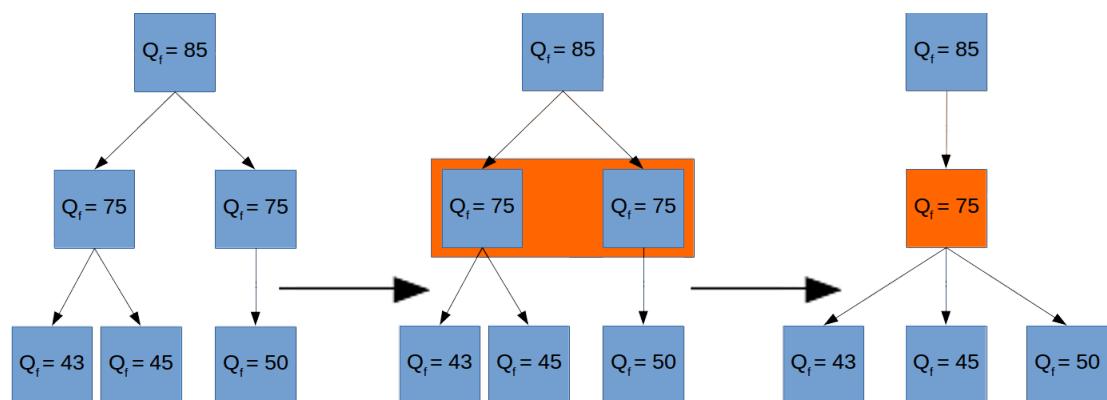


FIGURE 3.7 – Fusion des images identiques dans l'arbre

## Résultats

### 4.1 Génération des données

Nous avons choisi de créer trois datasets différents, un avec des arbres de 15 images, un autre avec des arbres de 25 images et un dernier avec des arbres de 50 images. Cela nous permettra d'observer les différences éventuelles que peuvent amener des choses telles que la distance à la racine et le nombre de recompression dans la précision de l'estimation du parent.

La génération se fait comme suit : à partir d'une image graine (non compressée), une première image est compressée avec  $85 < Q_f < 99$ , c'est l'image racine. Cette image racine est ajoutée à notre pool d'images. Tant que le nombre désiré d'images n'a pas été atteint, une image est prise aléatoirement dans la pool, est compressée avec  $Q_{f_{parent}} - 15 < Q_{f_{enfant}} < Q_{f_{parent}} - 1$  et est ajoutée à la pool d'image. On fixe cependant  $Q_f = 30$  comme limite basse du facteur de qualité, au delà de cette limite, l'image est trop détériorée, et en plus de ne pas être un cas d'utilisation réaliste dans les réseaux sociaux, ne contient plus assez d'informations pour être correctement utilisée.

La génération des datasets en eux même s'est fait à partir des six images d'origine de *BOWS 2*[4] où chaque image à servi à générer douze arbres pour chaque dataset, pour un total de 72 arbres par dataset, soit 6480 images tout dataset confondu.

### 4.2 Résultats expérimentaux

Nous allons commencer par donner le résultat de nos algorithme sur un arbre choisis aléatoirement dans notre dataset, pour bien détailler la sortie de chaque étape, puis nous allons donner des résultats plus significatifs obtenus en lançant notre programme sur l'ensemble des données.

#### Détail sur un arbre

Nous choisissons un arbre de 15 images, les résultats ne sont pas significativement différent avec les autres tailles d'arbre, et il est plus simple de tout montrer avec un nombre restreint d'images. Notons que les résultats pour cet arbre sont donnés à titre d'exemple, et ne représentent pas forcément les résultats que l'on peut obtenir sur un grand nombre d'arbre.

Commençons par la détection de pics et l'autocorrélation. Nous utilisons les 35 premiers coefficients DCT, les coefficients suivant sont trop quantifiés, et souvent à 0, et ne contiennent pas suffisamment d'informations pour être intéressants, ils peuvent même fausser les résultats. Nous donnons l'histogramme du coefficient DC ainsi que son autocorrélation figure 4.1, puis le coefficient 18 dans l'ordre zigzag figure 4.2 et enfin le coefficient 31 figure 4.3 pour permettre au lecteur de visualiser les données avec lesquels nous travaillons. Ces exemples sont donnés à titre indicatif, l'apparence des histogrammes et autocorrélations peut grandement varier selon le facteur de qualité de l'image. L'image dont sont tirées les figures est compressée avec comme historique de compression  $H_{Q_f} = \{86, 85, 73\}$ . C'est la période, ou l'écart entre deux pics de l'autocorrélation de chaque coefficient DCT qui nous cherchons à obtenir pour ensuite pouvoir les comparer avec les différentes tables de quantification.

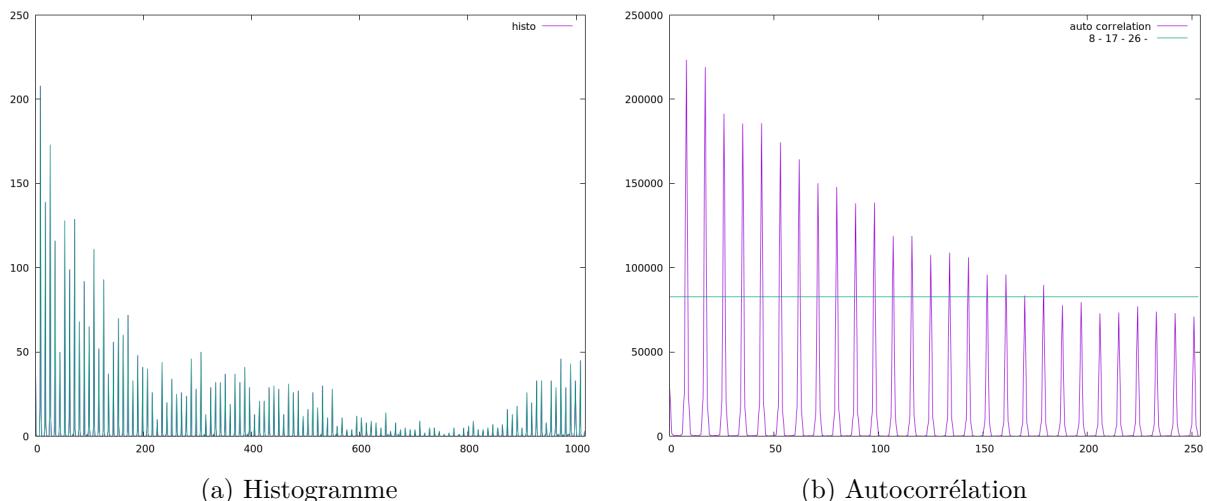


FIGURE 4.1 – Histogramme et autocorrélation du coefficient DC avec figure 4.1b, en vert le seuil et en haut à droite les pics détectés

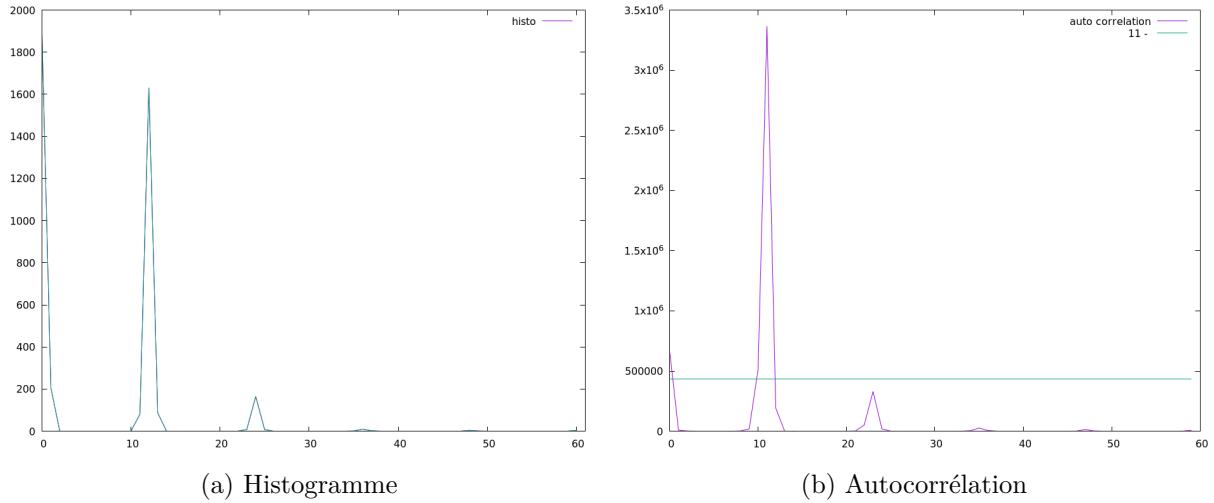


FIGURE 4.2 – Histogramme et autocorrélation du 18ème coefficient AC avec figure 4.2b, en vert le seuil et en haut à droite les pics détectés

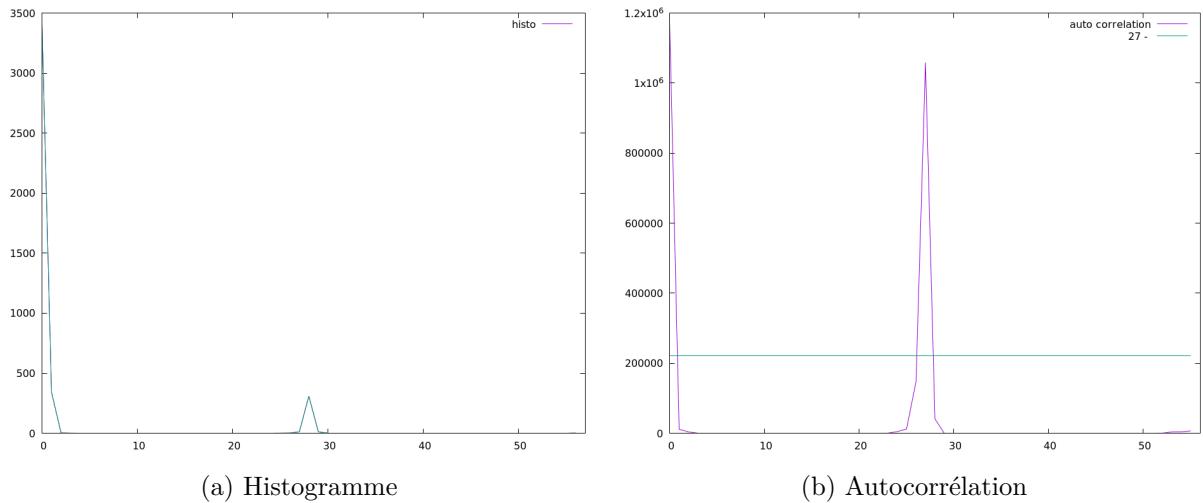


FIGURE 4.3 – Histogramme et autocorrélation du 31ème coefficient AC avec figure 4.3b, en vert le seuil et en haut à droite les pics détectés

On voit très nettement que la quantité d'information disponible baisse très fortement lorsque l'on s'éloigne du coefficient DC. C'est pour cette raison que nous restons sur 35 coefficients.

Les 35 périodes calculées sont données table 4.5, avec pour comparaison la table de quantification d'une images compressée avec  $Q_f = 71$  (table estimée), comme l'a été estimée l'image après la première estimation grossière, et la table de quantification de  $Q_f = 73$  (table réelle), le vrai  $Q_f$  de l'image. Les valeurs à 1 indiquent qu'aucune valeur de période n'a pu être calculée, et sont donc ignorées.

<b>table calculée</b>	9	6	6	8	6	5	9	8	7	8	10	9	9	10	13	22	14
<b>table estimée</b>	9	7	6	6	7	8	8	8	8	9	14	11	9	10	10	14	13
<b>table réelle</b>	9	6	6	5	6	8	8	7	8	9	13	10	9	9	10	13	12

FIGURE 4.4 – Premières valeurs des tables de quantification

<b>table calculée</b>	13	12	12	13	1	19	20	16	22	31	28	1	1	31	28
<b>table estimée</b>	13	14	15	23	30	34	23	17	21	20	28	42	37	32	32
<b>table réelle</b>	12	13	14	22	28	31	22	16	20	19	26	39	35	30	30

FIGURE 4.5 – Dernières valeurs des tables de quantification

Toutes les images de l’arbre auront leur  $Q_f$  grossièrement estimé de la même manière, puis une estimation plus précise grâce aux formules 3.1 et 3.2 affinera les résultats. La table 4.6 montre comment ont été estimés les images de l’arbre de manière grossière puis de manière précise, en donnant également leur véritable  $Q_f$  comme comparaison. L’erreur moyenne d’estimation de la table 4.6 est de 0.533333.

Image	$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$	$I_8$	$I_9$	$I_{10}$	$I_{11}$	$I_{12}$	$I_{13}$	$I_{14}$
<b>Estimation grossière</b>	71	81	71	58	78	46	61	85	81	79	65	74	84	71	74
<b>Estimation fine</b>	71	81	71	58	78	46	61	87	82	80	65	74	85	71	75
<b>Valeurs réelles</b>	73	81	71	57	79	44	62	86	82	80	65	74	85	71	75

FIGURE 4.6 – Estimations grossières et fines des  $Q_f$  des images de l’arbre

Ces  $Q_f$  vont ensuite servir à compresser les images pour pouvoir les comparer deux à deux, il semble peu intéressant de détailler les calculs qui sont effectués, aussi nous nous contenterons de donner la matrice de parenté (figure 4.7) produite en sortie. L’arbre calculé à partir de cette matrice (figure 4.8) est exactement identique à l’arbre original.

-	$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$	$I_8$	$I_9$	$I_{10}$	$I_{11}$	$I_{12}$	$I_{13}$	$I_{14}$
$I_0$	-	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$I_1$	0	-	0	0	0	0	0	0	1	0	0	0	0	0	0
$I_2$	0	0	-	0	0	0	0	0	0	0	0	0	1	0	0
$I_3$	0	0	0	-	0	0	0	0	0	0	0	0	0	1	0
$I_4$	0	0	0	0	-	0	0	0	0	0	0	0	1	0	0
$I_5$	0	0	0	1	0	-	0	0	0	0	0	0	0	0	0
$I_6$	0	0	0	0	0	0	-	0	0	0	1	0	0	0	0
$I_7$	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0
$I_8$	0	0	0	0	0	0	0	1	-	0	0	0	0	0	0
$I_9$	0	1	0	0	0	0	0	0	0	-	0	0	0	0	0
$I_{10}$	0	0	0	0	0	0	0	0	0	0	-	0	0	1	0
$I_{11}$	0	0	0	0	0	0	0	1	0	0	0	-	0	0	0
$I_{12}$	0	0	0	0	0	0	0	1	0	0	0	0	-	0	0
$I_{13}$	0	0	0	0	0	0	0	0	1	0	0	0	0	-	0
$I_{14}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-

FIGURE 4.7 – Matrice de parenté

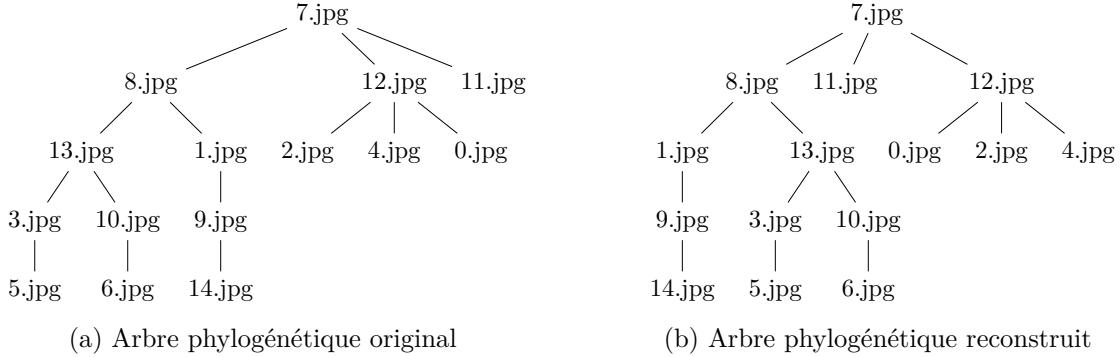


FIGURE 4.8 – Exemple d’arbre original et d’arbre reconstruit

On voit qu’il y a au maximum un seul 1 par ligne dans la matrice de parenté. En effet, seuls les parents directs sont détectés, et pas les ancêtres plus éloignés.

## Résultats sur les datasets

Nous avons fait tourner notre collection d’algorithmes sur les trois datasets, et avons obtenu des résultats très satisfaisants. Nous mesurons la qualité de nos résultats grâce à sept métriques, trois servant à mesurer la précision de l’estimation du facteur de qualité et les quatre autres servant à mesurer la qualité du calcul de l’arbre sont celles introduites par DIAS et al. [9].

Les trois métriques servant à mesurer la précision de l’estimation du  $Q_f$  sont l’erreur moyenne d’estimation, qui servira à savoir si l’estimation est juste la plupart du temps, la sur-estimation moyenne et la sous-estimation moyenne, qui, si tout va bien, seront symétriques, et serviront à savoir si notre algorithme a plutôt tendance à sur-estimer ou sous-estimer  $Q_f$ , et dans quel cas.

Métrique \ Dataset	15 images	25 images	50 images
<b>mean error</b>	0.516667	0.639444	0.833333
<b>mean underestimate</b>	1.61111	1.86111	1.94444
<b>mean overestimate</b>	1.06944	2.125	6.68056
<b>roots</b>	0.958333	0.888889	0.847222
<b>edges</b>	0.997024	0.992477	0.989796
<b>leaves</b>	0.995949	0.991517	0.987451
<b>ancestry</b>	0.994444	0.968889	0.969699

FIGURE 4.9 – Table de résultats, les trois premières métriques sont des moyennes d'entiers et les quatre dernières sont des pourcentages

La table 4.9 montre les résultats obtenus par nos algorithmes. On peut directement remarquer que les résultats sont strictement meilleurs lorsque l'on réduit le nombre d'images dans le set. À noter que le meilleur résultat possible pour les trois premières métriques est zéro, c'est à dire que tout les  $Q_f$  ont été estimés parfaitement, et que le meilleur résultat pour les quatre dernières est un, lorsque la vérité terrain et l'arbre reconstruit sont identiques.

Il semble important de préciser que l'estimation du  $Q_f$  rencontre des difficultés lorsque  $Q_f$  est proche de 100. En effet un grand nombre des valeurs de la table de quantification est à 1 ou 2, ce qui ne quantifie pas ou pas suffisamment le coefficient DCT pour que nous puissions extraire suffisamment d'information pour discriminer une image compressée avec  $Q_f = 97$  d'une image compressée avec  $Q_f = 99$  par exemple. C'est en partie une des causes du manque de précision d'identification de la racine. En effet, dans notre cas d'étude, la racine est l'image avec le  $Q_f$  le plus proche de 100, il est difficile de l'identifier avec certitude lorsque l'estimation du  $Q_f$  n'est pas certaine.

On voit que les erreurs moyennes, même si elles restent inférieures à 1 pour les trois datasets, et sont donc tout à fait correctes, sont de moins en moins bonnes lorsque le nombre d'images augmente, avec notamment la sur-estimation moyenne qui est très importante pour le dataset de 50 images.

Une telle sur-estimation s'explique par le fait que les coefficients DCT deviennent trop bruités pour pouvoir correctement estimer la période lorsqu'ils ont été compressés un trop grand nombre de fois, le signal n'est plus périodique et l'autocorrélation ne montre pas de pics. Lorsqu'aucun pic n'est détecté, notre algorithme estime que l'image a un  $Q_f$  élevé, ce qui fausse la reconstruction de l'arbre, car dans notre cas, une image ayant subi un grand nombre de recompressions a forcément un  $Q_f$  petit. C'est une autre source d'erreurs dans l'estimation de la racine. Plus l'arbre est grand, plus les images ont été compressées un nombre important de fois, et donc plus elles ont de chance de pouvoir fausser les résultats. Le problème vient de notre algorithme de reconstruction d'arbre. Une image est marquée comme la racine si elle n'a aucun parent. La racine n'a effectivement aucun parent, mais une image ayant eu son  $Q_f$  mal estimé n'aura aucun parent **détecté**. Il devient dans ce cas difficile de savoir la quelle des deux est la vraie racine.

Nous avions considéré que deux images avec le même historique de compression étaient identiques, il semble cependant que les images ayant un petit  $Q_f$  (inférieur à 35 environ) échappent parfois à ce phénomène, une image parent compressée avec le  $Q_f$  de son enfant et son enfant sont parfois différentes, sans que nous ayons pu identifier la cause de ce problème.

Un autre problème, qui lui n'affecte pas la racine, mais plutôt le reste de l'arbre, est la convergence des blocs, comme expliqué chapitre 2 section 2.2. Nous avons fixé comme limite basse pour la génération de l'arbre  $Q_f = 30$ , si une image avec  $Q_f = 30$  a un enfant, cet enfant aura également  $Q_f = 30$ , et ce de manière récursive. On peut donc arriver dans le cas où les enfants sont identiques à leurs parents, ce qui rend impossible leur différenciation, ce phénomène est fort heureusement assez rare.

## Perspectives et conclusion

Au terme de cette étude, nous avons pu voir que le problème de la reconstruction d'arbre phylogénétique d'images dans les réseaux sociaux n'est pas trivial et peut se diviser en deux sous-parties : l'identification de la racine et l'estimation des relations parent-enfant. Nous avons également pu voir dans l'état de l'art les différentes approches de reconstruction d'arbre de phylogénie. Nous avons choisi de nous concentrer uniquement sur l'étude des recompressions JPEG pour le calcul de l'arbre. Cette décision est motivée par le fait que nous sommes intéressés par le domaine du forensics et que l'étude des recompressions un sujet suffisamment vaste et complexe. Cela nous a amené à étudier la détection des compressions multiples.

De cette étude nous avons formalisé un théorème et réduit la reconstruction de l'arbre à une décision binaire entre deux images. Nous avons également précisément identifié les buts à atteindre dans la suite de ce stage : identifier une fonction qui trouve, s'il est présent, un marqueur qui prouve qu'il n'y a pas de relation parent-enfant entre deux images.

## Bibliographie

- [1] Herbert BAY et al. “Speeded-up robust features (SURF)”. In : *Computer vision and image understanding* 110.3 (2008), p. 346–359.
- [2] Tiziano BIANCHI et Alessandro PIVA. “Detection of nonaligned double JPEG compression based on integer periodicity maps”. In : *Information Forensics and Security, IEEE Transactions on* 7.2 (2012), p. 842–848.
- [3] Tiziano BIANCHI et Alessandro PIVA. “Image forgery localization via block-grained analysis of JPEG artifacts”. In : *Information Forensics and Security, IEEE Transactions on* 7.3 (2012), p. 1003–1017.
- [4] *BOWS 2 dataset*. URL : <http://bows2.ec-lille.fr/index.php>.
- [5] Matthias CARNEIN et al. “Forensics of high-quality JPEG images with color subsampling”. In : *Information Forensics and Security (WIFS), 2015 IEEE International Workshop on*. IEEE. 2015, p. 1–6.
- [6] Matthias CARNEIN et al. “Telltale Watermarks for Counting JPEG Compressions”. In : *Proceedings of the Electronic Imaging 2016*. Publication status : Published. San Francisco, USA, 2016.
- [7] Sung-Hyuk CHA. “Comprehensive survey on distance/similarity measures between probability density functions”. In : *City* 1.2 (2007), p. 1.
- [8] Zanoni DIAS et al. “First steps toward image phylogeny”. In : *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*. IEEE. 2010, p. 1–6.
- [9] Zanoni DIAS et al. “Image phylogeny by minimal spanning trees”. In : *Information Forensics and Security, IEEE Transactions on* 7.2 (2012), p. 774–788.
- [10] Xiaoying FENG et Gwenaël DOËRR. “JPEG recompression detection”. In : *IS&T/SPIE Electronic Imaging*. International Society for Optics et Photonics. 2010, 75410J–75410J.
- [11] Martin A. FISCHLER et Robert C. BOLLES. “Random Sample Consensus : A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In : *Commun. ACM* 24.6 (juin 1981), p. 381–395. ISSN : 0001-0782. DOI : 10.1145/358669.358692. URL : <http://doi.acm.org/10.1145/358669.358692>.

- [12] Fangjun HUANG et al. “Detecting double JPEG compression with the same quantization matrix”. In : *Information Forensics and Security, IEEE Transactions on* 5.4 (2010), p. 848–856.
- [13] Alexis JOLY et al. “Content-based copy retrieval using distortion-based probabilistic similarity search”. In : *Multimedia, IEEE Transactions on* 9.2 (2007), p. 293–306.
- [14] *JPEG*. URL : <https://fr.wikipedia.org/wiki/JPEG>.
- [15] Lyndon KENNEDY et Shih-Fu CHANG. “Internet image archaeology : automatically tracing the manipulation history of photographs on the web”. In : *Proceedings of the 16th ACM international conference on Multimedia*. ACM. 2008, p. 349–358.
- [16] ShiYue LAI et Rainer BOHME. “Block convergence in repeated transform coding : JPEG-100 forensics, carbon dating, and tamper detection”. In : *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, p. 3028–3032.
- [17] Jan LUKÁŠ et Jessica FRIDRICH. “Estimation of primary quantization matrix in double compressed JPEG images”. In : *Proc. Digital Forensic Research Workshop*. 2003, p. 5–8.
- [18] *Phylogénie*. URL : <https://fr.wikipedia.org/wiki/Phylogen%C3%A8se>.
- [19] Gregory K WALLACE. “The JPEG still picture compression standard”. In : *Consumer Electronics, IEEE Transactions on* 38.1 (1992), p. xviii–xxxiv.