

CPW projet

Projet de cours ITI

Quentin Zeller, Etienne Guignard

Année 2017-2018

05.04.2017

Version 1.3

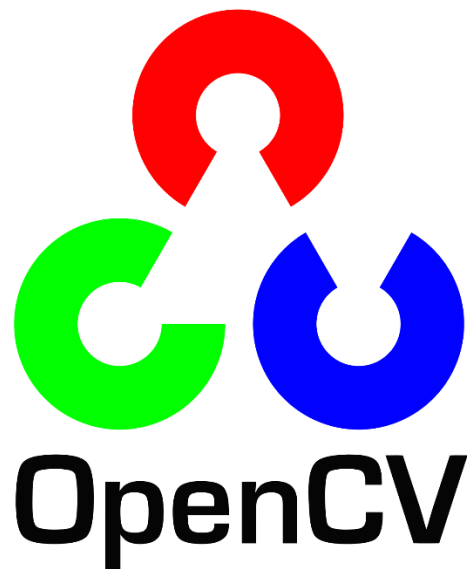


Table des matières

1	Introduction	4
1.1	Problématique	4
1.2	Objectifs	4
2	Analyse.....	5
2.1	OpenCV	5
2.2	Démarche.....	5
2.3	Méthodes.....	5
3	Réalisation.....	6
3.1	Environnement de développement.....	6
3.2	Prérequis.....	6
3.3	Extraction des scènes.....	6
3.3.1	Ffmpeg	6
3.3.2	Histogramme.....	7
3.4	Traitements intermédiaires	7
3.5	Méthodes.....	8
3.5.1	ORB.....	8
3.5.2	SSIM	9
3.5.3	Reconnaissance faciale	9
3.5.4	Histogramme.....	10
3.6	Librairie NetworkX	10
3.7	Cuda OpenCV	10
3.7.1	Compilation.....	10
3.7.2	Résultats.....	14
4	Discussion.....	15
4.1	Vidéos.....	15
4.2	Algorithmes.....	16
4.2.1	ORB.....	16
4.2.2	SSIM	17
4.2.3	Reconnaissance faciale	19
4.2.4	Histogramme.....	20
4.2.5	Autres méthodes.....	21
4.3	Graphes.....	21

4.3.1	Les communautés	21
4.3.2	Les cliques	23
5	Résultats.....	25
6	Conclusion.....	26
7	Références	27
8	Table des illustrations	27
9	Table des tableaux	28

1 Introduction

Cereal Partners Worldwide (CPW) est une joint-venture¹ qui combine l'expertise de deux sociétés : Nestlé et General Mills. Établie en 1990, CPW emploie 4600 personnes et fabrique la majorité des céréales pour le petit-déjeuner les plus populaires au monde. La société a donc une portée internationale, car elle vend ses produits aux clients dans plus de 130 pays. En outre, la société a pour objectif d'améliorer ses performances en analysant ses propres informations à partir des données qu'elle produit.

1.1 Problématique

Dans la société, l'équipe marketing est en charge de la création de spots publicitaires visant à faire la promotion des céréales produites par l'entreprise. En effet, comme leurs produits sont distribués dans plus d'une centaine de pays, il leur est souvent nécessaire de modifier le contenu de la vidéo de référence afin de correspondre aux différentes cultures. Ces adaptations entraînent par conséquent une multitude de vidéos associées. Ne disposant pas de connaissances suffisantes dans le domaine de l'analyse automatique de vidéo, ces vidéos ne sont pas triées en amont. En effet, cette opération de tri est effectuée par une personne, ce qui prend un temps considérable. Par conséquent, l'entreprise ne peut pas remplir correctement l'objectif visant à fournir des informations sur des données qu'elle produit.

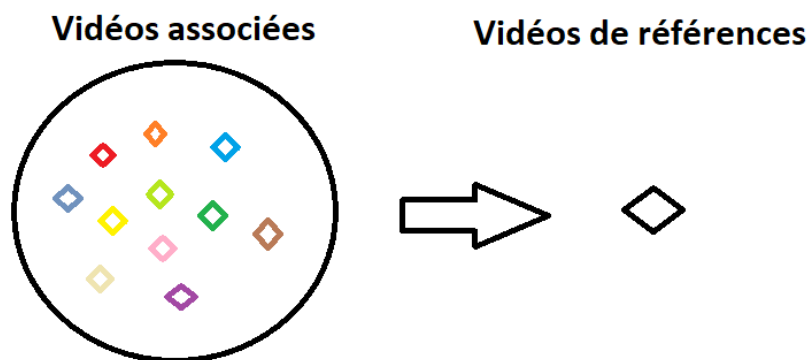


Figure 1: problématique CPW

1.2 Objectifs

L'objectif de ce projet est d'effectuer une analyse des différentes options qu'offre OpenCV pour effectuer l'analyse d'images et de vidéos afin de pouvoir répondre à la problématique posée par la société CPW.

¹ Filiale commune entre deux ou plusieurs entreprises dans le cadre d'une coopération économique internationale.

2 Analyse

Dans cette partie, nous allons décrire les différentes méthodes que nous avons employées afin de pouvoir répondre au mieux aux besoins de la société CPW. En outre, nous allons mettre en évidence la démarche employée pour arriver à ce résultat, soit regrouper les vidéos associées aux vidéos de référence.

2.1 OpenCV

Dans ce projet, il nous a été suggéré d'utiliser OpenCV (1) afin d'effectuer des traitements sur des fichiers vidéo et des images. En effet, OpenCV (Open Source Computer Vision Library) est publié sous une licence BSD et par conséquent, il est gratuit pour une utilisation académique ou commerciale. Il dispose d'interfaces C++, Python et Java et prend en charge Windows, Linux, Mac OS, iOS et Android. OpenCV a été conçu principalement pour des applications en temps réel. Enfin, écrit en C/C ++, OpenCV peut tirer parti du traitement CPU multicœur, mais peut aussi être compilé afin d'effectuer des traitements sur des GPU.

2.2 Démarche

Après avoir fait quelques recherches sur les possibilités qu'offrait OpenCV, nous avons décidé d'appliquer la démarche suivante :

- 1) Lecture de la vidéo et extraction des images impliquant un changement de scène
- 2) Traitement des images extraites (suppression des images noires/blanches, floues)
- 3) Utilisation d'algorithmes pour la comparaison de deux images afin d'obtenir une liste d'images concordantes.
- 4) Utilisation de fonctions en relation avec la théorie des graphes pour regrouper les images concordantes et ainsi regrouper les vidéos associées aux vidéos de références.
- 5) Interprétation et traitement des résultats mis en évidence par les fonctions de regroupement d'images.
- 6) Affichage des résultats sur la forme de listes ou/et d'arbres.

2.3 Méthodes

Après avoir défini la manière dont nous allons effectuer le traitement des vidéos et des images, nous avons effectué une recherche plus approfondie des différentes méthodes proposées par la librairie OpenCV. Nous en avons retenu quatre qui nous semblaient intéressantes et pertinentes :

- ORB
- SSIM
- Reconnaissance faciale
- Histogramme
- Autres méthodes

3 Réalisation

3.1 Environnement de développement

Pour le développement des deux applications, nous avons travaillé avec le même environnement tout au long du projet :

- Système d'exploitation : Windows 10 Pro 64x
- Environnement de développement : Pycharm professionnel édition
- OpenCV : Version 3.4 et contrib plus
- Langage : Python 3.6
- Versioning du code : Github <https://github.com/grzeller/OpenCV-Video-Comparison>

3.2 Prérequis

Pour exécuter les diverses méthodes ainsi que la fonctionnalités présentes dans le projet que nous avons réalisé, il est nécessaire d'installer plusieurs librairies :

- Télécharger OpenCV *contrib plus* pour Windows : <https://www.lfd.uci.edu/~gohlke/pythonlibs/>
- pip install "chemin fichier *contrib plus*"
- pip install scikit-image
- pip install Pillow

3.3 Extraction des scènes

Afin de pouvoir définir si une vidéo associée fait bien partie d'une vidéo de référence, nous avons décidé d'extraire les changements de scène pour chacune des vidéos. Nous avons utilisé les deux méthodes suivantes :

- ffmpeg
- OpenCV avec histogramme des couleurs

3.3.1 Ffmpeg

Avec ffmpeg, il suffit d'exécuter la commande (fig. 2), avec un seuil de détection spécifique pour que les scènes d'une vidéo soient extraites au format *png*. Dans certain cas, selon la qualité de la vidéo, aucune scène n'est extraite. Par conséquent, il est nécessaire d'abaisser le seuil de détection. Voici un exemple d'extraction des scènes :

```
ffmpeg.exe -i video.mpg -vf select='gt(scene\,0.4)',showinfo -vsync vfr scene_%d.png_
```







	scene_1.png	17.04.2018 23:42	PNG File	70 KB
	scene_2.png	17.04.2018 23:42	PNG File	137 KB
	scene_3.png	17.04.2018 23:42	PNG File	129 KB
	scene_4.png	17.04.2018 23:42	PNG File	40 KB
	scene_5.png	17.04.2018 23:42	PNG File	155 KB
	video.mpg	11.04.2018 21:49	VLC media file	2 030 KB

Figure 2: ffmpeg extraction

3.3.2 Histogramme

Pour découper les scènes de manière efficace et rapide, nous avons utilisé des histogrammes de couleurs. Nous comparons au fil du temps l'évolution d'histogramme. Si celui-ci diffère trop à un instant T , donc entre deux images, nous en déduisons qu'un événement important s'est produit au niveau de la vidéo, comme un changement de scène mais aussi un changement de plan. Par interpolation, le contenu des images suivantes est donc beaucoup plus intéressant, car il aura changé. Nous détectons les scènes par l'application d'une simple corrélation mathématique entre les différents histogrammes RGB. Pour pallier au problème des fondus, nous considérons plusieurs images dans le calcul de corrélation, en l'occurrence quatre dans notre algorithme. Après avoir découpé notre vidéo en fonction de ces événements que nous considérons comme des scènes, nous faisons une extraction d'image au milieu de ladite scène.

Par la suite, il serait intéressant d'améliorer cet algorithme en déterminant, par le biais d'autres algorithmes implémentés dans la partie suivante (cf. 3.4) que sont des algorithmes détectant la qualité d'images, si les images extraites sont bonnes ou non. Si les images ne sont pas bonnes, nous pourrions en choisir une qui ait une plus grande valeur graphique.

3.4 Traitements intermédiaires

Il y a environ 4000 scènes pour environ 580 vidéos, ce qui représente une quantité d'informations très importante à traiter. Durant le processus d'extraction, les changements de scènes sont parfois caractérisés par des images totalement noires ou blanches. Par conséquent, nous les éliminons pour éviter des problèmes de faux positif au moment de la comparaison entre les scènes. Voici la fonctionnalité qui permet d'effectuer cette opération :

```
is_black_or_white = sum(img.convert("L").getextrema()) in (0, 2)
```

Figure : noir et blanc code

Certaines vidéos de mauvaise qualité apparaissent floues, ce qui implique que les scènes extraites le sont aussi. La comparaison entre les scènes devient ainsi presque impossible. Par conséquent, nous avons décidé d'éliminer d'office les scènes floues et ainsi réduire le temps de traitement. Pour effectuer cette opération, nous nous sommes basés sur la méthode de *Laplacian*, disponible avec OpenCV, qui consiste à effectuer une détection de contours d'une image en noir et blanc. Dans le cas d'une image floue, plus la variance de *Laplacian* est grande, plus la photo sera de bonne qualité et inversement. Voici un exemple avec deux scènes qui viennent de la même vidéo :



Figure 3: flou exemple

Voici la fonction disponible dans la librairie d'OpenCV qui permet d'effectuer cette opération de détection de flou dans une image :

```
is_image_blurry = cv2.Laplacian(image, cv2.CV_64F).var()
```

Figure 4: floue code

3.5 Méthodes

Dans cette section seront décrites les méthodes que nous avons employé afin de détecter les similitudes entre les vidéos mais également entre les images extraites de ces vidéos.

3.5.1 ORB

ORB (2) (Oriented FAST and Rotated BRIEF) est la combinaison de deux algorithmes FAST (3) et BRIEF (4), avec toutefois quelques modifications pour en améliorer les performances. ORB est une bonne alternative à SIFT (5) et à SURF (6) qui sont des algorithmes brevetés et par conséquent, payants. Cet algorithme permet de détecter des points clé dans une images, indépendamment l'orientation ou e la taille. En utilisant un *matcher*, on peut comparer deux images et déterminer si des points correspondent. Si on trouve des similitudes, on récupère la distance statistique entre les deux points et ainsi cela détermine si la correspondance est bonne ou non. Plus la distance est petite, plus la correspondance est bonne et inversement. Voici un exemple de l'algorithme appliqué sur deux images de boîte de céréale.



Figure 5: ORB comparaison

Voici la fonction disponible dans la librairie d'OpenCV permettant d'effectuer cette opération de comparaison dans une image :

```
orb = cv2.ORB_create()
key_points_base, destination_base = orb.detectAndCompute(base, None)
key_points_compare, destination_compare = orb.detectAndCompute(compare, None)
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(destination_base, destination_compare)
```

Figure 6: ORB code

3.5.2 SSIM

SSIM est un algorithme ayant été développé pour mesurer la qualité visuelle d'une image compressée par rapport à l'image originale. L'idée de SSIM est de mesurer la similarité de structure entre les deux images, plutôt qu'une différence pixels à pixels. Comme ORB, l'algorithme fonctionne indépendamment de l'orientation ou de la taille de l'image.

3.5.3 Reconnaissance faciale

La librairie OpenCV dispose d'une fonctionnalité de détection de visages et de reconnaissance de visages qui peuvent être appliquée sur des images et des vidéos. Nous avons décidé d'utiliser ces deux méthodes combinées afin de trouver des visages correspondants dans les scènes. Après avoir fait plusieurs tests, nous avons remarqué que le résultat de la reconnaissance de visages ne comportait pas ou très peu de faux positif. L'opération de détection et de reconnaissance de visages se fait de la manière suivante :

- 1) Détection de visages dans toutes les images à l'aide d'un classificateur déjà pré-entraîné et utilisant la méthode basée sur les fonctions de Haar (7).
- 2) Extraction des visages détectés dans un fichier *png*.
- 3) Apprentissage à partir des visages détectés avec la fonction *LBPH* d'OpenCV et sauvegarde des résultats sous forme matricielle dans un fichier *yml*.
- 4) Nouvelle détection de visages dans toutes les images puis, comparaison avec les visages qui ont été appris.

Voici comment il faut procéder pour effectuer une détection et une extraction de visages dans un fichier image :

```
haar_face_cascade = cv2.CascadeClassifier(faces_cascade_path)
faces = haar_face_cascade.detectMultiScale(image, scaleFactor=1.2, minNeighbors=5)
for (x, y, w, h) in faces:
    cv2.imwrite(path, image[y:y + h, x:x + w])
```

Figure 7: détection de visage

Pour effectuer l'apprentissage des visages détectés et extraits, voici comment il faut procéder avec les fonctions disponibles dans la librairie OpenCV :

```
face_recognizer = cv2.face.LBPHFaceRecognizer_create()
face_recognizer.train(faces, np.array(labels))
face_recognizer.save('trainer.yml')
```

Figure 8: reconnaissance de visage code

Voici un exemple d'extraction de visages à partir d'un fichier correspondant à une scène de vidéo :



Figure 9: flou détection

3.5.4 Histogramme

La librairie OpenCV dispose d'une fonctionnalité qui calcule l'histogramme en trois dimensions d'une image. Par ce calcul, on définit un descripteur d'images qui représente la combinaison de la valeur des pixels rouges, verts et bleus. Dans un tableau, OpenCV stocke alors ce descripteur sous la forme d'un vecteur. L'objectif est d'indexer chaque descripteur dans un fichier pour pouvoir par la suite les comparer avec toutes les autres images et trouver les correspondances.

3.6 Librairie NetworkX

NetworkX permet de charger et stocker des réseaux dans des formats de données standards grâce aux structures de données fournies. La librairie permet aussi d'analyser la structure d'un réseau, de créer des modèles réseau et bien plus encore. Nous l'avons utilisé pour afficher graphiquement les relations entre les vidéos mais aussi pour stocker et charger les résultats obtenus par les différentes méthodes.

3.7 Cuda OpenCV

Avec l'utilisation d'un CPU standard, le temps de traitement de chacune des méthodes employées est considérable. Au vu des résultats obtenus en termes de durée de traitement, nous avons donc essayé d'utiliser une carte graphique possédant beaucoup plus de cœurs afin d'améliorer la vitesse de traitement. OpenCV avec Python n'est à la base pas prévu pour fonctionner avec la technologie Cuda de Nvidia. C'est pourquoi il est nécessaire de recompiler la librairie OpenCV avec Cuda et ainsi, l'associer à Python.

3.7.1 Compilation

La compilation d'OpenCV avec Cuda et Python nécessite d'installer plusieurs éléments qui sont entre autres liés au type de la carte graphique employée :

- Anaconda Python 3.6 ou Python 3.6
- NVIDIA CUDA toolkit 8.0
- Visual Studio 2015 avec les *Common Tools* pour Visual C++ 2015
- CMake 3.11

Après avoir installé tous les éléments, il faut aller sur le GitHub² officiel d'OpenCV afin de télécharger le code pour la compilation. CMake va permettre de générer un projet pouvant être utilisé par Visual Studio pour la compilation d'OpenCV.

Au démarrage de CMake, il faut spécifier un dossier vide où OpenCV sera compilé ainsi que le dossier des sources d'OpenCV préalablement téléchargé sur GitHub. De plus, il faut ajouter manuellement les deux entrées ci-dessous, car nous avons remarqué que le chemin de Cuda toolkit n'était pas détecté automatiquement :

- CUDA_BIN_PATH
- CUDA_TOOLKIT_ROOT_DIR

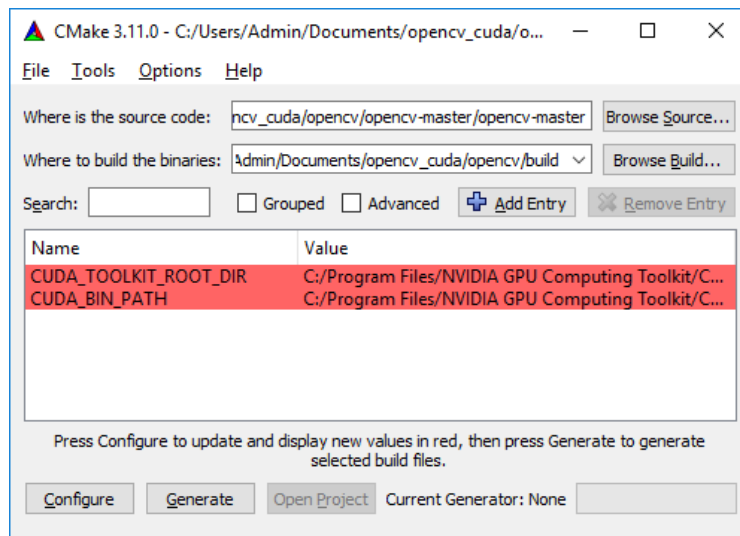


Figure 10: CMake configuration

² <https://github.com/opencv/opencv>

Après avoir ajouté ces deux entrées, il faut cliquer sur *Configure* (fig. 10), pour mettre en évidence les différents paramètres disponibles à la compilation et choisir la version de Visual Studio avec laquelle le projet va être généré.

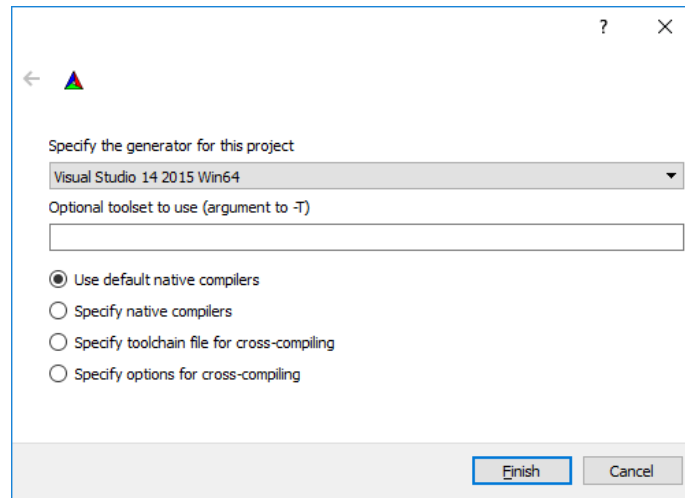


Figure 11: CMake génération

A la suite de ça, une multitude d'options concernant la compilation apparaissent. Voici donc les éléments (fig. 12 à 14) qu'il est important de sélectionner et de vérifier :

Cuda :



Figure 12: CMake option Cuda

Python:

PYTHON3_EXECUTABLE	C:/Users/Admin/Anaconda3/python.exe
PYTHON3_INCLUDE_DIR	C:/Users/Admin/Anaconda3/include
PYTHON3_INCLUDE_DIR2	
PYTHON3_LIBRARY	C:/Users/Admin/Anaconda3/libs/python36.lib
PYTHON3_LIBRARY_DEBUG	PYTHON_DEBUG_LIBRARY-NOTFOUND
PYTHON3_NUMPY_INCLUDE_...	C:/Users/Admin/Anaconda3/lib/site-packages/numpy/core/include
PYTHON3_PACKAGES_PATH	C:/Users/Admin/Anaconda3/Lib/site-packages

Figure 13: CMake Pyhton librairies

OpenCV :

BUILD_opencv_apps	<input checked="" type="checkbox"/>
BUILD_opencv_calib3d	<input checked="" type="checkbox"/>
BUILD_opencv_core	<input checked="" type="checkbox"/>
BUILD_opencv_dnn	<input checked="" type="checkbox"/>
BUILD_opencv_features2d	<input checked="" type="checkbox"/>
BUILD_opencv_flann	<input checked="" type="checkbox"/>
BUILD_opencv_highgui	<input checked="" type="checkbox"/>
BUILD_opencv_imgcodecs	<input checked="" type="checkbox"/>
BUILD_opencv_imgproc	<input checked="" type="checkbox"/>
BUILD_opencv_java_bindings_g...	<input checked="" type="checkbox"/>
BUILD_opencv_js	<input type="checkbox"/>
BUILD_opencv_ml	<input checked="" type="checkbox"/>
BUILD_opencv_objdetect	<input checked="" type="checkbox"/>
BUILD_opencv_photo	<input checked="" type="checkbox"/>
BUILD_opencv_python3	<input checked="" type="checkbox"/>
BUILD_opencv_python_binding...	<input checked="" type="checkbox"/>
BUILD_opencv_shape	<input checked="" type="checkbox"/>
BUILD_opencv_stitching	<input checked="" type="checkbox"/>
BUILD_opencv_superres	<input checked="" type="checkbox"/>
BUILD_opencv_ts	<input checked="" type="checkbox"/>
BUILD_opencv_video	<input checked="" type="checkbox"/>
BUILD_opencv_videoio	<input checked="" type="checkbox"/>
BUILD_opencv_videostab	<input checked="" type="checkbox"/>
BUILD_opencv_world	<input checked="" type="checkbox"/>

Figure 14: CMake OpenCV librairies

Après avoir vérifié que toutes les informations de la compilation sont bonnes, il faut cliquer sur *Generate* (fig. 15), et ainsi générer la solution avec Visual Studio. Si aucune erreur n'est survenue, voici ce que la console retourne :

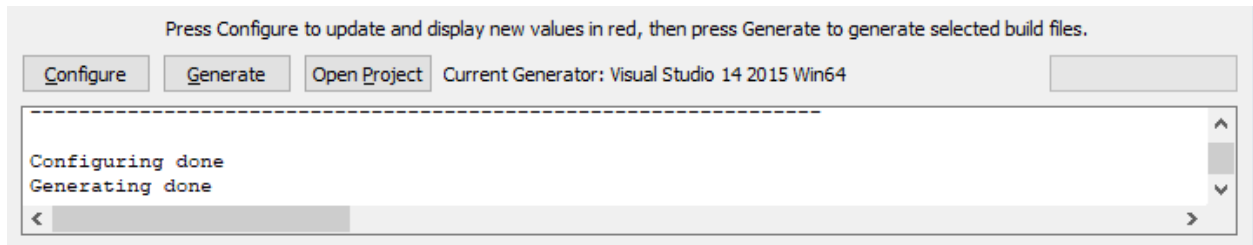


Figure 15: CMake résultat

Si aucune erreur ne survient, il faut ensuite cliquer sur *Open Project* (fig. 15), ce qui entraîne l'ouverture de Visual Studio.

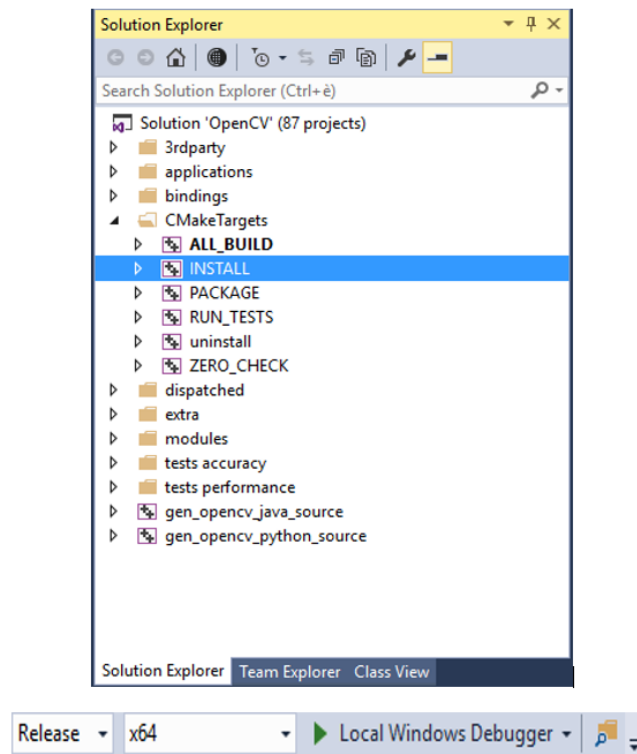


Figure 16: Visual Studio compilation

Il suffit alors de *Build* le projet *INSTALL* en faisant attention de le faire en mode *Release*. Il faut attendre environ 3 à 4 heures pour que la librairie soit entièrement compilée. Quand la compilation est terminée sans erreur, voici ce que retourne la console de Visual Studio (fig. 17) :

```
===== Build: 80 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Figure 17: Visual Studio résultat compilation

Pour vérifier l'association d'OpenCV à Python, il suffit de créer un fichier python, ajouter "import cv2" et enfin, le compiler avec Python. Si aucune erreur ne survient, alors l'opération s'est déroulée correctement.

3.7.2 Résultats

Nous n'avons malheureusement pas réussi à faire fonctionner OpenCV, combiné avec Cuda et Python, bien qu'aucune erreur ne soit survenue durant le processus de compilation. Nous n'avons pas eu le temps de chercher quelle était la source du problème.

4 Discussion

Cette partie rassemble les discussions au sujet des algorithmes implémentés ainsi qu'éventuellement quelques remarques sur les méthodes et procédés. Elle tient compte aussi des améliorations futures ainsi que des lacunes d'implémentation.

4.1 Vidéos

Premièrement, une discussion sur les vidéos et leurs contenus semble nécessaire pour bien comprendre toute la problématique. En effet, l'intitulé du projet, tel qu'il est formulé, ne rend pas compte de toute la difficulté que peut engendrer le contenu des vidéos.

Tout d'abord, parmi l'ensemble des 500 vidéos que nous avons eu à disposition, nous avons été confrontés au problème de la qualité. En effet, certaines vidéos, et pas uniquement quelques-unes, sont de qualité médiocre. Certaines semblent avoir été enregistrées en qualité VGA en provenance d'une source analogique. Elles peuvent contenir des artefacts, des bandes noires sur le haut et sur le bas de la vidéo mais peuvent aussi changer de ratio. Tous ces éléments rendent difficile toute comparaison et empêchent également, pour un certain nombre de vidéos, l'utilisation des techniques de reconnaissance nécessitant une préservation spatiale stricte.

Un autre problème est rapidement apparu : celui des éléments se retrouvant dans plusieurs vidéos. En effet, certaines scènes de films à propos de StarWars ou encore des Minions se retrouvent à l'identique dans des publicités alors la marque mise en avant n'est pas semblable. Ceci nous complique la tâche en nous empêchant notamment une recherche "naïve". D'autant plus que ces scènes ont une durée relativement longue.

On aperçoit que nous avons un problème de correspondance (fig. 18) entre les vidéos de céréales contenant des scènes du dessin animé, "Les Minions". En effet, nous avons d'un côté, à gauche de l'image (orange et bleu), des vidéos correspondant au clip CiniMinis et à droite, dans le plus gros cluster (rouge), des images correspondant aux vidéos Nesquik. Ces deux publicités différentes contiennent toute deux ces scènes. Toutefois, même si tous ces clusters (8) sont reliés, nous pouvons remarquer que l'algorithme de communauté a réussi à mettre en évidence les différents ensembles de vidéos. Nous remarquerons que les liens inter-clusters sont dû aux scènes des Minions. Les clusters se différencient par chance, car pas tous les pays décident d'ajouter cette scène. Il faudra donc tenir compte de ceci lors du développement de l'algorithme de reconnaissance et utiliser une méthode plus performante que celle-ci.

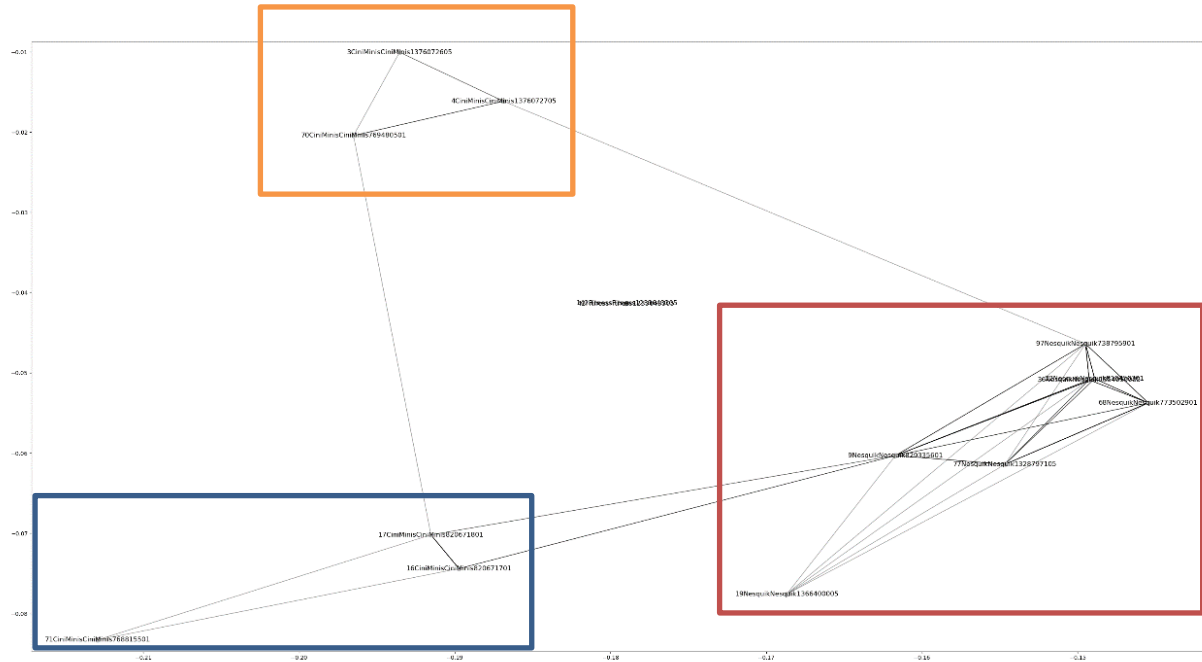


Figure 18: Matching des vidéos avec le plan "Gru - Minions"

4.2 Algorithmes

4.2.1 ORB

Le visuel de l'algorithme (fig. 19) de détection ORB (Oriented FAST and Rotated BRIEF), discuté dans la partie (cf. 3.5.1). Pour ce faire, ORB nous avons utilisé l'outil NetworkX (cf. 3.6) permettant de modéliser des graphes. Nous avons également utilisé l'algorithme de communauté de M. Louvain (cf. 4.3.1) dont nous discuterons plus loin. Celui-ci nous a permis de décomposer notre graphe en plusieurs parties, les parties fortement connexes devraient être isolées les unes des autres. Il permet aussi d'isoler des éléments de type clusters, même si ceux-ci sont connectés à un graphe principal. Nous pouvons constater que cette méthode est efficace principalement du point de vue visuel et fonctionne parfaitement comme preuve de concept. Différentes couleurs, niveaux de gris, sont associées aux clusters. Cependant, les positions des points ne nous permettent pas toujours de représenter correctement l'information. En effet, des agglomérats se forment au milieu du visuel quand bien même certains graphes sont disjoints.

Cet algorithme prouve son utilité. Cependant, pour que la détection soit efficace, il faudrait encore améliorer certains points dont le choix d'une méthode de détection de communauté plus efficace que celle de M. Louvain. Il faudrait aussi garder toutes les détections de l'algorithme ORB, car dans ce projet uniquement les distances de Hamming plus grande que 0.65 ont été gardées pour la construction du graphe. Il peut être utile de traiter ces valeurs en aval et y mettre des pondérations. Finalement, il faudrait mettre en place d'autres techniques moins "naïves". Beaucoup de concepts utilisant la théorie des graphes pourraient nous être utile mais il faut les implémenter. Dans ce travail, nous avons utilisé

l'algorithme "natif" à NetworkX permettant de détecter les cliques maximums. A noter que la détection de cliques est un problème NP-Complet³.

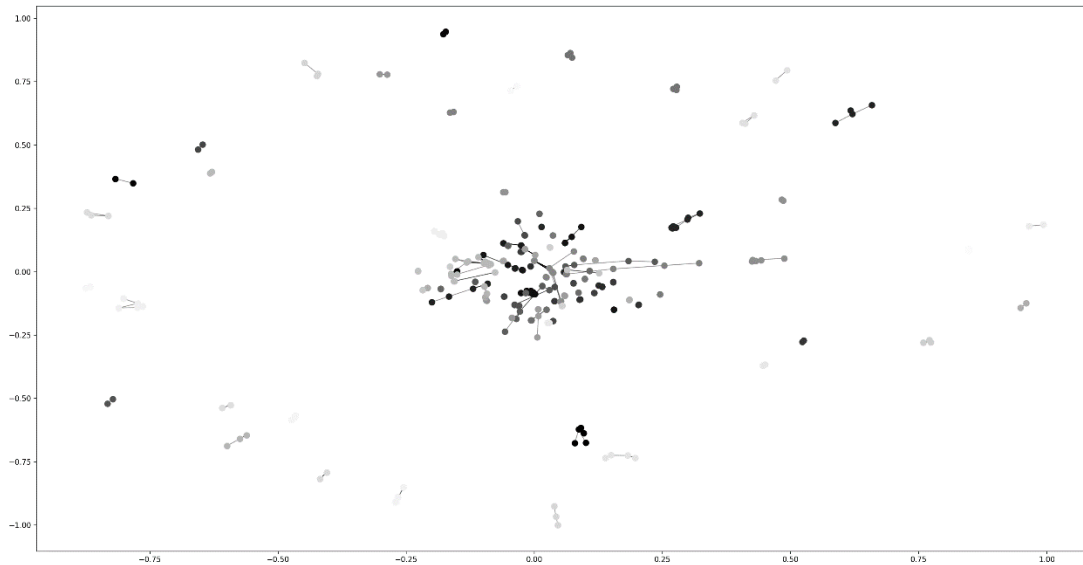


Figure 19: Graphe du calcul de l'ORB & la méthode de communauté de Louvain

4.2.2 SSIM

Le graphe de reconnaissance SSIM (Structural SIMilarity) permet de détecter une vraisemblance structurelle entre deux images et qui donc est flexible aux altérations. Pour ce faire, nous avons utilisé NetworkX qui permet de construire la logique d'un graphe de type nœud-arrête et l'algorithme de M.Louvain permettant la mise en évidence de communauté comme pour l'algorithme ORB (cf. 4.2.1). Nous remarquons également que cet algorithme est très prometteur. Les indices de similitudes que l'algorithme SSIM nous renvoie sont en pourcentage. Nous avons remarqué que seules les similitudes en dessus de 80 pourcents pouvaient nous intéresser et ne génèrent pas trop de faux positifs. En effet, cet algorithme permet de détecter aussi des éléments qui se rapprochent au niveau graphique, c'est-à-dire qu'un dessin animé à une ressemblance à un autre dessin animé, une plage a une ressemblance avec une autre plage. Ce qui ne nous intéresse pas dans ce cas. Nous avons exprès gardé un pourcentage assez laxiste ce qui permet le post traitement de l'information. Chaque vidéo possède une multitude de liens entre elles, tandis que les faux positifs sont isolés. Il est donc facile de les détecter. La première approche simple est de ne pas considérer les liens uniques entre vidéos. D'autres approches plus élaborées pourraient être mises en place notamment dans l'injection de poids dans les liens. Nous nous apercevons qu'un amas central se forme au milieu du visuel tout comme l'algorithme ORB. Il n'est pas constitué que d'un graphe connexe ce qui n'est pas voulu. Cependant, il est aussi dû au fait que certains faux positifs se profilent. Nous remarquons en analysant plus en détail que l'algorithme de détection de communauté fonctionne relativement bien. Nous en parlerons dans la section référente

³ NP-complet : 1. Vérification en temps polynomial (certificat). 2. Réduction polynomiale de tous les problèmes NP à celui-ci.

aux communautés (cf. 4.3.1). Nous avons pris l'exemple de cet algorithme et du graphe représenté dans le visuel ci-contre.

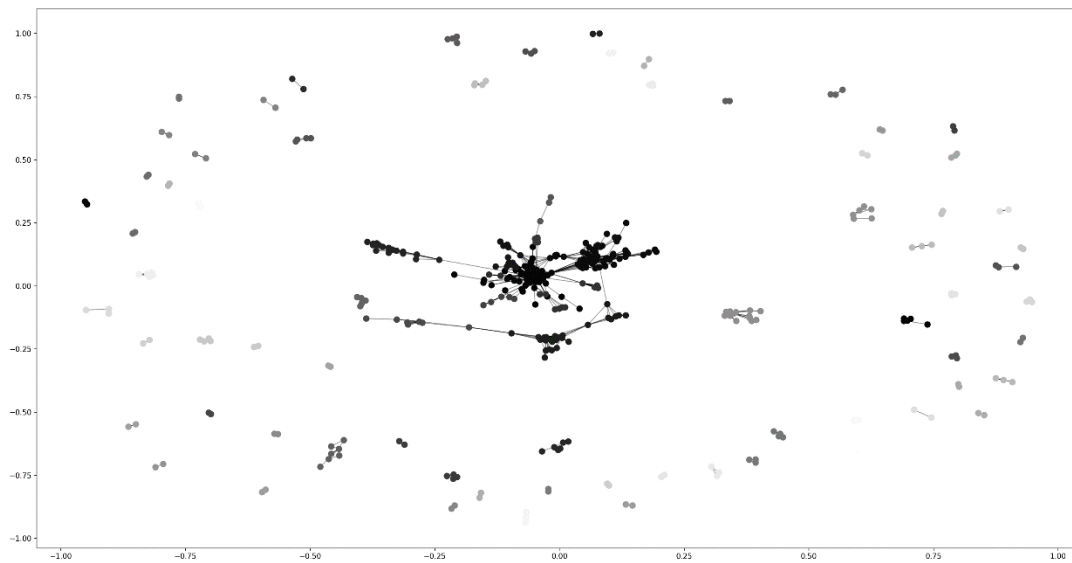


Figure 20: Graphe du calcul de l'algorithme ORB & L'algorithme de communauté Louvain

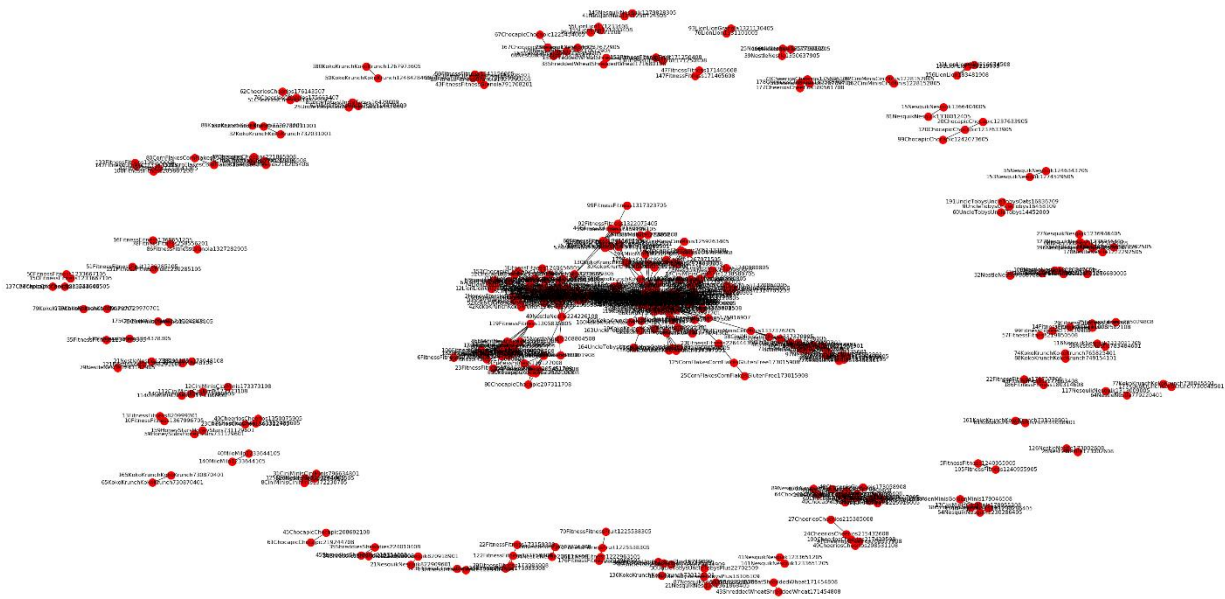


Figure 21 : Graphe du calcul de l'algorithme ORB & Affichage automatique favorisant les communautés

4.2.3 Reconnaissance faciale

Nous avons traité toutes les images avec un algorithme de reconnaissance faciale. Cette fonctionnalité est très développée dans l'industrie et des algorithmes très performants permettent la découverte des visages dans une image. Cet algorithme est très intéressant comparé au deux algorithmes précédents (ORB et SSIM), car il ne nécessite pas de comparaison entre deux images. Seule la reconnaissance image par image est nécessaire. Cela permet donc d'obtenir une base de données des images déjà calculées. Par exemple, à l'ajout de vidéos supplémentaires, il n'est pas nécessaire de recalculer tout l'ensemble. Cette méthode est aussi très intéressante, car il existe des modules hardware extrêmement rapides permettant la détection de visages. Les nouveaux téléphones portables en sont notamment munis.

Le graphe (fig. 22) nous montre toutes les détections qui ont été réalisées sur notre ensemble de vidéos, et plus particulièrement sur les extraits d'images à partir des scènes vidéo. Dans ce graphe, les liens de couleur bleu foncé sont des liens faibles et les liens de couleur claire sont des liens forts entre vidéos. Nous pensons cependant qu'il existe un problème dans l'algorithme de détection, surtout dans la partie de construction des liens. En effet, notre base de données ne comporte que des doublets de vidéos et non pas des triplets ou multipléts. Sur l'ensemble des vidéos identiques, il est quasi improbable que cela se produise. Cependant, nous pouvons prouver l'efficacité de ces doublets tout simplement par la lecture visuelle du graphe. Tous les doublets correspondent à des vidéos du même type. Il serait donc intéressant d'investiguer plus loin dans cette méthode. Il serait aussi possible d'analyser l'entièreté des vidéos et non pas uniquement des extraits de photos de scènes.

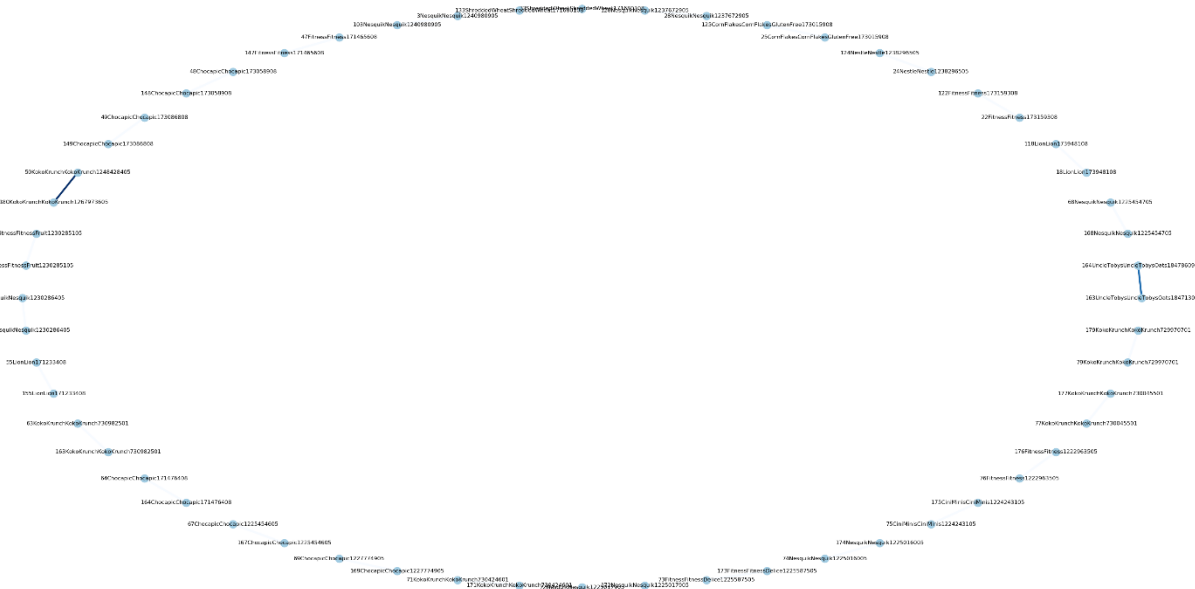


Figure 22 : Graphe circulaire du calcul de la reconnaissance faciale entre scène

4.2.4 Histogramme

Nous avons utilisé dans cet algorithme la fonction `histogram3D` d'OpenCV qui nous permet de comparer le spectre de couleurs entre deux images. Cette méthode est utile pour comparer deux images quasi similaires. De plus, elle est résistante aux changements de résolution ou aux déformations. Nous avons cependant eu un problème dans notre algorithme : uniquement les noms de vidéos sont pris en compte et pas les numéros de versions proposés dans l'ensemble des vidéos. Nous avons donc un résultat erroné (fig .23), où les points représentent à présent un sous-ensemble de vidéos de la même marque regroupés par la nomenclature des fichiers. Nous nous sommes rendus compte facilement de l'erreur, car ce visuel ne comporte pas beaucoup de nœuds et les images sont toutes reliées.

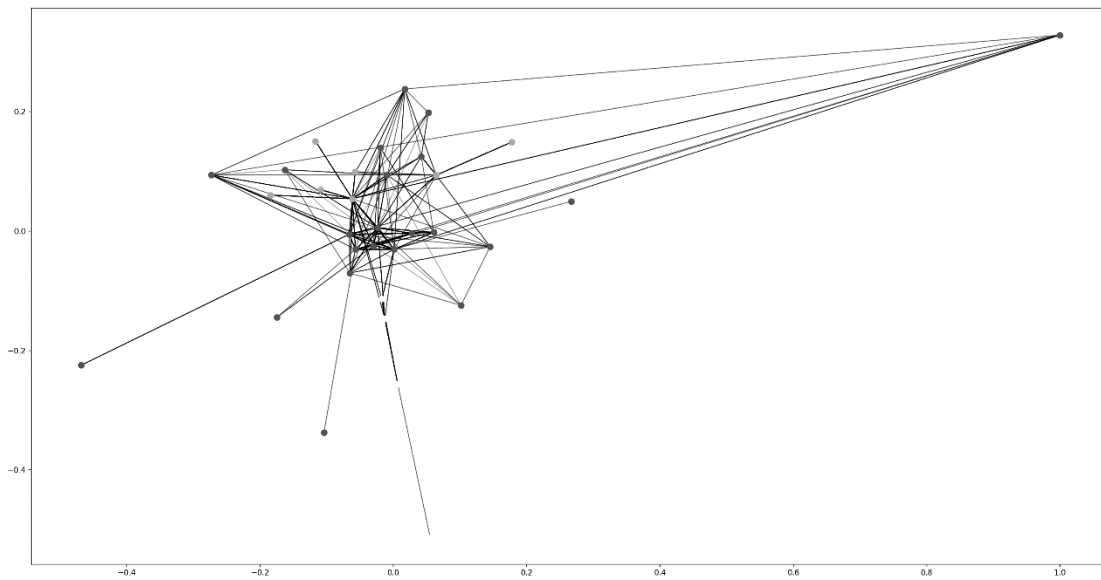


Figure 23 : Graphe du calcul des corrélations histogramme, erroné

4.2.5 Autres méthodes

Plusieurs autres méthodes, qui n'ont pas pu être implémentées, sont à investiguer :

- Analyse transversale des vidéos. C'est-à-dire analyser le spectre entier de la vidéo et comparer les distances avec les autres vidéos. Ici ce n'est plus une analyse discrète sur des images mais une analyse dans l'axe temporel. Nous pourrions imaginer un histogramme d'événements tels que la luminance, le changement de scène, ou même les changements audios.
- Approche de distances :
 - Dot-Matrix [Gibbs et al]
 - Similarité de Jaro-Winkler [Rémmi Auguste]
 - Lerenstein
 - Danerau-Lerenstein
 - LCS et Hamming (pour analyse de distances)
- DTW: Dynamic time wrapping
- SIFT: Scale-Invariant Feature Transform (! Propriétaire)
- MSE: Structural similarity (similaire à l'algorithme SSIM)
- Machine learning :
 - Reconnaissance d'objets
 - Analyse de contenu
 - Description de contenu
 - Utilisation des clouds, IBM, Google, Microsoft

4.3 Graphes

Nous avons jugé nécessaire de recomposer nos liens sous forme de graphes, ce qui nous permet de réutiliser la vaste théorie qui s'y réfère. Pour ce faire, nous avons utilisé la librairie NetworkX (cf. 3.6) disponible en Python qui est une librairie très répandue pour modéliser les graphes dans ce langage. Tous nos algorithmes utilisent des comparaisons symétriques pour l'instant. Nous avons donc utilisé des graphes non dirigés. Nous avons aussi plusieurs connections entre mêmes nœuds. Nous avons décidé de représenter ceci par des arcs différents et non pas par des attributs de ces mêmes arêtes. Il nous faut donc utiliser un modèle de graphe permettant une multitude d'arêtes entre deux mêmes nœuds. Dans NetworkX, cela s'appelle des MultiGraphs. Nous avons utilisé le moteur natif à Python pour afficher les visuels que l'on peut voir au fil de ce document. Malheureusement, ce moteur graphique ne prend pas en charge l'affichage des multiples arêtes entre les nœuds. Il faudrait donc utiliser une autre méthode d'affichage telle que celle consistant à différencier le nombre par la couleur ou une autre méthode de rendu.

4.3.1 Les communautés

Comme on peut le voir (fig. 24 et 25), un graphe s'il n'est pas proprement affiché est vite difficile à interpréter. La position des nœuds est primordiale pour que l'on puisse interpréter les liens, surtout quand le nombre de liens devient important. Il nous faut donc trouver un algorithme qui puisse reconnaître et interpréter les forces d'attraction qu'il y a entre nœuds. Nous avons, pour preuve de concept, utilisé l'algorithme "Louvain Community Detection" (8). Par la suite, il serait utile un algorithme beaucoup plus élaboré permettant de prendre en compte le nombre mais également le poids des liens présents entre les vidéos. Cependant, l'algorithme présenté dans ce projet nous donne déjà des résultats très satisfaisants.

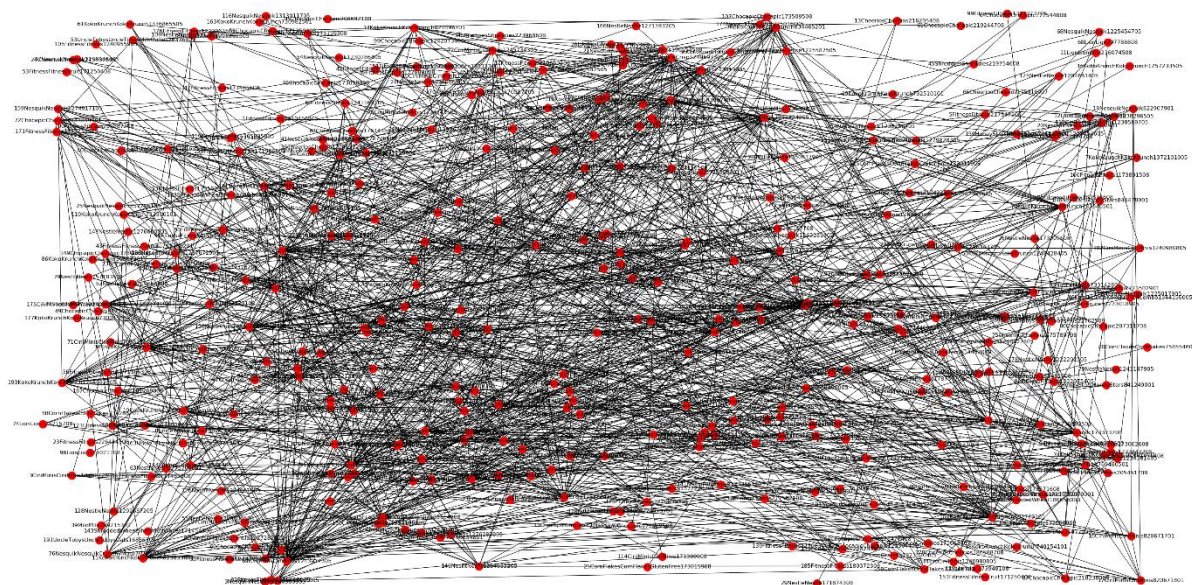


Figure 24: Graphe de la détection ORB sans algorithme de placement.

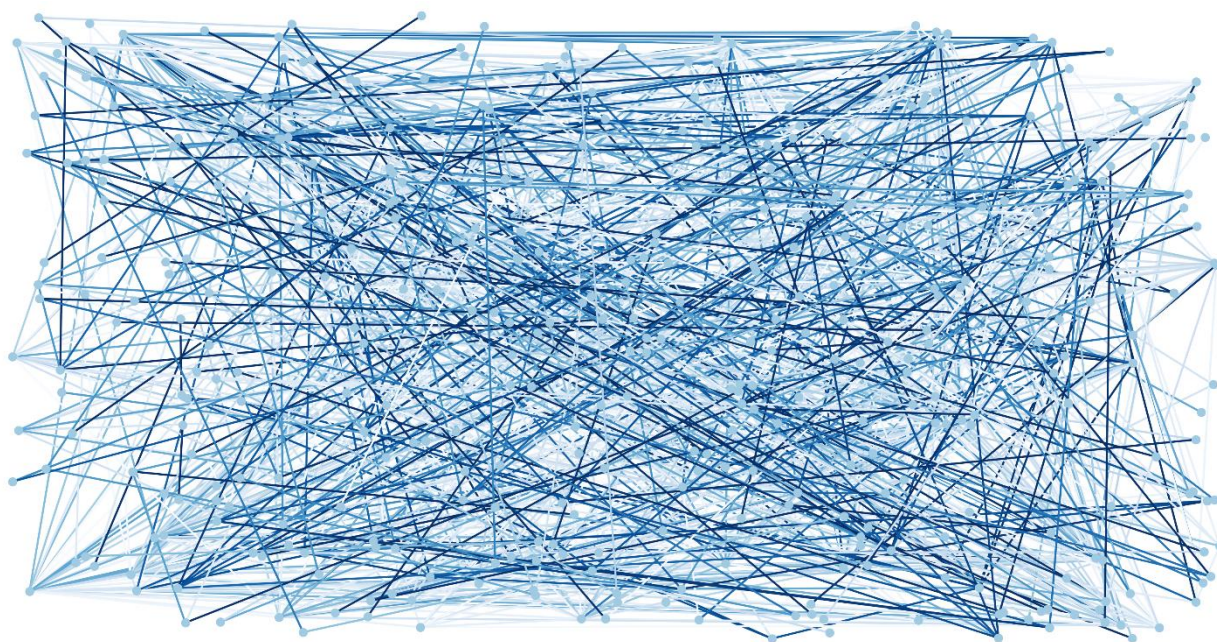


Figure 25 : Second graphe du calcul de l'ORB sans algorithme de placement. (Plus foncé, plus lié)

Nous voyons sous-ensemble (fig. 25) du graphes SSIM présentés dans la section SSIM (cf. 4.2.2). Celui-ci fait partie du gros sous-ensemble central. En analysant le visuel, il est possible de voir que l'algorithme de communauté a mis en évidence tout une partie du graphe bien que celui-ci soit connexe à un plus gros sous-ensemble. Des liens forts entre les différentes vidéos "Fitness" sont représentés et peuvent être traités pour les dissocier du groupe principal. La connections avec le groupe principal est dû à un lien faible et donc facilement identifiable. Ce lien est présent car les vidéos concernées possèdent des scènes similaires à la fin de la vidéo. Dans ce cas, une image de trois boites de cornflakes (Nesquik, Fitness, Chocapic) sont présente et donc font la correspondance entre vidéo.

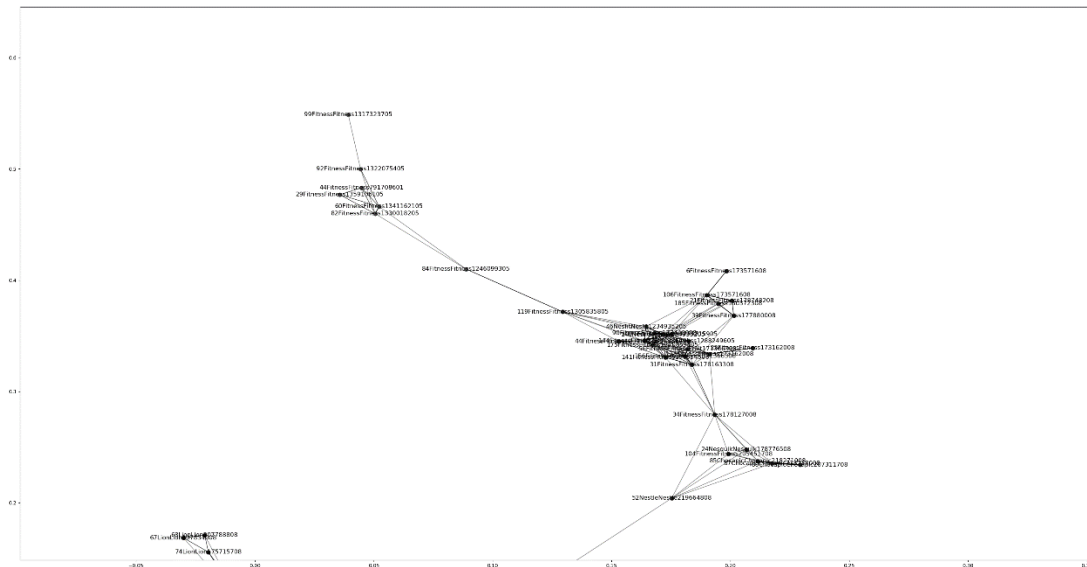


Figure 26 : Mise en évidence d'une communauté

4.3.2 Les cliques

Pour séparer les vidéos entre elles, nous avons utilisé un algorithme de détection de cliques. Il existe dans l'api NetworkX plusieurs fonctions de reconnaissance de cliques. Ce qui nous intéresse ici c'est de pouvoir identifier toutes les cliques maximales. Bien que ce problème soit NP-Complet, la relative simplicité du graphe nous permet d'avoir des résultats rapidement. Quelques secondes suffisent à trouver toutes les cliques maximales. Nous pouvons donc identifier facilement des ensembles de vidéos. Cependant, si des liens viennent à manquer dans des vidéos identiques ou de mêmes types, plusieurs cliques se forment. Il faudrait donc les regrouper algorithmiquement. L'avantage de cette méthode est que la probabilité qu'un sous-ensemble plus grand que trois nœuds ait un faux positif est faible, car le principe d'une clique est que tout élément de l'ensemble est connecté avec tous les autres. Ceci donne aussi de la robustesse aux vidéos auxquelles sont ajoutées des scènes promotionnelles, car elles ne seront a priori pas présentes dans toutes les vidéos de la même marque.


```
[ '140nesFitnessFit1234935205', '171FitnessFitness1271715905', '144FitnessFitness1234381405', '175FitnessFitness1270859505', '31FitnessFitness178163308', '156FitnessFitnessFruit171366308', '123FitnessFitness173162008',
'140nesFitnessFit1224935205', '171FitnessFitness1271715905', '144FitnessFitness1234381405', '175FitnessFitness1270859505', '31FitnessFitness178163308', '156FitnessFitnessFruit171366308', '123FitnessFitness173162008',
'140nesFitnessFit1234935205', '171FitnessFitness1271715905', '144FitnessFitness1234381405', '175FitnessFitness1270859505', '31FitnessFitness178163308', '156FitnessFitnessFruit171366308', '123FitnessFitness173162008',
'140nesFitnessFit1224935205', '40nesFitnessFit1224935205', '144FitnessFitness1234381405', '141FitnessFitness190554508', '119FitnessFitness1305835805', '156FitnessFitnessFruit171366308', '151FitnessFitness1288249605',
'140nesFitnessFit1224935205', '40nesFitnessFit1224935205', '144FitnessFitness1234381405', '119FitnessFitness1305835805',
'140nesFitnessFit1234935205', '56FitnessFitnessFruit171366308', '144FitnessFitness1234381405', '156FitnessFitnessFruit171366308', '141FitnessFitness190554508', '123FitnessFitness173162008',
'140nesFitnessFit1234935205', '56FitnessFitnessFruit171366308', '144FitnessFitness1234381405', '156FitnessFitnessFruit171366308', '151FitnessFitness1288249605',
'10CiniMinisCiniMinis1227128805', '183CiniMinisCiniMinis12668005', '69CiniMinisCiniMinis124397365', '110CiniMinisCiniMinis1237128095',
'4CiniMinisCiniMinis1376072705', '70CiniMinisCiniMinis769400501', '17CiniMinisCiniMinis820671801', '16CiniMinisCiniMinis820671701', '10CiniMinisCiniMinis824993901',
'4CiniMinisCiniMinis1376072705', '70CiniMinisCiniMinis769400501', '17CiniMinisCiniMinis820671801', '16CiniMinisCiniMinis820671701', '71CiniMinisCiniMinis768815501', '3CiniMinisCiniMinis1376072605',
'35FitnessFitness1234378305', '135FitnessFitness1234378305', '3CiniMinisCiniMinis1376072605',
'89UncleTobysUncleTobysPlus20694909', '23UncleTobysUncleTobysPlus16218809',
'9KokoKrunchKokoKrunch837367501', '41KokoKrunchKokoKrunch1347639705', '8KokoKrunchKokoKrunch830137401', '61KokoKrunchKokoKrunch1336865505', '9KokoKrunchKokoKrunch1371930105' ]
```

Figure 27: résultats

Cette image (fig. 27) a été choisie dans le but de montrer un extrait de la sortie terminale de notre algorithme de détection de cliques. Les vidéos sont représentées par un numéro et un nom de marque. Les clusters sont définis par des parenthèses. Ainsi donc (fig. 28), on définira un cluster de kokokrunch de 5 vidéos différentes.⁴

```
[ '5KokoKrunchKokoKrunch837367501', '41KokoKrunchKokoKrunch1347639705', '8KokoKrunchKokoKrunch830137401', '61KokoKrunchKokoKrunch1336865505', '9KokoKrunchKokoKrunch1371930105' ]
```

Figure 28: cluster kokokrunch

Cet extrait a été spécialement choisi pour montrer que l'algorithme fonctionne. D'autres extraits semblent moins concluants car les noms diffèrent mais sont pour certains correcte et font correspondance avec les scènes promotionnelles insérées dans les publicités. Il faudrait donc trouver une méthode pour dissocier ces deux phénomènes. Nous voyons également que différentes cliques se profilent pour des noms de marques de publicité identiques. Deux cas de figure sont possibles. Les vidéos sont de même marque mais ne sont pas identiques dans le contenu ou alors, l'algorithme n'a pas détecté de correspondance directe entre chacune des vidéos et plusieurs cliques se forment. La réponse semble se situer davantage au niveau du deuxième cas. En effet, nous voyons qu'une bonne partie des vidéos "fitness" sont présentes dans les cliques. Il en est de même pour les vidéos "CiniMinis". Nous en déduisons simplement qu'elles appartiennent tout compte fait à la même famille de vidéos et il serait simple dans ce cas de figure de les regrouper en post-traitement.

⁴ Nous avons gardé la nomenclature des vidéos tels que celle-ci nous a été fournie.

5 Résultats

Ce travail consistait d'une part à rechercher des solutions dans la comparaison de vidéos pour une question spécifique propre à CPW et d'autre part, à reconnaître que différentes publicités appartiennent à un sous-ensemble. Tout ceci en utilisant principalement les librairies OpenCV.

Plusieurs solutions ont été trouvées et expliquées dans ce document. Au vu du temps alloué à ce projet (environ 40 heures), nous avons décidé d'explorer le maximum de fonctionnalités et de laisser l'implémentation de nos algorithmes comme preuve de concept. Tous les algorithmes énoncés dans ces documents ont donc été testés et une procédure de leur utilisation a été mise en place. Pour que les solutions existantes soient pleinement fonctionnelles, nous estimons qu'il faudrait un temps de travail supérieur, au moins le double. Malgré ceci, nous avons pu isoler des ensembles de vidéos notamment en exécutant l'algorithme de reconnaissance de cliques sur les graphes générés par la reconnaissance ORB et SSIM. La reconnaissance des vidéos n'est bien entendu pas complète et comporte quelques faux positifs ainsi que des non-reconnaissances. Une solution de test et de quantification d'erreurs devrait être mis en place pour pouvoir calculer, valider et quantifier les résultats obtenus. Nous estimons tout de même que les résultats sont très bons et satisfaisants compte tenu de l'algorithme rudimentaire que nous avons développé. L'illustration dans la section (cf. 4.3.2) en témoigne, il est possible d'isoler passablement de vidéos et de les regrouper en fonction de leurs similarités.

Ensuite, nous pouvons voir sur le tableau ci-contre les différents temps d'exécution ainsi que la compatibilité des algorithmes de comparaison d'images avec le calcul GPU. Les temps d'exécution des opérations sur les graphes ne sont pas mentionnés car ils ne sont que de quelques secondes, les graphes n'étant probablement pas assez complexes pour poser problème à ce niveau-là. Nous avons également essayé de compiler la librairie OpenCV avec la compatibilité Cuda-Nvidia. Cependant, l'opération de compilation, dont le temps d'exécution est estimé à 4h, a échoué à multiples reprises. Nous n'avons donc plus eu le temps de poursuivre sur cette voie et tester les différents temps d'exécutions de nos algorithmes sur GPU. Nous avons cependant vérifié la compatibilité de ceux-ci.

Tableau 1 : Temps d'exécution et compatibilité GPU

<i>Temps d'exécution</i>	<i>Compatibilité GPU</i>	<i>CPU Xeon E3- 1231 V3</i>	<i>Intel i7-3930k</i>
<i>ORB</i>	OUI	13h	81h
<i>SIFT</i>	OUI (non OpenCV)	-	-
<i>SSIM</i>	OUI	17h	84h
<i>Histogrammes</i>	OUI	10h	15h
<i>Reconnaissance faciale</i>	OUI	10min	20min

Nvidia affirme que l'amélioration de la rapidité de calcul est de l'ordre de 30 à 60 fois pour les processeurs plus récents. En pratique, pour la détection de visages par exemple, l'augmentation de la rapidité de détection est généralement de l'ordre d'une dizaine de fois. Certaines puces récentes offrent cependant des modules hardware spécifiques à la détection de visages.

Nous avons ci-contre le tableau de synthèse du nombre de cliques trouvées sur nos deux algorithmes principaux. Pour comprendre ces valeurs, il faut souligner que le seuil appliqué pour l'algorithme SSIM est beaucoup plus laxiste que celui appliqué pour l'algorithme ORB. Le taux de faux positifs est donc bien représentatif. Nous gagnons pour le cas de l'algorithme SSIM une efficacité dans la reconnaissance de grandes cliques. En effet, les faux positifs sont en général singuliers et ne sont donc pas pris en compte dans les cliques. Nous donnons ainsi la possibilité à certaines vidéos où un lien pourrait manquer (faux négatif) de pouvoir s'intégrer plus facilement. De même, nous donnons donc ainsi de la valeur aux grandes cliques et donc validons des plus grands ensembles de vidéos.

Au contraire, dans l'algorithme ORB, nous nous assurons de la validité du résultat pour tous les ensembles avec comme défaut d'avoir des mises en évidence de famille de vidéo plus petites. Il serait intéressant d'investiguer un mélange de ces deux méthodes. Pour résumer, nous voyons qu'avec quelques algorithmes et un peu de savoir-faire au niveau de la théorie des graphes, il est possible de ressortir beaucoup d'informations et de caractériser la plupart des vidéos.

Tableau 2 : Statistiques sur les ensembles de cliques.

	<i>Nombre de clusters</i>	<i>Taille moyenne</i>	<i>Longueur maximale</i>	<i>Seuil appliqué</i>	<i>Appréciation taux faux positifs</i>
<i>SSIM</i>	248	4.6	17	80%	Bon (5% à 10%)
<i>ORB</i>	159	2.3	6	65%	Bon (1% à 5%)

6 Conclusion

Pour conclure, dans ce travail, nous avons pu expérimenter et réfléchir à différentes méthodes permettant la comparaison d'éléments multimédias. L'horizon de recherche est très vaste et peut constituer le travail d'une vie. Beaucoup d'études et d'outils sont existents, notamment au niveau du *machine learning*, mais nécessitent des ressources computationnelles beaucoup trop importantes pour nos ensembles de données. De plus, les prérequis en *machine learning* sont d'un niveau supérieur et concernent une discipline à part entière. Nous avons donc développé d'autres méthodes plus efficaces en ressources. En discrétisant nos vidéos en images, il nous a été possible de faire des comparaisons plus simples à gérer tant au niveau logique qu'au niveau mémoire et ressource informatique. Ce travail de recherche a été très instructif car les outils et concepts utilisés ne se limitent pas uniquement à la comparaison de similitudes de vidéos mais concernent tout problème nécessitant OpenCV, les graphes, ainsi que la recherche de distance entre objets informatiques.

7 Références

1. OpenCV introduction. [En ligne] <https://opencv.org/>.
2. ORB introduction. [En ligne] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.
3. FAST introduction. [En ligne]
https://fr.wikipedia.org/wiki/Features_from_Accelerated_Segment_Test.
4. BRIEF introduction. [En ligne] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html.
5. SIFT introduction. [En ligne] https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html.
6. SURF introduction. [En ligne] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html.
7. HAAR introduction. [En ligne]
https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html.
8. Cluster introduction. [En ligne] https://en.wikipedia.org/wiki/Computer_cluster.
9. Louvain, Pierre. Louvain Community Detection. *Github*. [En ligne] 2015.
<https://github.com/taynaud/python-louvain>.

8 Table des illustrations

Figure 1: problématique CPW	4
Figure 2: ffmpeg extraction.....	6
Figure 3: flou exemple	7
Figure 4: floue code	8
Figure 5: ORB comparaison.....	8
Figure 6: ORB code.....	8
Figure 7: détection de visage	9
Figure 8: reconnaissance de visage code.....	9
Figure 9: flou détection.....	10
Figure 10: CMake configuration.....	11
Figure 11: CMake génération.....	12
Figure 12: CMake option Cuda.....	12
Figure 13: CMake Python librairies.....	12
Figure 14: CMake OpenCV librairies	13
Figure 15: CMake résultat.....	13
Figure 16: Visual Studio compilation	14
Figure 17: Visual Studio résultat compilation	14
Figure 18: Matching des vidéos avec le plan "Gru - Minions"	16
Figure 19: Graphe du calcul de l'ORB & la méthode de communauté de Louvain.....	17

Figure 20: Graphe du calcul de l'algorithme ORB & L'algorithme de communauté Louvain	18
Figure 21 : Graphe du calcul de l'algorithme ORB & Affichage automatique favorisant les communautés	18
Figure 22 : Graphe circulaire du calcul de la reconnaissance faciale entre scène.....	19
Figure 23 : Graphe du calcul des corrélation histogramme, erroné.....	20
Figure 24: Graphe de la détection ORB sans algorithme de placement.....	22
Figure 25 : Second graphe du calcul de l'ORB sans algorithme de placement. (Plus foncé, plus lié).....	22
Figure 26 : Mise en évidence d'une communauté.....	23
Figure 27: résultats	24
Figure 28: cluster kokokrunch.....	24

9 Table des tableaux

Tableau 1 : Temps d'execution et compatibilité GPU.....	25
Tableau 2 : Statistiques sur les ensembles de cliques.	26