Applications multimédia

TD et TME

Jean-Loup Guillaume

TD 1

TECHNIQUES DE CODAGE ET DE COMPRESSION

1. LANGAGE / CODAGE / VALENCE

1.1. Rappels

Toute fraction intelligible d'un message est constituée de ***symboles***. Le ***langage*** est l'ensemble de ces symboles.

Un codage va associer à chaque symbole un ***mot de code***. Chaque mot de code est constitué d'un ensemble de ***signes élémentaires***, encore appelés ***symboles de code***, et est de **longueur l_i** correspondant au nombre de signes élémentaires qui le décrivent. La ***valence*** du codage est le nombre de symboles de code. La ***taille*** d'un codage est le nombre de mots de code.

Si tous les mots de code n'ont pas tous la même longueur, le codage est ***de longueur variable***. Le nombre de mots de code associés à un langage dont la longueur est égale à p est alors noté L_p.

Un codage est :

- ***intelligible*** si la suite des signes élémentaires correspondant à une succession de symboles possède une unique interprétation ;
- ***instantané*** s'il est possible de le décoder au fur et à mesure de l'arrivée des mots de code ;
- ***préfixe***, si aucun mot de code n'est le début d'un autre mot de code ;
- ***complet*** s'il est intelligible et si tout ajout d'un mot de code de longueur inférieure ou égale à n le rend inintelligible et $L_p = 0$ pour tout $p > n$.

Une condition nécessaire pour qu'un codage de valence V (dont les mots de code ont une longueur maximale n) soit complet et intelligible est donnée par l'égalité de Kraft-McMillan :

$$\sum_{p=1}^{n} \frac{L_p}{V^{p-1}} = V$$

1.2. Exercices

Soit le langage représentant les quatre symboles *A, C, G, T*. On considère le codage suivant : *A* est représenté par le mot de code « 0 », *C* par « 01 », *G* par « 001 », *T* par « 101 ».

1.2.1. Quels sont les symboles de code ? Quelle est la valence du codage ?

1.2.2. Quels sont les valeurs L_1, L_2, L_3, L_4 ? Est-ce un codage intelligible, préfixe, vérifie t'il l'égalité de Kraft-McMillan ?

On considère maintenant tous les codes bivalents (de valence 2) et de longueur maximale 3.

1.2.3. Ecrire l'égalité de Kraft-McMillan dans ce cas particulier.

1.2.4. Résoudre cette équation (donner toutes les valeurs possibles de L_1, L_2 et L_3 vérifiant cette égalité).

1.2.5. Pour chaque triplet de valeurs, donner un code bivalent asssocié.

Soit le codage représentant les quatre symboles *A*, *C*, *G*, *T* où *A* est représenté par le mot de code « 0 », *C* par « 100 », *G* par « 101 », *T* par « 111 ».

1.2.6. Quels sont les valeurs L_1, L_2, L_3, L_4 ? Est-ce un codage intelligible ? Est-ce un codage complet ?

2. CODAGES COMPRESSIFS : SHANNON, FANO-SHANNON ET HUFFMAN

Dans toute la suite, sauf mention explicite du contraire, on considère des codes binaires.

2.1. Rappels

A chaque symbole s_i est associée une **probabilité** p_i représentant la fréquence d'occurrence du symbole. Ces probabilités sont estimées, soit *a priori* sur des *corpus* (échantillons) représentatifs de l'application considérée, soit *a posteriori* sur le corpus réel de l'application. La première solution ne fournit qu'une estimation, alors que la seconde nécessite un pré-traitement du corpus.

On définit l'***entropie*** d'une source de la façon suivante :

$$H(S) = -\sum_i p_i \log_2(p_i)$$

La base du logarithme fixe l'unité de l'entropie : dans le cas de la base 2, l'entropie s'exprime en ***shannon*** (sh). On peut démontrer que l'entropie est maximum dans le cas de l'équiprobabilité des symboles.

L'efficacité d'un code est estimée par ses caractéristiques :

- **Longueur moyenne** : $l_{moy} = \sum_i l_i \ p_i$. On peut montrer que $\frac{H(S)}{\log_2(V)} \le l_{moy}$, la limite théorique est donc : $l_{min} = \frac{H(S)}{\log_2(V)}$, ce qui se simplifie si le code est bivalent.
- **Rendement** $R = \frac{l_{min}}{l_{moy}} = \frac{H(S)}{l_{moy}\log_2(V)}$
- **Redondance** : $\rho = 1 - R$

Tous ces paramètres sont relatifs à l'entropie de la source.

2.2. Entropie d'une source

Soit une source (S) à 11 symboles (s_1 à s_{11}) définie par les probabilités suivantes :

S	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
p_i	0.22	0.15	0.12	0.11	0.10	0.08	0.07	0.06	0.04	0.03	0.02

2.2.1. Calculer l'entropie de la source.

2.2.2. Que vaudrait l'entropie si tous les symboles étaient équiprobables ?

2.3. Codage de Shannon

Les symboles s_i sont codés par un nombre l_i d'éléments unitaires tel que

$$-\frac{\log_2(p_i)}{\log_2(V)} \leq l_i < -\frac{\log_2(p_i)}{\log_2(V)} + 1$$

2.3.1. Donner un code de Shannon binaire pour la source ci-dessus. Une méthode simple consiste à calculer la longueur de chaque mot de code puis à construire un arbre binaire respectant ces longueurs

2.3.2. Calculer les caractéristiques de ce code : longueur moyenne, rendement et redondance.

2.4. Codage de Fano-Shannon

Il s'agit de construire un arbre en équilibrant à chaque fois au maximum les sous-arbres droit et gauche.

2.4.1. Donner un code binaire pour la source ci-dessus en appliquant la méthode de Fano-Shannon.

2.4.2. Calculer les caractéristiques de ce code.

2.5. Codage de Huffman

L'algorithme consiste à construire un arbre *V*-aire à partir des feuilles (symboles d'origine) vers la racine :

1. Ordonner les symboles suivant les probabilités croissantes.
2. Rassembler***** les *V* symboles les moins probables en un super-symbole dont la probabilité est la somme des probabilités des symboles intervenant. Adjoindre à chacun des *V* symboles un élément différent de l'alphabet.
3. Si la nouvelle source comporte plus d'un symbole, aller en 2 sinon fin de l'algorithme.

Le code de Huffman des symboles de source s'obtient par un parcours de la racine vers les feuilles.

***Attention :** Pour $V > 2$, si $r = (n-1) \bmod (V-1) \neq 0$ alors au premier passage à l'étape 2, rassembler r+1 symboles.

2.5.1. Donner un code binaire pour la source ci-dessus en appliquant la méthode de Huffman. Par convention, on choisira de coder les symboles ou super-symboles de plus faible probabilité par « 0 ».

2.5.2. Calculer les caractéristiques de ce code.

2.5.3. Soit « 001111001011011 » la séquence interprétée par un décodeur en bande de base, à quel message correspond cette séquence ?

2.5.4. Un code quaternaire trouve son application dans le cadre d'une transmission en bande de base d'un codage de valence 4. Donner un code quaternaire en appliquant la méthode de Huffman en faisant attention à la note plus haut. Quel code préfèrera-t-on (binaire ou quaternaire) ?

2.6. Comparaison de l'efficacité des codes

2.6.1. Conclure sur l'efficacité des codes binaires. L'un de ces codes est-il absolument optimal ?

3. CODAGE PAR BLOC DE SYMBOLES

3.1. Rappels

L'objectif d'un codage par bloc de symboles est d'améliorer l'efficacité d'un codage par symbole (non optimal) pour se rapprocher de la borne inférieure théorique.

3.2. Exercices

On considère 2 sources S_A et S_B, chacune constituée de 3 symboles et définies par leurs probabilités :

S_A	s_1	s_2	s_3
$P(S_A = s_i)$	0.38	0.34	0.28

S_B	s_1	s_2	s_3
$P(S_B = s_i)$	0.5	0.25	0.25

3.2.1. Calculer l'entropie des deux sources.

3.2.2. Quel est, parmi les codages de Shannon, Fanno-Shannon et Huffman, le codage le plus efficace pour chacune des deux sources ? Comparer leur efficacité.

3.2.3. Montrer que le codage de Huffman est optimal si les probabilités p_i des symboles à coder sont des puissances négatives de la valence V du codage.

On se propose de coder les symboles 2 par 2. Cela revient à définir une nouvelle source qui émet un bi-symbole. On applique cette transformation à la source S_A en supposant que la probabilité d'un bi-symbole est le produit de la probabilité des symboles le composant.

3.2.4. Donner pour cette nouvelle source S_A^2, le codage le plus efficace. Conclure sur l'efficacité du codage par bloc ?

4. CODAGES AVEC PREDICTION POUR LES IMAGES

4.1. Rappels

La procédure de modulation par impulsions codées différentielles (MICD) ou Differential Pulse Coding Modulation (DPCM), consiste à calculer d'abord la différence entre le signal d'entrée x et une valeur de prédiction p.

En pratique, p est une combinaison linéaire de x et de ses voisins. Différentes combinaisons existent pour déterminer la valeur de prédiction p. Si $x_{i,j}$ est le pixel considéré alors p peut être défini par x_{i-}

$_{1,j}$ ou $x_{i,j-1}$ (valeur du pixel précédent sur la même ligne ou sur la même colonne) ou par une les relations suivantes : $x_{i-1,j-1}$, $x_{i,j-1}+x_{i-1,j}-x_{i-1,j-1}$, $x_{i,j-1}+(x_{i-1,j}-x_{i-1,j-1})/2$, $x_{i-1,j}+(x_{i,j-1}-x_{i-1,j-1})/2$, $(x_{i,j-1}+x_{i-1,j})/2$

Cette valeur de prédiction est connue du décodeur. L'erreur « x–p » est ensuite quantifiée à l'aide d'une matrice de quantification et l'on obtient e_q. On code alors en mots binaires ces valeurs par indexation. On reconstruit simplement la valeur codée en ajoutant e_q à la valeur de prédiction.

Lors de la décomposition, la connaissance des pixels reconstruits permet de calculer les valeurs de prédiction d'indices supérieurs, et en leur ajoutant les erreurs de prédiction quantifiées, on reconstruit les pixels de l'image.

Exemple de table de quantification (optimisée par le CCETT)

Erreur de prédiction	Valeur quantifiée de e : e_q	Erreur de prédiction	Valeur quantifiée de e : e_q
$-255 \le e \le -70$	- 80	$9 \le e \le 18$	12
$-69 \le e \le -50$	- 58	$19 \le e \le 32$	25
$-49 \le e \le -33$	- 40	$33 \le e \le 47$	39
$-32 \le e \le -19$	- 25	$48 \le e \le 64$	55
$-18 \le e \le -9$	- 12	$65 \le e \le 83$	73
$-8 \le e \le -3$	- 4	$84 \le e \le 104$	93
$-2 \le e \le 2$	0	$105 \le e \le 127$	115
$3 \le e \le 8$	4	$128 \le e \le 255$	140

La forme la plus simple du codage prédictif est appelé « linéaire Modulation » ou Delta Modulation. Le prédicteur est une fonction dépendant simplement de la valeur de la donnée précédente, et on utilise un quantificateur sur 1 bit ce qui permet une représentation sur 1 bit du signal.

On peut également utiliser une DPCM bidimensionnelle, comme le fait JPEG en mode sans perte. L'erreur de prédiction n'est pas quantifiée.

4.2. Exercices

Si l'on considère que la valeur de prédiction pour les indices i,j est obtenue par $p = (x_{i,j-1} + x_{i-1,j}) / 2$ pour i et j >1. Si i (ou j) = 1 (1ère ligne ou colonne respectivement) $p = x_{i,j-1}$ (resp. $x_{i-1,j}$), et $x_{1,1}$ est transmis tel quel.

On utilise la table de quantification précédente, la transmission des indexes est codée sur 4 bits (au lieu de 8 pour les données).

Matrice originale

100	102	106	92
98	100	104	100
70	80	92	98
72	76	84	90

4.2.1. Calculer la matrice de prédiction avec les règles énoncées plus haut.

4.2.2. Calculer la matrice d'erreurs de prédiction.

4.2.3. Quantifier la matrice d'erreurs de prédiction à l'aide la table de quantification précédente. La transmission des indexes est codée sur 4 bits au lieu de 8 pour les données (il y a seulement 16 valeurs possible d'erreurs quantifiées).

4.2.4. Reconstruire l'image (en arrondissant à l'entier supérieur). Quelle est l'erreur moyenne ?

5. CODAGE EN GROUPE

5.1. Rappels

Le block truncation coding ou codage en groupe est une méthode très simple à mettre en œuvre qui opère sur des blocs d'images. On calcule pour chaque bloc la moyenne X et l'écart type σ. Ces deux valeurs sont transmises ainsi qu'une matrice de signes, qui indique pour chaque point du bloc s'il se trouve au dessus ou en dessous de la moyenne. La valeur reconstruite sera (X + σ) ou (X - σ) selon le cas.

Cette technique peut être améliorée en schématisant un quantificateur sur un bit. On calcule donc pour chaque bloc n × n = m, la moyenne et l'écart type.

$$\overline{X} = \frac{1}{m}\sum_{i,j} x_{i,j} \qquad \overline{X^2} = \frac{1}{m}\sum_{i,j} x_{i,j}^2 \qquad \sigma^2 = \overline{X^2} - \overline{X}^2$$

Pour une valeur de seuil particulière il attribue la valeur a (resp. b) pour le bloc reconstruit si la valeur correspondante du bloc d'origine est supérieure au seuil (resp. inférieure). Dans le cas particulier où le seuil est la moyenne, les valeurs de a et de b sont calculées en résolvant les équations suivantes :

$$m\overline{X} = (m-q)a + qb \qquad m\overline{X^2} = (m-q)a^2 + qb^2$$

Où q est le nombre de pixels du bloc supérieur au seuil, ce qui donne finalement :

$$a = \overline{X} - \sigma\sqrt{\frac{q}{(m-q)}} \qquad b = \overline{X} + \sigma\sqrt{\frac{(m-q)}{q}}$$

Alors chaque bloc est défini par les valeurs de la moyenne, de l'écart type et d'un plan de nxn bits constitué de 0 et de 1, indiquant si les pixels sont au dessus ou au dessous du seuil.

Cette méthode est utilisé pour la compression d'images en niveaux de gris et réduit le nombre de niveaux.

5.2. Exercices

Soit le bloc 4x4 suivant :

$$x_{i,j} = \begin{bmatrix} 121 & 114 & 56 & 47 \\ 37 & 200 & 247 & 255 \\ 16 & 0 & 12 & 169 \\ 43 & 5 & 7 & 251 \end{bmatrix}$$

5.2.1. Calculer la moyenne, l'écart type et q.

5.2.2. En déduire a et b (en arrondissant au plus proche), puis le plan de bits 4x4 et le bloc reconstruit.

5.2.3. Que préserve cette méthode ?

6. AUTRES CODAGES

6.1. Le codage à base de dictionnaires (Lempel-Ziv-Welch)

Initialisation : un dictionnaire est créé, typiquement avec tous les caractères du code ASCII.

Les caractères sont lus un par un (c) et stockés dans un buffer (B) :

- Si B+c est dans le dictionnaire alors B=B+c
- Sinon B est envoyé, B+c est ajouté dans le dictionnaire, B=c

A la fin le buffer est renvoyé s'il n'est pas vide (il correspond forcément à un mot du dictionnaire).

6.2. Exercices

6.2.1. Choisir un mot quelconque contenant des répétitions de groupes de lettres et le coder avec l'algorithme LZW (par exemple le mot « DAD DADA DADDY DO »).

6.2.2. Proposer un algorithme de décodage et l'appliquer.

6.2.3. Quel est l'inconvénient de cette méthode, comment y remédier ?

6.3. Le codage arithmétique

Le principe de cette méthode est de calculer les fréquences d'apparition des symboles puis d'associer à chaque symbole un intervalle réel compris entre 0 et 1.

Par exemple si on a la chaîne « abbc », on associe les intervalles suivants :

- a : fréquence ¼ : intervalle]0 ; 0.25[
- b : fréquence ½ : intervalle]0.25 ; 0.75[
- c : fréquence ¼ : intervalle]0.75 ; 1[

Par la suite on lit les lettres une par une et on encode :

a : intervalle]0 ; 0.25[, que l'on découpe en 4 : le premier quart pour a, les deux suivants pour b, le dernier pour c suivant le principe précédent.

b : intervalle]0.0625 ; 0.1875[que l'on découpe à nouveau en 4.

b : intervalle]0.09375 ; 0.15625[, etc.

Une fois terminé le traitement de la chaine, il suffit d'envoyer un réel quelconque dans l'intervalle.

6.4. Exercices

6.4.1. Encoder la chaine cbacbbba avec le codage arithmétique.

6.4.2. Proposer une méthode de décodage.

6.4.3. Quel(s) inconvénient(s) voyez-vous à cette méthode ?

TME 2

CONCEPTION DE FICHIERS SMIL

L'objectif de ce TME est de coder des petites applications SMIL pour mettre en évidence les différentes possibilités du langage ainsi qu'une application plus conséquente. Les fichiers SMIL sont notamment lisibles avec RealPlayer sous Windows.

1. CE QU'IL FAUT FAIRE

Vous devez au cours de ce TME créer des exemples de fichiers SMIL pour illustrer chacun des points principaux du langage d'une part, puis proposer un exemple qui intègre tous ces points. Les fichiers SMIL devront contenir des objets texte, images et vidéo (éventuellement du son dans la mesure où cela ne perturbe pas le déroulement du TME). Le choix des contenus est laissé libre, charge à vous de trouver sur Internet ceux dont vous aurez besoin.
Une page utile pour les attributs des différentes balises et bien plus :
http://service.real.com/help/library/guides/realone/ProductionGuide/HTML/realpgd.htm

1.1. Contenu de base

Un fichier SMIL contient en général des informations sur les différentes régions de l'espace dans lesquelles on peut disposer des objets, puis une description de l'animation elle-même contenant un scénario. Un exemple typique serait :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
"http://www.w3.org/2001/SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <layout type="text/smil-basic-layout">
      <root-layout id="rootLayout" width="640" height="480" backgroundColor="white"/>
      <region id="region_1" .../>
      <region id="region_2" .../>
    </layout>
  </head>
  <body id="body_mainSequence">
    ...
  </body>
</smil>
```

1.2. Insertion d'objets

Des objets peuvent être insérés dans le corps du document avec les balises :

```
<video region="reg1" src="test.avi" title="Vidéo" dur="6s"/>
<text region="reg2" src="test.txt" type="text/plain" begin="2s" title="Texte" dur="4s"/>
<img region="reg3"  src="test.jpg" dur="5" id="img_test">
```

1.3. Lecture en séquence

La lecture en séquence de plusieurs objets est possible avec l'attribut seq :

```
<seq>
  <img ...>
  <img ...>
</seq>
```

1.4. Lecture en parallèle

La lecture en parallèle de plusieurs objets est possible avec l'attribut par :

```
<par>
  <img ...>
  <text ...>
</par>
```

1.5. Transitions

Des transitions peuvent être ajoutées entre plusieurs objets mis en séquence. Dans l'exemple ci-dessous, on définit deux transitions, une pour faire un fondu au noir et l'autre pour un fondu à l'ouverture. Le code correspondant est placé dans l'entête après le layout :

```
<transition id="fromBlack" type="fade" subtype="fadeFromColor" fadeColor="black"
dur="1s"/>
<transition id="toBlack" type="fade" subtype="fadeToColor" fadeColor="black" dur="1s"/>
```

Par la suite on peut faire des transitions au début ou à la fin de la lecture d'un objet dans le corps du fichier :

```
<img  dur="3s" src="test.jpg" region="video_region" transIn="fromBlack" transOut="toBlack"
fill="remove"/>
```

1.6. Animations

Les objets peuvent être animés, des régions peuvent notamment être déplacées. Dans l'exemple ci-dessous, le coté gauche du bloc est déplacé de la position 0 à la position 100 sur un durée de 5 secondes, ce qui a pour effet de déplacer le bloc entier :

```
<animate attributeName="left" dur="5s" targetElement="region_1" values="0;100"/>
```

1.7. Effets supplémentaires et combinaison des possibilités

D'autres effets sont disponibles, notamment une gestion avancée des horloges pour séquencer des objets sur différents types d'événements (horloge, clavier, souris, etc.), pour faire du préchargement de contenu (peu utile quand tout est en local sur votre machine) ou des choix de différentes images/textes/sons/vidéos (switching) en fonctions de variables fixées par l'utilisateur (la langue dans le menu outils->préférences notamment).

Vous pouvez tester ces différentes approches et d'autres selon le temps disponible.

1.8. Site complet

Vous devez réaliser un site complet en SMIL. Par exemple cela peut consister à faire un site de visualisation de photos avec galeries. Tout autre choix utilisant pleinement les capacités de SMIL vues précédemment est satisfaisant.

TME 3

ETUDE DE PROTOCOLES P2P

L'objectif de ce TME est d'étudier des protocoles pair-à-pair uniquement à partir de traces d'utilisation et du logiciel Wireshark.

1. WIRESHARK/ETHEREAL

Pour analyser les informations circulant sur les réseaux, les administrateurs disposent de sniffers. Ces outils se présentent sous la forme d'équipements pouvant se connecter directement sur le réseau ou de logiciels installés sur un ordinateur relié au réseau à analyser.
Lorsque le réseau à étudier est à médium partagé — Ethernet par exemple — l'interface de toute machine connectée "voit" potentiellement tout le trafic échangé sur le réseau local. Pour ne pas juste "voir", mais "regarder" explicitement le trafic afin de récupérer toutes les trames qui circulent (y compris celles qui ne lui sont pas destinées) pour les analyser, un mode de "promiscuité" est habituellement disponible sur l'interface réseau. Ce mode ne perturbe ni le trafic du réseau, ni celui de la machine support, ce qui permet d'ajouter la fonction "sniffer" à cette dernière avec un logiciel adéquat.
Le logiciel WireShark[1] est un analyseur de protocoles (sniffer). Il peut utiliser directement l'interface de votre machine pour réaliser la capture de toutes les informations circulant sur le réseau local sur lequel vous êtes connecté[2]. Pour des raisons évidentes de sécurité, nous vous fournirons des captures déjà réalisées. Vous utiliserez alors la fonction principale de l'outil : l'analyse multiprotocolaire.

1.1 Introduction à WireShark

Sur une machine de la salle de T.M.E. sous le système GNU/Linux, accédez à votre compte utilisateur. Téléchargez une trace quelconque parmi celles mises à disposition.
Dans un terminal, exécutez la commande *wireshark* (et avec la commande *man wireshark* obtenez les informations de base). **Vous devez lancer l'application en mode « sans privilège ».**
Initialement l'écran est vide car aucune capture n'a été réalisée ou chargée. Cliquez sur le menu File et sélectionnez Open. Une fenêtre de sélection de fichier "Wireshark: Open Capture Files" apparaît.

- Sélectionnez un fichier de trace.
- Ne pas spécifier de filtres dans le champ Filter (nous y reviendrons plus loin).
- Désactivez les options MAC name resolution, Network name resolution et Transport name resolution. Cliquez finalement sur Ouvrir.

[1] WireShark est un logiciel libre. Il est disponible sur un grand nombre de plates-formes matérielles et systèmes d'exploitation (outre les machines à architecture i368 avec système GNU/Linux que vous utilisez actuellement). Vous pouvez le télécharger sur www.wireshark.org.
[2] L'interface Ethernet doit être configurée par WireShark dans le mode "promiscuous" afin d'accéder à tout le trafic diffusé sur le segment où votre machine est connectée. Pour changer de mode, l'application doit être exécutée avec les privilèges de l'administrateur.

L'interface de Wireshark est constituée de trois fenêtres principales (voir figure ci-dessous) :

1 - La fenêtre du haut liste les trames capturées et résume leurs caractéristiques. En cliquant sur l'une des trames dans cette fenêtre, le contenu de la trame apparaît en hexadécimal dans la fenêtre du bas, et le contenu de chaque champ est décrit dans la fenêtre du milieu. Ces trois fenêtres sont coordonnées (un clic dans l'une des fenêtres a un impact sur les deux autres).

2 - La fenêtre du milieu décrit précisément chaque champ de la trame. Ces champs, ainsi que diverses informations fournies par le logiciel, y sont présentés dans une structure arborescente. Chaque niveau est visualisable de manière indépendante.

3 - La fenêtre du bas est constituée essentiellement de la trame en hexadécimal. Les champs sélectionnés dans l'arbre de la fenêtre du milieu sont surlignés dans la trame. Les données sont présentées selon trois colonnes :

- La première colonne indique, sur 4 caractères hexadécimaux, le rang du premier octet de la ligne courante dans la trame ;

- La deuxième colonne affiche la valeur hexadécimale de 16 octets capturés (un octet est codé sur deux caractères hexadécimaux) ;

- La troisième colonne représente les caractères ASCII correspondant aux 16 octets de la seconde colonne (la correspondance n'est significative que lorsque du texte lisible se trouve encodé dans ces octets).

1.2 Filtres d'affichage WireShark

En plus des trois fenêtres décrites précédemment, il existe un champ fort intéressant, situé juste au dessus de la première fenêtre, et qui est le champ *filter*. Il sert à sélectionner certaines trames en fonction de champs particuliers (par exemple les adresses, les protocoles etc.). Après écriture du filtre, il est activé en cliquant sur le bouton *apply*. Ainsi, seules les trames autorisées par le filtre sont visibles dans les fenêtres. Un clic sur le bouton *clear* désactive le filtre d'affichage en cours d'utilisation.
La syntaxe d'un filtre simple est la suivante :

Nom_du_champ *Relation* *Valeur*

Exemple de filtre : vous ne voulez voir apparaître que les trames correspondant au protocole http. Pour cela, vous pouvez appliquer le filtre suivant : *tcp.port == 80*

Pour créer des filtres plus complexes, pour pouvez combiner plusieurs filtres simples grâce aux opérateurs logiques *and* et *or*. Pour vous faciliter l'écriture d'un filtre, cliquez sur l'onglet « expression » à droite de l'espace réservé à l'écriture du filtre. Vous aurez ainsi accès aux différents noms de champs utilisables dans les filtres ainsi qu'à des valeurs prédéfinies. Si vous êtes plus à l'aise, vous pouvez écrire directement le filtre dans l'espace réservé.

2.3 Décodage

Il arrive que WireShark n'interprète pas correctement des paquets d'un protocole qu'il est censé connaître. Pour pallier à ce problème il est possible de lui indiquer de manière explicite qu'un paquet donné est lié à un protocole spécifique. Dans la trace eDonkey par exemple, le paquet 9 est de type edonkey, un clic droit + 'decode as' permet de faire reconnaitre ce paquet et les suivants de la connexion TCP.

2. ANALYSE DE PROTOCOLES PAIR-A-PAIR

L'objectif du TME est de décrire le fonctionnement de protocoles P2P à partir de traces. Plusieurs traces seront fournies associées aux protocoles eDonkey, Bittorrent et Kademlia. Une étude complète de eDonkey est demandée, pour les autres et en fonction du temps une étude plus superficielle.

Il faut noter que les vieilles versions de WireShark décodent plus mal que la dernière, notamment pour le protocole Kademlia qui n'est vraiment supporté que depuis la version 0.99.8. Des fichiers texte contenant les traces sont disponibles au cas où, mais ils sont beaucoup moins simples à utiliser.

2.1 eDonkey

Le protocole eDonkey est un protocole semi-centralisé dans lequel les clients se connectent à un serveur principal en TCP qui stocke les associations entre les pairs et les fichiers qu'ils fournissent. Les échanges se font donc entre un client et le serveur pour tout ce qui concerne les publications ou demandes de fichiers et directement entre les pairs pour les échanges de fichiers.

Ouvrir la trace eDonkey puis chercher les paquets mal décodés (typiquement le paquet 10). La machine sur laquelle le client a été exécuté est 82.228.126.137. Vous devez identifier à partir de la trace :

- Les différents intervenants dans la suite d'échanges, notamment leur nom, adresses IP, ports utilisés, etc. Vous devez indiquer où vous avez trouvé ces informations dans la trace (numéro de paquet, couche protocolaire, etc).
- La suite des échanges en indiquant quels sont les intervenants et les objectifs des échanges.

Vous remarquerez certainement que certains paquets sont mal interprétés par WireShark ce qui limite la compréhension du fonctionnement global (tous les paquets unknown). N'hésitez pas à extrapoler sur l'utilité des paquets manquants.

2.2 Kademlia

Mêmes questions avec la trace fournie à condition que la version 0.99.8 de WireShark soit installée. Une trace texte est disponible si besoin.

Kademlia utilise une DHT donc la mise en œuvre n'est pas complètement visible dans la trace, beaucoup de choses se passant en local. Décrivez ce que pouvez voir dans la trace.

2.3 Bittorrent

Mêmes questions avec les deux traces fournies. Les principes de fonctionnement ont été vus en cours.

TME 4 / PROJET

CONCEPTION D'UN CODEUR JPEG : PREMIERES ETAPES

L'objectif de ce TME (sur 8 heures) est d'aborder plusieurs des étapes essentielles de la méthode de compression JPEG et de faire quelques rappels sur le codage de Huffman, les passages en mode fréquentiel ainsi que les différents modes de représentation des couleurs.

1. CE QU'IL FAUT FAIRE

Dans la suite on utilise le terme image pour désigner une matrice carrée contenant des valeurs entre 0 et 255. Chaque étape doit être soigneusement validée avec des cas concrets d'utilisation. Pour les tests, veillez à ne pas prendre les mêmes « images » que vos voisins.

Vous pouvez commencer en créant l'image de manière statique dans votre programme mais au final votre programme devra lire une image dans un fichier puis travailler dessus, vous devrez donc penser à créer cette fonction de lecture assez rapidement.

Enfin pour que tout soit plus simple, faites toutes vos sorties écran en format texte, ainsi écrivez plutôt la suite de caractères '9', '4' plutôt que le caractère ascii de code 94='^', ainsi vous pourrez lire directement vos résultats. De même si vous devez faire une sortie binaire, écrivez des suites de '0' et de '1'.

1.1. Transformée en cosinus discrète

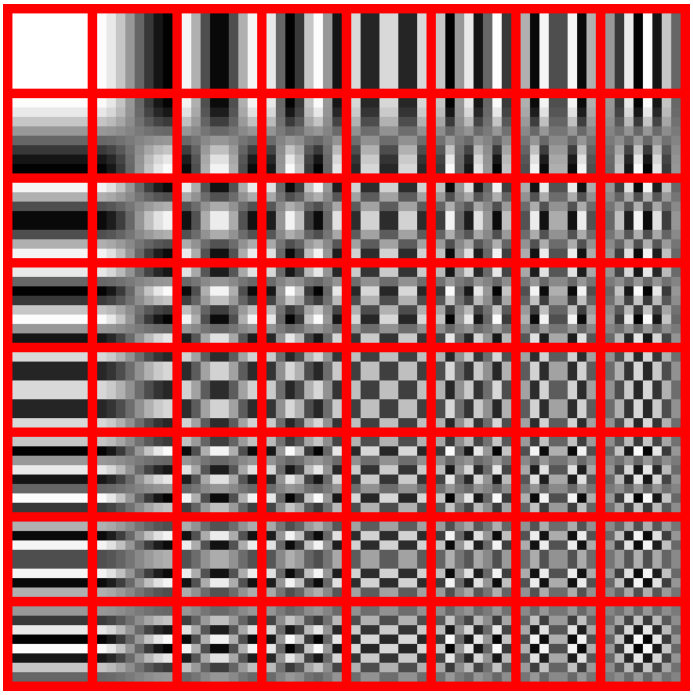
Etant donnée une matrice carrée NxN dont les coefficients sont les f(i,j). La DCT revient à calculer une matrice NxN à l'aide de la formule ci-dessous. La deuxième formule est la DCT inverse :

$$\begin{cases} F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} \\ f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} \end{cases} \qquad C(k) = \begin{cases} 1/\sqrt{2} \; si \;\; k = 0 \\ 1 \qquad si \;\; k \neq 0 \end{cases}$$

2.1.1. Programmez une DCT et une DCT inverse.

2.1.2. Que se passe-t-il si on calcule la DCT inverse de la DCT d'une image?

2.1.3. Calculez la DCT pour plusieurs images 8x8 du tableau ci-dessous (ou des images approchantes). Remarquez-vous quelque chose (l'image en haute qualité est visible sur (http://en.wikipedia.org/wiki/Image:Dctjpeg.png) ?



1.2. Méthode en zig-zag

1.2.1. Programmez une fonction qui à partir d'une matrice 8x8 retourne un vecteur contenant le parcours en zig-zag de la matrice (L1C1,L1C2,L2C1,L3C1,L2C2,L1C3,L1C4, etc.) et la fonction inverse qui à partir d'un vecteur retourne une matrice. Vérifiez que le zig-zag conjugué avec le zig-zag inverse donne bien la matrice originale.

1.2.2. Modifiez les fonctions pour qu'elles travaillent avec n'importe quelle matrice carrée.

1.3. Quantification

On propose plusieurs matrices de quantification :

- Q(i,j)=xi+yj+z (typiquement x=y=z=1)
- Q(1,1)=3, Q(2,1)=Q(1,2)=5, Q(3,1)=Q(2,2)=Q(1,3)=7, etc. avec éventuellement des coefficients multiplicatifs judicieusement choisis.

1.3.1. Programmez les opérations de quantification et de déquantification pour n'importe quelle matrice de quantification.

1.3.2. Comparez les différentes quantifications proposées sur différentes images auxquelles vous avez appliqué une DCT, lesquelles permettent d'avoir de longues suites de 0, quelle est la perte au niveau de l'image originale (comparez l'image originale et l'image originale + DCT + Quantification + Déquantification + Inverse DCT).

1.4. Méthode RLE (Run Length Encoding)

La méthode RLE consiste à transformer une suite de valeurs contenant de nombreux zéros par une suite de couples (nombre de zéros consécutifs, valeur non nulle suivante). En JPEG on code le nombre de zéros consécutifs sur 4 bits, soit des valeurs entre 0 et 15. S'il y a plus de 15 zéros consécutifs on peut utiliser le couple (15,0) pour signifier qu'il y a 16 zéros consécutifs. Ainsi une suite de 34 zéros consécutifs (34=16+16+2) suivie d'un 7 sera codée par (15,0) (15,0) (2,7).

En JPEG seuls les coefficients AC (toutes les valeurs sauf le coin supérieur gauche du bloc) sont codés en RLE.

1.4.1. Programmez la méthode RLE pour une séquence d'entiers de longueur quelconque et vérifiez que les résultats sont corrects notamment dans le cas où il y a plus de 16 zéros consécutifs.

Les valeurs sont toutes stockées en binaire, par exemple si une sortie RLE est (x,y) alors le contenu effectivement encodé sera (x sur 4 bits, nombre de bits pour stocker y, valeur de y en binaire) avec les conventions suivantes :

1 -1,1 (0,1)

2 -3,-2,2,3 (00,01,10,11)

3 -7,…,-4,4,…,7 (000,001,010,011,100,101,110,111)

4 -15,…,-8,8,…,15 (0000,0001,…)

…

Ainsi (le couple 2,8) sera stocké en pratique (2,4,1000), la valeur 2 étant stockée sur 4 bits et la valeur 4 sur 4 bits, soit 12 bits au total. Seul le premier octet (2,4) sera codé avec Huffman par la suite.

1.4.2. Programmez la méthode RLE en faisant une sortie binaire (suites de caractères '0' et '1' pour que ce soit lisible facilement).

1.5. Codage de Huffman

Voir le cours pour des rappels sur la méthode de Huffman.

1.5.1. Programmez le calcul de « l'entropie » pour les fréquences calculées à partir d'une suite de lettres fournie par l'utilisateur.

1.5.2. Programmez un codeur de Huffman qui, à partir d'une suite de lettres, va construire l'arbre, fournir un tableau de correspondance entre les symboles et les mots de code et afficher le message codé en binare. Testez votre méthode sur des exemples simples en indiquant le gain obtenu et la borne théorique.

1.5.3. Faites-en sorte que la valence du code soit un paramètre de votre codeur de Huffman : pour $V > 2$, si $r = (n–1) \bmod (V–1) \neq 0$ alors au premier passage rassembler r+1 symboles au lieu de V.

1.5.4. Appliquez votre codeur de Huffman sur une sortie de la méthode RLE (uniquement le premier octet de chaque triplet).

1.6. Evaluation globale

1.6.1. Comparez l'image originale et la sortie binaire finale après toutes les étapes. Quel est le gain en espace (en prenant uniquement en compte l'image et pas les tables de Huffman et autres).

1.6.2. En prenant en compte le stockage de la correspondance de Huffman, cela augmente la taille des données, proposez une méthode pour transmettre la table de correspondance sans surcout excessif.

1.7. Pour aller plus loin

Une image RGB contient trois couleurs pour chaque pixel. La première étape consiste à changer d'espace couleur (Y est la luminance, Cb et Cr sont la chrominance) :

$$\begin{cases} Y = 0{,}299R + 0{,}587G + 0{,}144B \\ Cb = -0{,}1687R - 0{,}3313G + 0{,}5B + 128 \\ Cr = 0{,}5R - 0{,}4187G - 0{,}0813B + 128 \end{cases}$$

Ce qui donne trois matrices distinctes pour chaque bloc. Ensuite on soustrait 128 à toutes les valeurs puis on applique un DCT, un parcours zig-zag et une quantification (éventuellement avec des matrices de quantification distinctes) aux trois matrices. Bien entendu une image réelle contient plus d'un bloc 8x8.

Un codeur JPEG distingue les coefficients DC (le premier en haut à gauche du bloc) des coefficients AC (les 63 autres). Les coefficients AC sont codés comme précédemment mais les coefficients DC sont calculés en différentiel et la suite de coefficients DC est encodée avec Huffman sur un principe similaire à ce que l'on a vu avec le RLE. Par exemple si la valeur DC d'un bloc vaut 41 ($41=DC_i-DC_{i-1}$) alors on a l'écrire : (6, 101001) car il faut six bits pour écrire 41 qui s'écrit 101001 en binaire. La suite de valeurs …, 6, … va aussi être codée avec Huffman. On va donc au final écrire pour un bloc :

Huffman(longueur DC_i-DC_{i-1}).Valeur(DC_i-DC_{i-1}).[Huffman(RLE(AC))]$_{1..63}$

1.7.1. Adaptez les fonctions pour gérer non pas un bloc 8x8 mais un ensemble de blocs pour image avec un seul canal couleur (1 octet par pixel). L'encodage doit être fait de manière complète en distinguant les coefficients AC et DC.

1.7.2. Programmez ou adaptez les fonctions pour gérer des images couleurs.

1.7.3. Evaluez l'encodeur final (taux de compression sur des images test), sans compter les tables.

1.7.4. Si vous avez encore du temps, vous pouvez chercher des informations supplémentaires sur Internet concernant le format de stockage exact des fichiers JPEG afin de fournir non pas une sortie texte mais une vraie sortie JPEG. Attention le format JPEG réel compresse très bien les grosses images mais une toute petite image peut être très volumineuse à cause de tout le contenu additionnel dans un fichier standard, notamment la définition des matrices de quantification, les tables de Huffman…