École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2011 - 2012

Rapport de Projet de Fin d'Etudes

Spotting automatique des logos dans les images de documents

Encadrants
Muhammed Muzzamil LUQMAN
muhammadmuzzamil.luqman@etu.univ-tours.fr
Jean-Yves RAMEL
ramel@univ-tours.fr
Dimosthenis Karatzas
dimos@cvc.uab.es

Étudiant
Zakaria MELK
zakaria.melk@etu.univ-tours.fr

DI5 2011 - 2012

Université François-Rabelais, Tours

Version du 6 mai 2012

Table des matières

Table des figures

Liste des tableaux

Remerciements

Tout d'abord, je tiens à remercier mon encadrant M. Muhammed muzzamil LUQMAN, pour son soutien, ses conseils pertinents et sa disponibilité tout au long du projet malgré ses occupations, et pour l'occasion, j'en profite pour le féliciter de son travail remarquable et la mention « Très honorable » qu'il l'a eu lors de sa soutenance de thèse.

De même, j'aimerais exprimer ma gratitude à mon deuxième encadrant M. Jean-Yves RAMEL, professeur à l'Université François Rabelais de Tours, de ses travaux de recherche intéressants, notamment l'outil de vectorisation « VectoGraph » qui représente une phase capitale dans la réalisation de ce projet. Ainsi qu'un grand merci à M. Dimosthenis Karatzas, chercheur à « Computer Vision Centre » de nous faire profiter de ses connaissances dans le traitement d'images afin que je nous puissions améliorer les résultats obtenus.

Je remercie également, M. Mathieu DELALANDRE et M. Nicolas RAGOT ainsi que toute l'équipe RFAI pour leurs contributions et leurs remarques constructives lors de ma soutenance de mi-parcours ainsi de m'avoir fourni les outils nécessaires pour la réalisation de ce projet dans les meilleures conditions.

Je souhaite autant remercier Mme Betty VIAS, responsable bibliothèque à Polytech'tours au département informatique pour ses motivations, les ressources documentaires, et aussi pour le maintien d'un environnement de travail convenable.

Enfin, je tiens à remercier mes parents et ma famille qui m'ont offert le soutien moral et financier tout au long de mes années d'études au Maroc comme en France et pour lesquels je dédie ce travail.

Introduction

Ce projet de fin d'études est inscrit dans le cadre des enseignements proposés en dernière année cycle d'ingénieur au département informatique de Polytech'Tours, pendant lequel nous nous sommes amenés à étudier une problématique, qu'elle soit du domaine de la recherche ou du monde professionnel, de mettre en pratique nos connaissances techniques et théoriques afin de proposer une solution.

Puisqu'il s'agit d'un travail individuel, cela nous a permis de passer par les différentes phases de la réalisation d'un projet informatique, en commençant par une phase de spécification système, suivie d'une spécification logicielle, ensuite le codage, et finalement une phase de tests.

Mo projet de fin d'études relève du domaine d'analyse d'images et reconnaissance des formes, il vise à concevoir et développer un moteur de localisation des logos dans les images de documents. Le sujet et proposé par deux organismes, L'équipe « Reconnaissance des Formes et Analyses d'Images » représentée par : Mr. Muhammed Muzzamil LUQMAN et le Professeur Jean-Yves RAMEL, ainsi que le centre de recherche « Computer Vision Centre » de Barcelone représenté par le Dr. Dimosthenis Karatzas.

Ce rapport est structuré en 6 chapitres :

- Le 1er chapitre est dédié à la présentation générale du projet en ce qui concerne ses objectifs, son contexte, ainsi qu'une étude de l'existant.
- Le 2^{e} chapitre présente en détail le travail réalisé pour la conception et les spécifications de la partie « Apprentissage » et également pour la partie « Localisation » du système.
- Le 3^{e} chapitre décrit l'architecture logicielle et les solutions techniques adoptées pour le développement.
- Le 4^{e} chapitre présente la gestion du projet.
- Le 5^{e} chapitre liste les problèmes rencontrés et les solutions trouvées.
- Le 6^{e} chapitre est consacré aux perspectives qui peuvent améliorer le système développé.

Présentation du projet

3.1 Objectifs du projet

L'objectif de ce PFE est le développement d'un moteur de localisation de logos adapté aux bases d'images de documents graphiques en se basant sur un outil nommé « VectoGraph », développé par J.Y Ramel, qui permet la transformation d'une image bitmap monochrome en un graphe correspondant en utilisant un algorithme de vectorisation.

Ce moteur est une application web basée sur une architecture client/serveur qui permet l'interaction entre les algorithmes implémentés ainsi que d'autres applications externes afin de fournir à l'utilisateur final une interface graphique sous forme de page web permettant de localiser un logo dans une base d'image « Tabacco800 Complex Document Image Database and Groundtruth » [1].

3.2 Contexte du projet

La partie majeure de ce projet se base sur l'encapsulation d'image dans des signatures en exploitant les graphes issus de la vectorisation, les étapes permettant d'effectuer ce traitement sont étudiées dans le cadre de la thèse de MM. Luqman .
Cette représentation numérique de l'image tire son importance de son optimalité au niveau de stockage et surtout la facilité de recherche dans le contenu afin de localiser les zones d'intérêt.

3.3 Acteurs

Maitrise d'ouvrage :

- Mr. Muhammed Muzzamil LUQMAN : ATER en informatique à l'Université François Rabelais de Tours
- Mr. Dimosthenis Karatzas : Chercheur à « Computer Vision Centre », Université Autónoma de Barcelone
- Mr. Jean-Yves RAMEL : Professeur à l'Université François Rabelais de Tours

Maitrise d'œuvre :

- Zakaria MELK : Élève ingénieur à Polytech'Tours

3.4 Etude de l'existant

L'idée derrière ce projet est le développement d'un système de localisation des zones d'intérêt dans les images de documents en utilisant qu'un seul langage qui est Java/J2EE, sauf pour la partie vectorisation quant à elle utilise l'outil « VectoGraph » qui est développé en C++.

Les fonctionnalités ont été conçues et développées en se basant uniquement sur la documentation fournie par mon encadrant et plus précisément ses articles « *Fuzzy Multilevel Graph Embedding* » [2], « *Recherche de sous-graphe par encapsulation explicite de graphes : Application à la localisation de contenu dans les images de documents graphiques* » [3], « *A Content Spotting System For Line Drawing Graphic Document Images* » [4], « *Vers une approche floue d'encapsulation de graphes : application à la reconnaissance de symboles* » [5], ainsi que les deux rapports de PFE « *Content Spotting : recherche de contenu dans les images de documents* » [6] et « *Apprentissage pour détection des régions d'intérêt dans les graphes* » [7]

3.4.1 VectoGraph

Principe

VectoGraph est un logiciel en C++ développé par [J.Y.Ramel RFAI, LI] permettant à partir d'une image « bmp monochrome » de donner une représentation sous forme d'un graphe en passant par une étape de vectorisation. Cette étape consiste à transformer le contour des formes contenues dans l'image en une chaîne de vecteurs tout en minimisant la perte d'information afin d'avoir une approximation fidèle au dessin, ensuite une étape d'appariement qui reliera les vecteurs obtenus pour donner des quadrilatères, chaque quadrilatère représente un trait dans l'image et possède des attributs comme la longueur de son axe médian, les angles des deux vecteurs constituants, etc. Au final, une étape de construction d'un graphe dont les sommets sont les quadrilatères, et la relation entre ces derniers sont représentées par des arcs pondérés.
La figure 3.1 montre la phase de la vectorisation suivie d'une représentation sous forme d'un graphe :

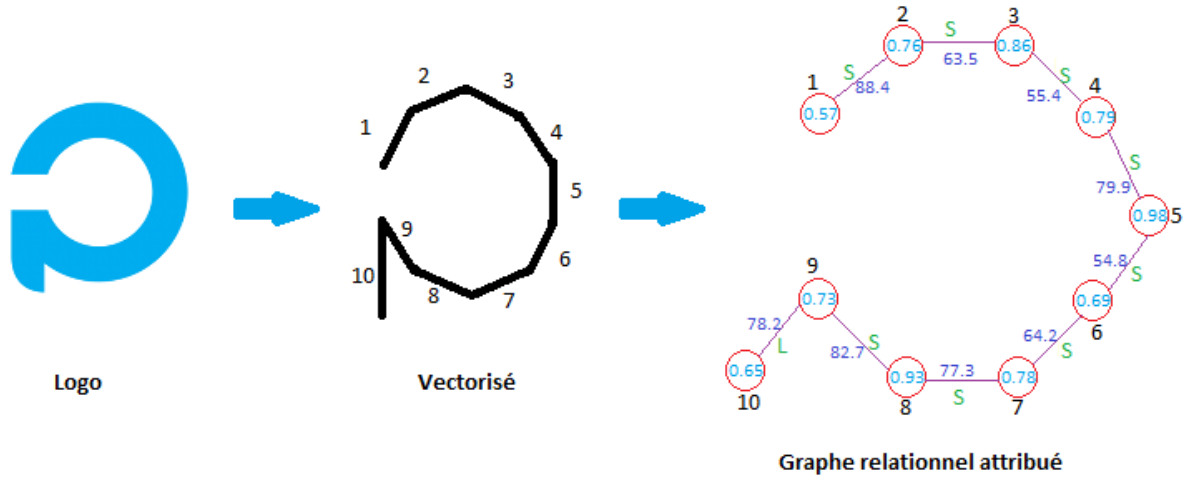
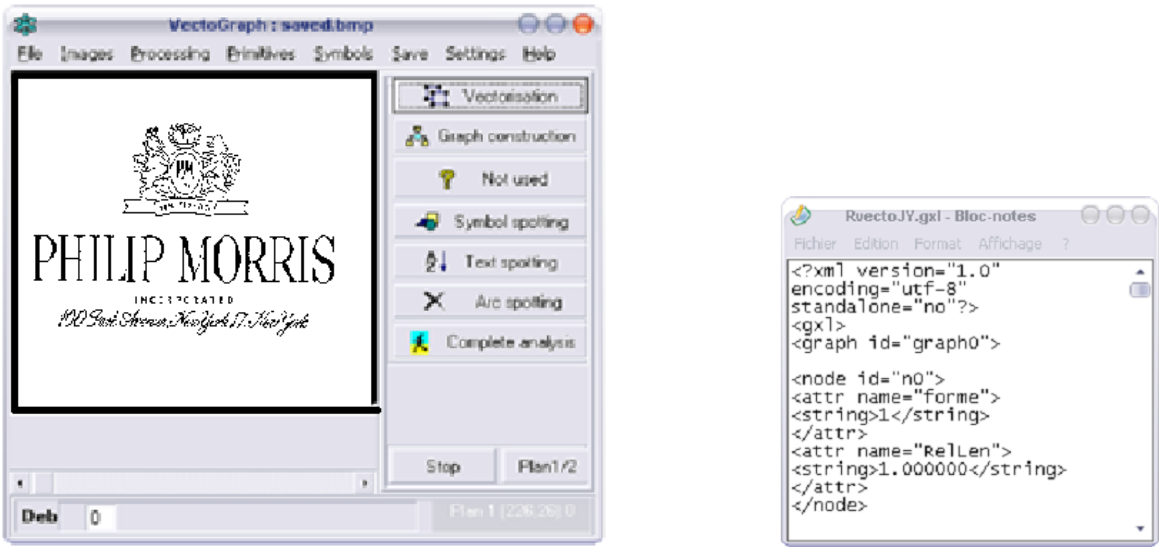


FIGURE 3.1 – Vectorisation et génération du graphe correspondant

Description

Le logiciel effectue plusieurs tâches, mais la partie qui nous intéresse est la vectorisation suivi par la construction du graphe correspondant.
La figure 3.2a et la figure 3.2b montrent respectivement un exemple d'entrée (un logo) et en sortie un exemple de graphe sous format GXL :



(a) VectoGraph (b) Fichier GXL

FIGURE 3.2 – VectoGraph

Weka

Weka est un logiciel développé en Java qui offre un contenu scientifique important surtout pour les chercheurs dans le domaine de l'apprentissage automatique.

Cet outil va intervenir dans ce projet dans les deux phases : la phase d'apprentissage (Learning) et plus précisément dans l'apprentissage d'un classificateur et dans le groupement des signatures, et également pendant la phase de requêtage lors de la classification.

FIGURE 3.3 – Weka

3.4.2 Base d'apprentissage

La collection d'images qui va servir comme base de test pour ce projet est Tabacco800[1], c'est une base de données composée de 1290 images de documents qui ont été collectés et numérisés au fil du temps. Les images contenues dans cette base montrent une portée réaliste et une complexité importante pour la communauté de recherche.
La figure 3.4 montre quelques exemplaires des documents contenus dans Tabacco800.

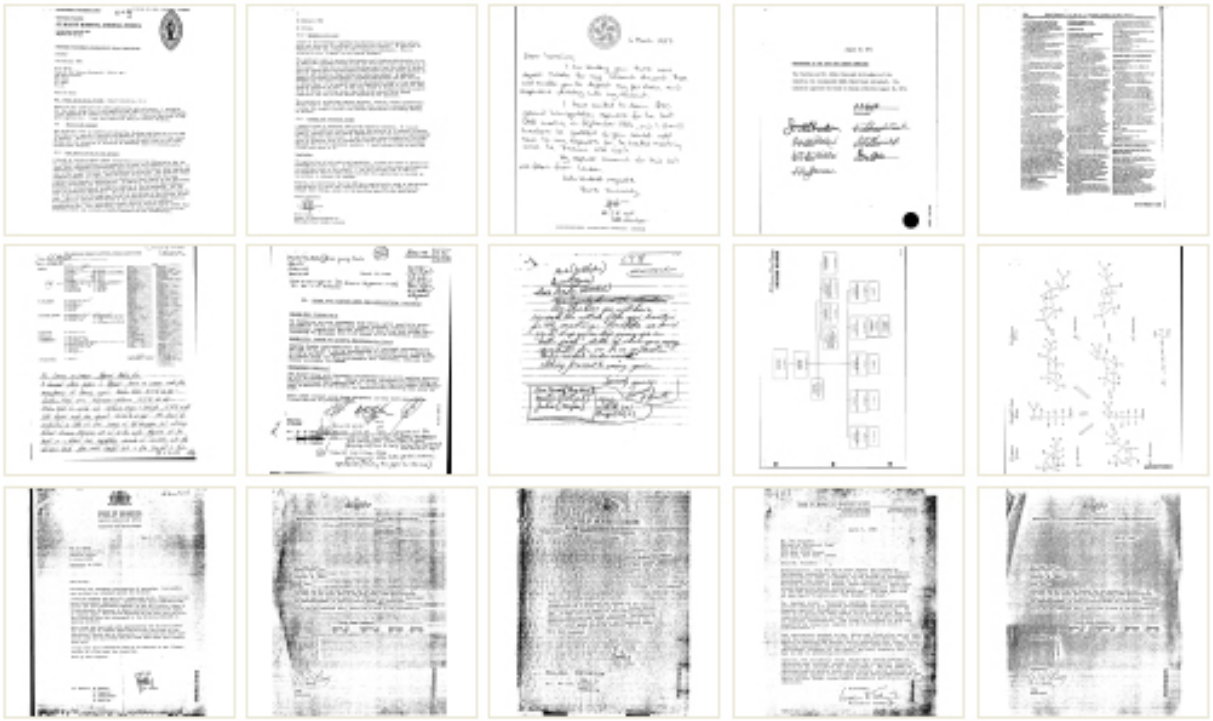


FIGURE 3.4 – Base d'images « Tabacco800 »

3.4.3 Architecture générale du système

Ce projet vise à développer une technique automatique de spotting ou localisation des logos dans les images de documents. Donc il s'agit de développer une application client/serveur permettant d'implémenter des algorithmes et de faire fonctionner les modules déjà réalisés par l'équipe RFAI. Cette application doit effectuer deux tâches, une première, dite non-supervisée, qui consiste à construire, après un certain traitement qu'on détaillera par la suite, une table Index qui va contenir pour chaque image : le graphe, les cliques, les vecteurs et les classes correspondants. La deuxième, dite supervisée, consiste à chercher, dans cette table, tous les images qui contiennent la partie sélectionnée par l'utilisateur (le logo).

La figure 3.5 montre les trois couches principales du système et l'interaction entre ses différents modules.

Le mode « Apprentissage » (Learning)

On dispose dans un premier temps d'une collection d'images (Tabacco800) de documents numérisés contenants des logos textuels et/ou graphiques, l'objectif du mode d'apprentissage est de trouver une nouvelle représentation de ces images permettant une recherche rapide et efficace dans leurs contenus. Pour cela, ce processus commence par une étape de pré-traitement en utilisant des techniques telles que la binarisation ou la segmentation afin d'améliorer la qualité des images initiales. Ensuite et pour chaque

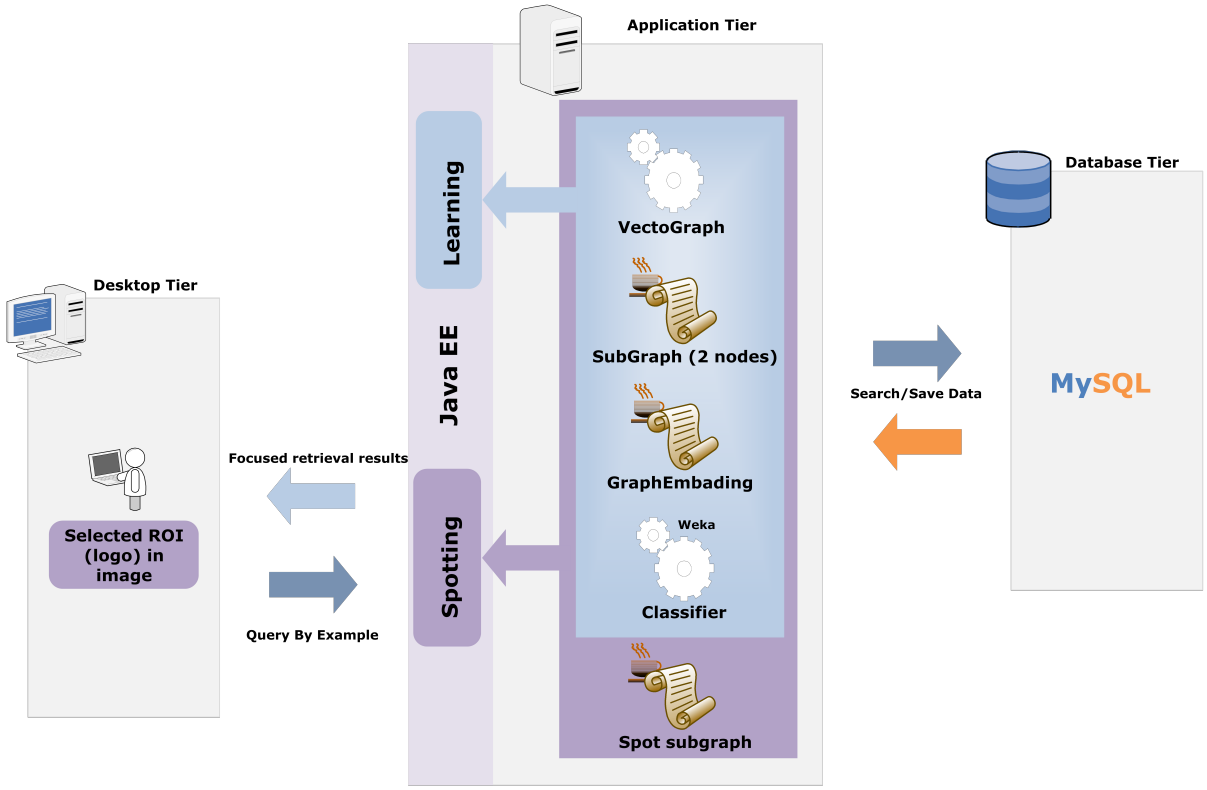


FIGURE 3.5 – Architecture du système

image, on va utiliser le logiciel VectoGraph pour la vectorisation et la construction du graphe correspondant sous format GXL, puis on va extraire tous les sous-graphes d'ordre 2 qui vont être encapsulés dans des signatures et regroupés dans des classes à l'aide d'un outil appelé Weka qui va construire en même temps un classificateur. Au final, on sauvegarde dans la table ń Index ż, pour chaque image, le graphe, les cliques, les signatures et les classes correspondants.

La figure 3.6 résume l'ensemble des étapes nécessaires pour la réalisation du processus d'apprentissage (Learnig) :

Le mode « Requête / Localisation » (Spotting)

Le deuxième mode de fonctionnement du système est un mode de repérage. Ce mode propose à l'utilisateur une interface graphique lui permettant de sélectionner la région qui contient le logo dans le document puis le système lancera un traitement similaire à celui du mode d'apprentissage en commençant par un traitement de la zone sélectionnée, puis une vectorisation et une construction du graphe représentant, ensuite, un extrait des sous graphes d'ordre 2 suivi d'une encapsulation dans des vecteurs. Et contrairement au mode précédent, l'étape suivante sera une utilisation des classes et du classificateur déjà construits pour localiser le logo dans la base d'images. La réponse à la requête envoyée par le client au début sera le document ou bien les documents qui contiennent le même logo ou un logo similaire.

La figure 3.7 résume l'ensemble des étapes nécessaires pour la réalisation du processus de localisation :

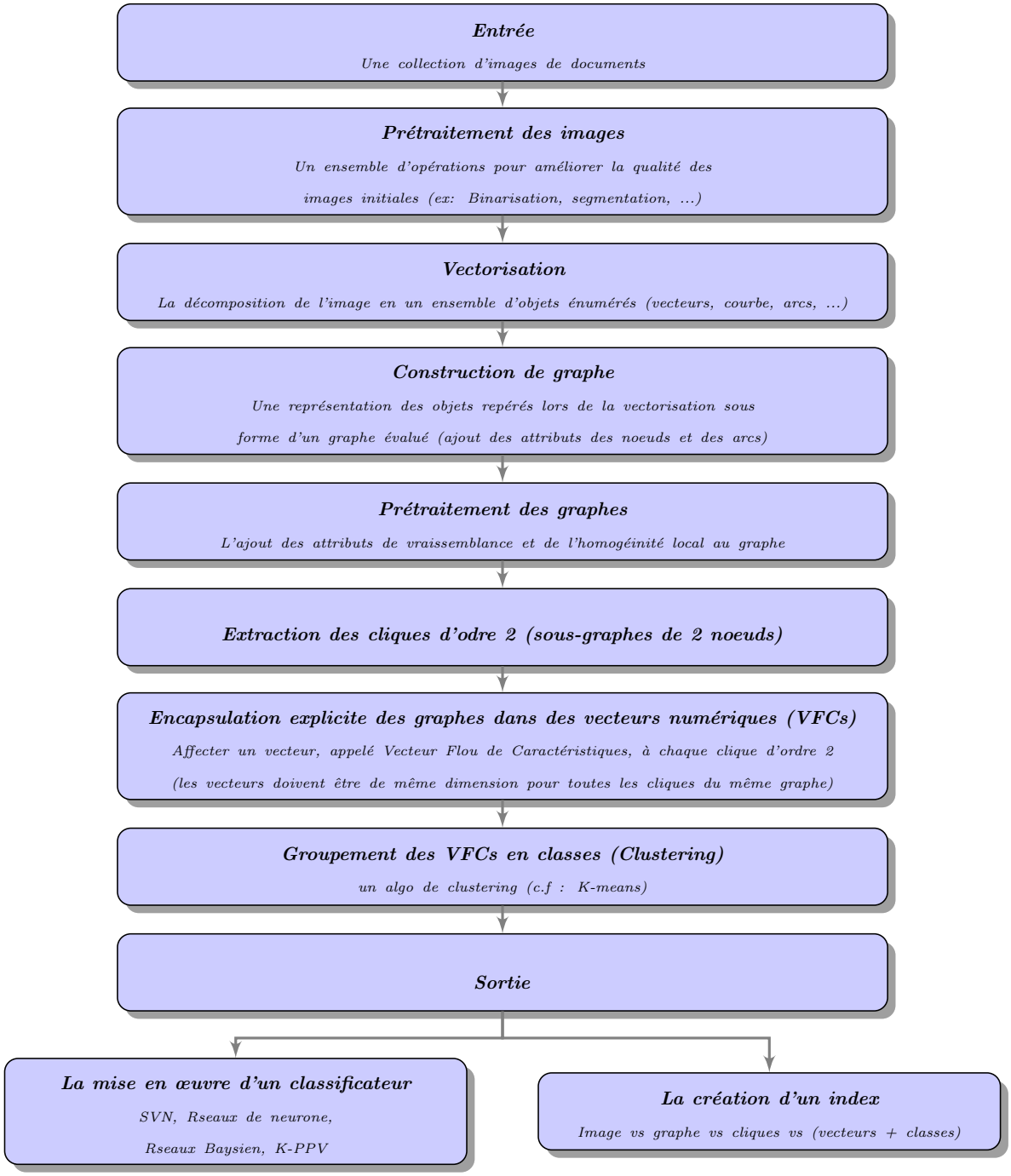


FIGURE 3.6 – Le mode « Apprentissage »

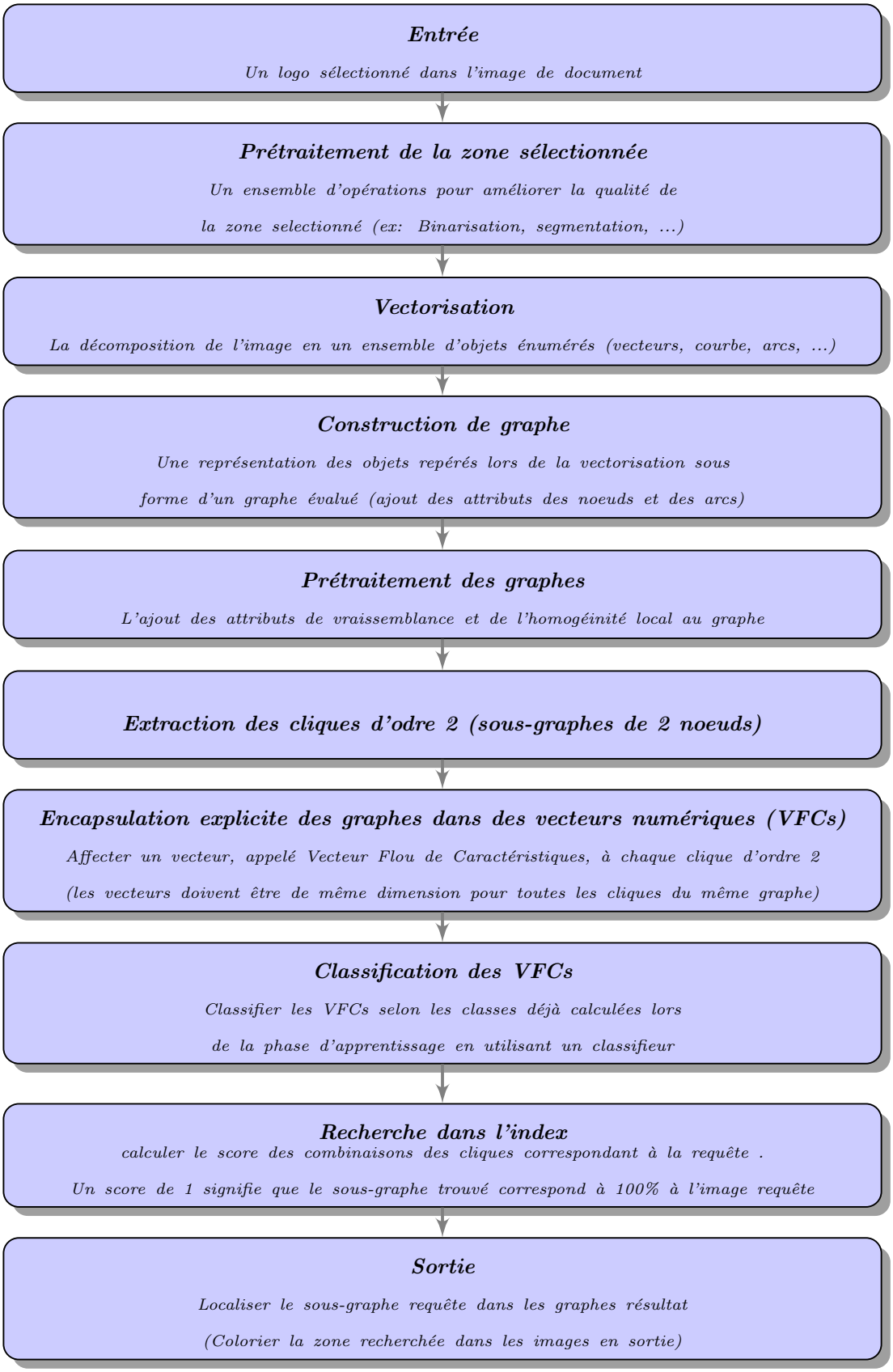


FIGURE 3.7 – Le mode « Requête »

Travail réalisé

Le processus de développement du projet est divisé en deux parties « Apprentissage » et « Requête », chaque partie est constituée de plusieurs étapes, l'entrée de l'étape encours représente la sortie de l'étape précédente ainsi de suite.

4.1 Le mode « Apprentissage »

4.1.1 Entrée

Comme je l'ai déjà mentionné dans la section 3.4.2, l'entrée de notre système sera une base d'images de documents numérisés contenants des logos textuels et/ou graphiques. La dégradation et le bruit sont variables pour chaque document, les images sont de format TIFF, donc il fallait les convertir en BMP pour qu'elles soient supportées par les modules utilisés par la suite.

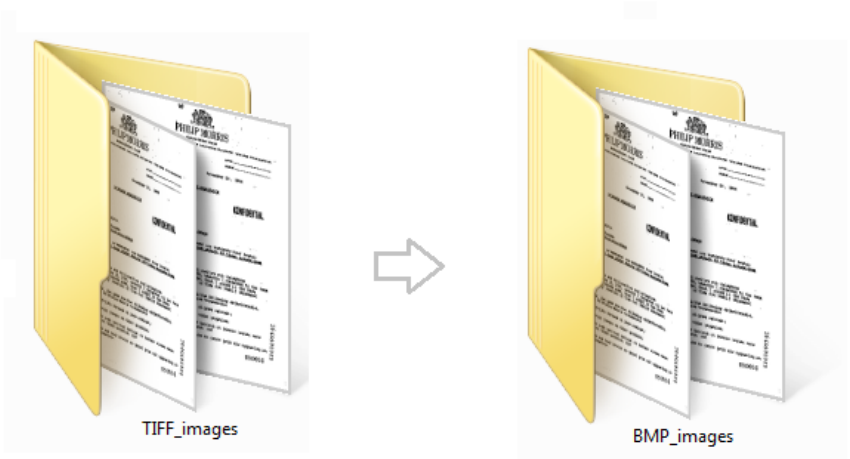
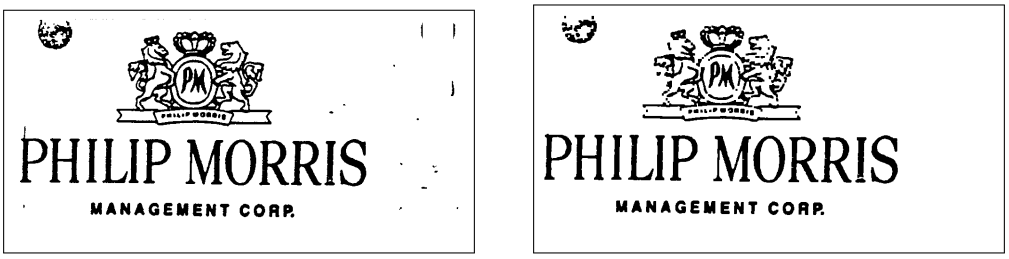


FIGURE 4.1 – Entrée

4.1.2 Pré-traitement des images

La première étape dans le mode « Apprentissage » consiste à améliorer la qualité des images et éliminer le bruit qu'elles contiennent, pour ce faire, deux algorithmes sont adoptés, **l'ouverture et la fermeture** (opening and closing)

– L'ouverture (Opening) : pour isoler les surfaces présentes dans l'image, éliminer le bruit et lisser les contours.
– La fermeture (Closing) : pour recoller des morceaux de surfaces proches.



(a) Avant le traitement (b) après le traitement

FIGURE 4.2 – Pré-traitement

L'implémentation des deux algorithmes est effectués grâce à la fonction *cvMorphologyEx* de la librairie *javaCV*[1]

4.1.3 Vectorisation

La deuxième étape consiste à créer pour une image donnée, le graphe correspondant en utilisant l'outil VectoGraph3.4.1.

Entrée : Le répertoire *images* contenant les images bitmap après avoir éliminer le bruit qu'elles contiennent

Sortie : Le répertoire *gxl* qui va contenir les fichiers gxl des images fournies en entrée

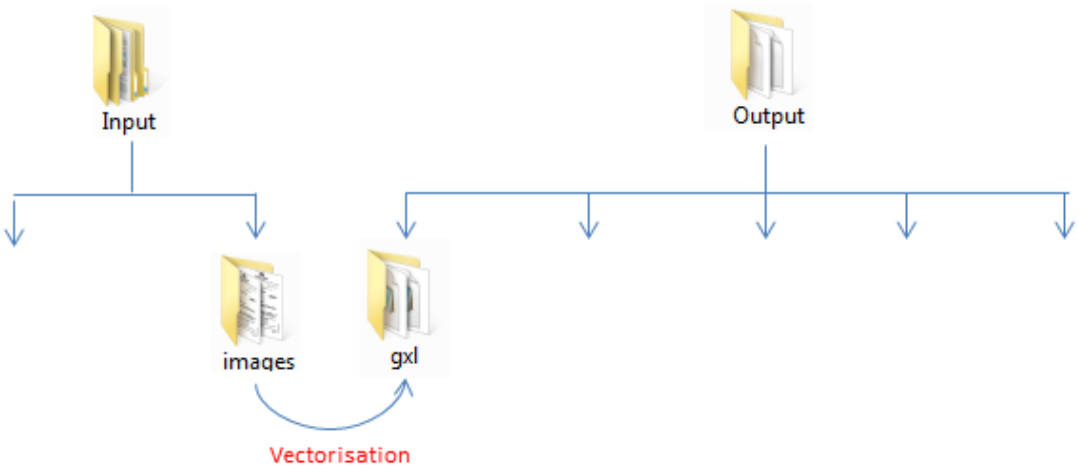


FIGURE 4.3 – Résultat de la vectorisation

Le processus de la vectorisation se fait d'une manière itérative sur l'ensemble des images contenues dans le fichier *images*, dans chaque itération, on supprime l'ancien fichier *RvectoGraph.gxl*, on exécute le programme *SymbSpotting_jy.gxl* ensuite on recopie le nouveau fichier *RvectoGraph.gxl* dans le répertoire *gxl*. Ce traitement est effectué par un thread qui s'exécute en arrière plan et qui appelle d'une manière répétitive le programme *SymbSpotting_jy.gxl* en passant comme paramètre l'image qu'on veut vectoriser.

1. javaCV est une interface qui permet d'utiliser les fonctions openCV dans un environnement java, pour plus d'informations : http ://code.google.com/p/javacv

La figure 4.4 montre l'interface graphique de la vectorisation au niveau du système :

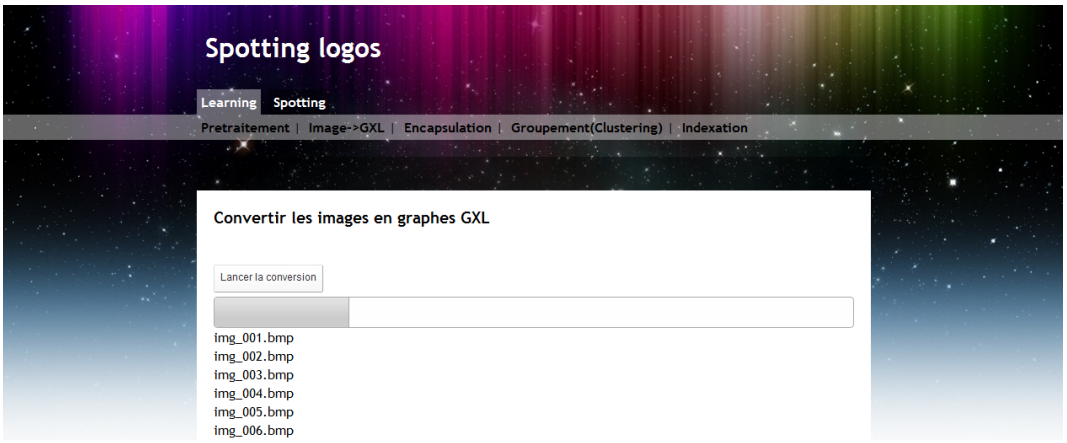


FIGURE 4.4 – Interface vectorisation

4.1.4 Lecture des fichiers GXL

Les graphes engendrés par la phase précédente sont enregistrés sous format *GXL (Graph eXchange Language)*, qui est un format *XML* destiné à l'échange des structures de données telles que les graphes. Afin de charger les données en mémoire, j'ai créé dans un premier temps un parseur GXL adapté aux fichiers gxl de la base d'images GREC en utilisant la librairie GXL [2] en java, la lecture se fait d'une manière dynamique et elle permet l'alimentation des structures de données définies dans le système tells que la matrice d'adjacence, la liste des nœuds, des arcs et leurs attributs, la distinction entre les attributs numériques et symboliques est basée sur le type de la balise gxl, si la valeur est de type *Integer* ou *Float*, donc il s'agit d'un attribut numérique, et dans le cas d'un attribut symbolique, la valeur est de type *String*.

Une deuxième version du parseur a était développée après le changement de la base d'images, ce qui a engendré une différence de la structure des fichiers GXL au niveau des types des attributs. les nouveaux fichiers gxl ont tous des types *String* que ça soit pour les attributs numériques ou symboliques. Dans ce cas, j'ai opté pour un autre type de parseur qui utilise le *SAX Simple API for XML* et qui a permis de résoudre ce problème.

4.1.5 Pré-traitement des graphes

Cette étape a comme but le calcul des attributs supplémentaires à ceux qui sont calculés grâce à l'étape précédente tels que la longueur relative, le type de liaison ou l'angle relatif entre les quadrilatères, ces attributs sont dits des attributs de ressemblance. ils permettent l'ajout des informations d'homogénéité locale aux graphes.
Le calcul de la ressemblance varie selon le type des attributs, pour une arête par exemple, la ressemblance entre les attributs numériques des nœuds qui sont reliés est calculé selon l'équation 1, et pour les attributs symboliques on utilise l'équation 2.

$$ressemblance = min(\mid a_1 \mid, \mid a_2 \mid)/max(\mid a_1 \mid, \mid a_2 \mid) \quad (1)$$

$$ressemblance = \left| \begin{array}{cc} 1 & b_1 = b_2 \\ 0 & sinon \end{array} \right| \quad (2)$$

2. http ://gxl.sourceforge.net/docs/gxl/api/index.html

De la même façon, les attributs de ressemblance pour les nœuds sont calculés en fonction des attributs des arêtes reliées. Pour un nœud, un attribut de ressemblance est calculé comme la moyenne de la ressemblance entre chaque paire d'arêtes en utilisant l'équation (1) pour les attributs numériques et l'équation (2) pour les attributs symboliques.

La figure 4.5 montre les attributs intégrés au graphe après le calcul de la ressemblance :

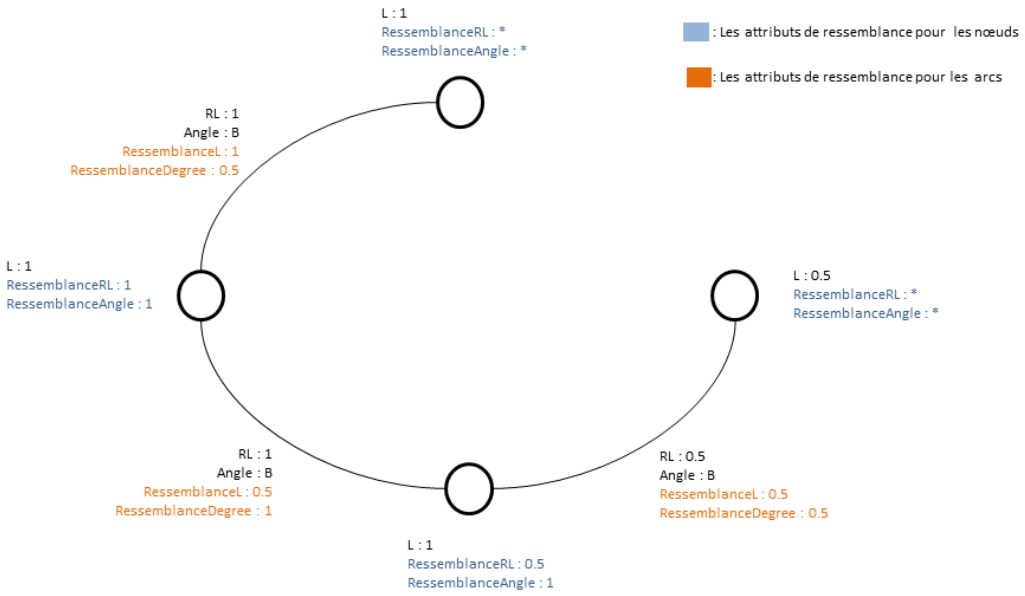


FIGURE 4.5 – Pré-traitement des graphes

4.1.6 Extraction des cliques d'ordre 2

L'étape qui suit la lecture des graphes et l'ajout des informations de la ressemblance, est l'extraction des cliques d'ordre 2 4.6. Pour chaque graphe AG_k on construit l'ensemble de cliques correspondant donné par :
$subAG_k = \{subAG_k^1, subAG_k^2, ..., subAG_k^i\}$ tel que $subAG_k^i$ est la i^e clique du k^e graphe.

4.1.7 Encapsulation des graphes

L'étape suivante est l'encapsulation des graphes (Graph embedding) 4.7 qui consiste à passer d'une représentation sous forme de graphes en une représentation numérique en encapsulant les cliques extraites dans des vecteurs appelés *Vecteurs Flous de Caractéristiques* (VFCs).

La phase d'apprentissage non supervisée

l'objectif de la phase d'apprentissage non supervisée est la construction, pour chaque attribut, son intervalle discret dans le cas d'un attribut symbolique, et un intervalle flou dans le cas d'un attribut numérique.

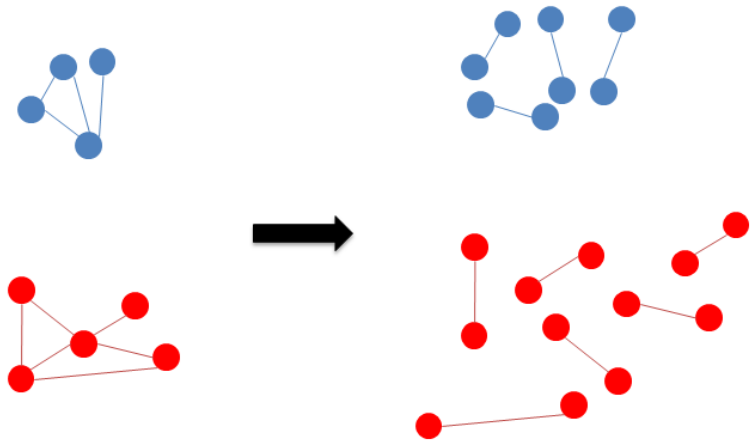


FIGURE 4.6 – Extraction des cliques d'ordre 2

FIGURE 4.7 – encapsulation des graphes

L'algorithme 2 présente une méthode qui, pour un attribut donné,permet de construire l'intervalle discret correspondant.

Algorithme : GetInitCrispInterval($listvaluesAttribute_i, n_i$)

Description : Calcule des intervalles discrets de même longueur
Entrée : La liste des valeurs de l'attribut *i*
Entrée : Nombre d'intervalles souhaités $n_i (n_i \geq 2)$
Sortie : n_i intervalles discrets pour l'attribut *i*

Fonction GetInitCrispInterval($listvaluesAttribute_i, n_i$: **liste, entier**) : **intervalles discrets**
    intervallesDeTailleFixe ← GetEqualSpacBin($listvaluesAttribute_i, n_i$)
    intervallesDiscrets ← vide
    st ← - ∞
    en ← intervallesDeTailleFixe[1]
    j ← 1
    Répéter
        intervallesDiscrets[j].start ← st
        intervallesDiscrets[j].end ← en
        st ← en
        en ← intervallesDeTailleFixe[j+1]
        j ← j+1
    jusqu'à ce que $(j > n_i)$
    Retourner intervallesDiscrets ;
Fin

Algorithme 2 – L'algorithme GetInitCrispInterval

Après avoir construire les intervalles discrets qui seront utilisés dans l'encapsulation des attributs symboliques, la deuxième étapes consiste à utiliser ces structures pour déduire des trapézoïdes qui vont constituer les intervalles de flou pour les attributs numériques selon le schéma 4.8.

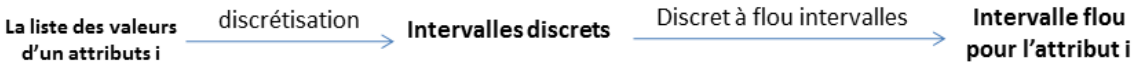


FIGURE 4.8 – Apprentissage des intervalles flous pour un attribut i

L'algorithme 5 explique en général, le pseudo code de la construction des intervalles flous. dans un premier temps il calcule les intervalles discrets correspondants à l'attribut passé en argument, ensuite il les intègre dans des trapézoïdes afin d'obtenir un ensemble d'intervalles trapézoïdaux.

Algorithme : GetFuzzyOverlapTrapzInterval($listvaluesAttribute_i, s_i$)

Description : Calcule des intervalles flous pour un attributs *i*
Entrée : La liste des valeurs de l'attribut *i*
Entrée : Nombre d'intervalles souhaités $s_i (s_i \geq 2)$
Sortie : s_i intervalles discrets pour l'attribut *i*

Fonction GetFuzzyOverlapTrapzInterval($listvaluesAttribute_i, s_i$: **liste, entier**) : **intervalles flous**

$n_i \leftarrow 2 * s_i - 1$
intervallesDiscrets ← GetInitCrispInterval($listvaluesAttribute_i, n_i$)
intervallesFlous ← vide
intervallesFlous[1].a ← - ∞
intervallesFlous[1].b ← - ∞
intervallesFlous[1].c ← intervallesDiscrets[1].end
intervallesFlous[1].d ← intervallesDiscrets[2].end
j ← 1
jdiscret ← 0
Répéter
  j ← j+1
  jdiscret ← j+2 intervallesFlous[j].a ← intervallesFlous[j-1].c
  intervallesFlous[j].b ← intervallesFlous[j-1].d
  Si $(jdiscret + 1 \geq n_i)$ **Alors**
    intervallesFlous[1].c ← + ∞
    intervallesFlous[1].d ← + ∞
    break
  Fin Si
  intervallesFlous[1].c ← intervallesDiscrets[jdiscret+1].end
  intervallesFlous[1].d ← intervallesDiscrets[jdiscret+2].end
jusqu'à ce que $(j > s_i)$
Retourner intervallesFlous ;
Fin

Algorithme 5 – L'algorithme GetFuzzyOverlapTrapzInterval

L'algorithme *GetFuzzyOverlapTrapzInterval* permet de construire un ensemble de trapézoïdes dont chacun est défini par 4 points (a,b,c,d)4.9, le degré d'appartenance d'un point appartient à l'intervalle continu [0,1]

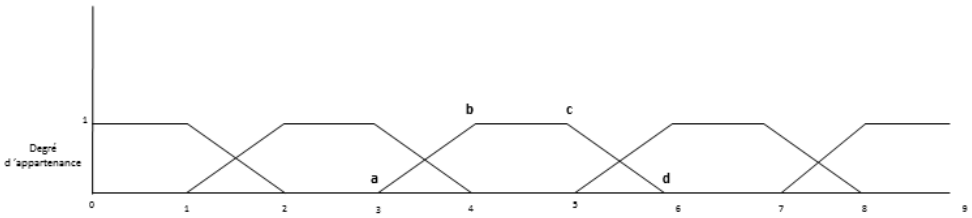


FIGURE 4.9 – Intervalles trapézoïdaux

Encapsulation du graphe

La phase de l'encapsulation d'un graphe utilise les intervalles construits lors de la phase d'apprentissage non supervisée, ensuite, pour chaque attribut, on déduit son histogramme en utilisant les intervalles discrets pour les attributs symboliques, et les intervalles flous trapézoïdaux pour les attributs numériques.

La concaténation des histogrammes de tous les attributs en ajoutant d'autres informations qu'on détaillera par la suite constitue un vecteur représentant le graphe dans un espace $\Re^n$.
La représentation numérique, autrement dit « *signature* », d'un graphe prendra la structure suivante :

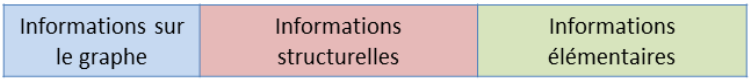


FIGURE 4.10 – Vecteur Flou de Caractéristiques

Informations sur le graphe : Cette partie intègre deux informations sur la structure générale du graphe, une première concernant son Ordre (le nombre de nœuds) et la deuxième information c'est sa Taille (le nombre d'arcs).

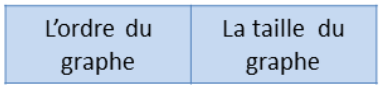


FIGURE 4.11 – Encapsulation des informations concernant le graphe

Informations structurelles : La deuxième partie intègre des informations sur le degré des nœuds (nombre d'arcs connectés à chaque nœud) et sur la ressemblance des attributs pour les nœuds et pour les arcs.

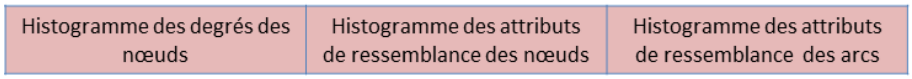


FIGURE 4.12 – Encapsulation des informations structurelles

Informations élémentaires : La dernière partie intègre les informations élémentaires concernant les attributs numériques et symboliques pour les nœuds et pour les arcs.

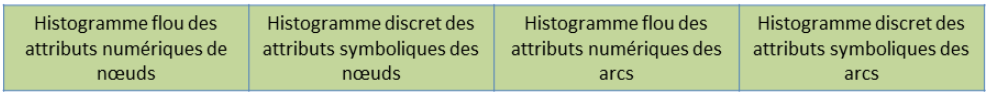


FIGURE 4.13 – Encapsulation des informations élémentaires

Implémentation de l'encapsulation dans le système

L'encapsulation du graphe (clique d'ordre 2) au niveau implémentation respecte le même ordre cité préalablement :

La phase d'apprentissage non supervisée :

Entrée :
- La liste des attributs numériques des nœuds et des arcs.
- La liste des attributs symboliques des nœuds et des arcs.
- Le nombre désiré d'intervalles pour chaque attributs numérique.

Sortie :
- Ensemble d'intervalles flous trapézoïdaux pour les attributs numériques
- Ensemble d'intervalles discrets pour les attributs symboliques

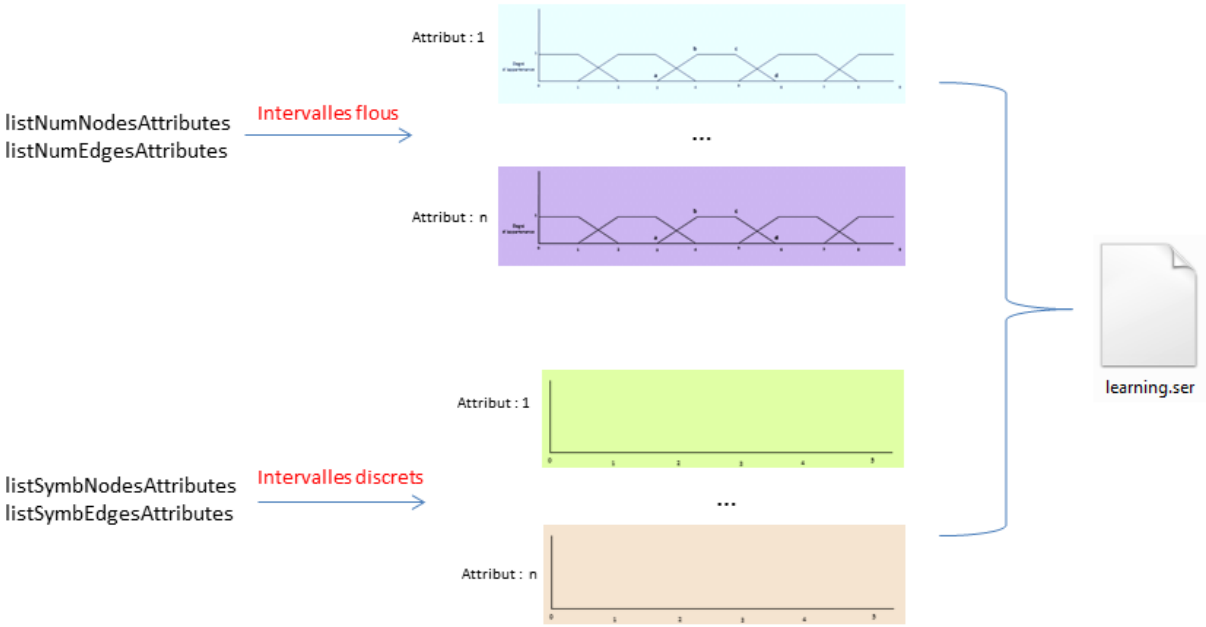


FIGURE 4.14 – La phase d'apprentissage non supervisée

Le système génère un fichier *Learning.ser* qui stocke les intervalles construits. Ce fichier en question est le résultat d'une sérialisation d'un objet Java, ce qui va permettre de lancer l'apprentissage une seule fois, ensuite on peut l'utiliser sur d'autre machines sur lesquelles le système est déjà installé.

Encapsulation du graphe :

Entrée :

- La liste des cliques d'ordre 2 issues de l'étape 4.1.6
- Le fichier *Learning.ser*

Sortie :

- Liste des Vecteurs Flous de Caractéristiques (signatures) de chaque clique

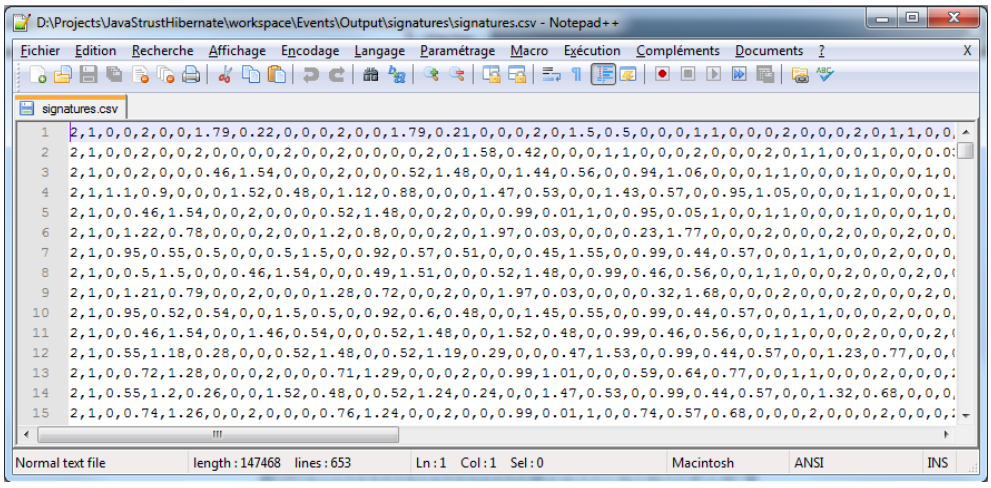


FIGURE 4.15 – Les vecteurs flous de caractéristiques (signatures)

La figure 4.16 montre le positionnement des fichiers résultat au niveau de l'arborescence du système.

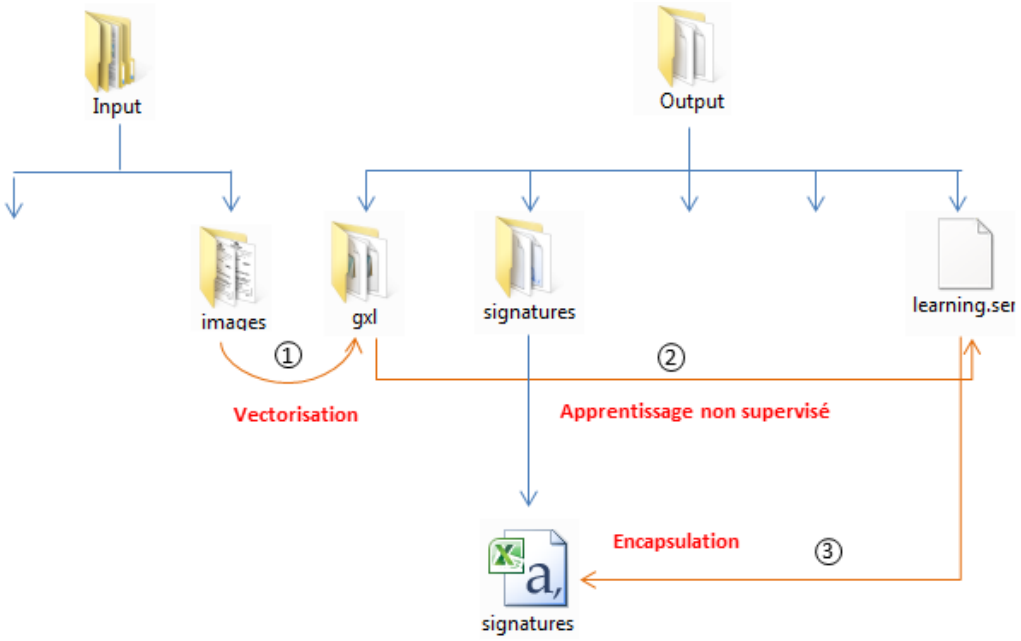


FIGURE 4.16 – Résultats de la phase d'encapsulation

4.1.8 Groupement des VFCs en classes (Clustering)

Après avoir construit les Vecteurs Flous de Caractéristiques (VFC) correspondants à chaque clique, l'étape qui suit est le groupement de ces vecteurs en classes homogènes, les vecteurs appartenant à la même classe partagent des caractéristiques similaires. la méthode utilisée est "K-means" de la librairie Weka en utilisant la fonction **weka.clusterers.SimpleKMeans** après l'avoir intégrer dans l'environnement java du système. Le résultat du groupement est enregistré dans un nouveau fichier qui s'appelle *forLearningClassifier.csv* de telle façon à ce que chaque signature finit par le nom de la classe pour laquelle est attribuée (voir figure 4.17)

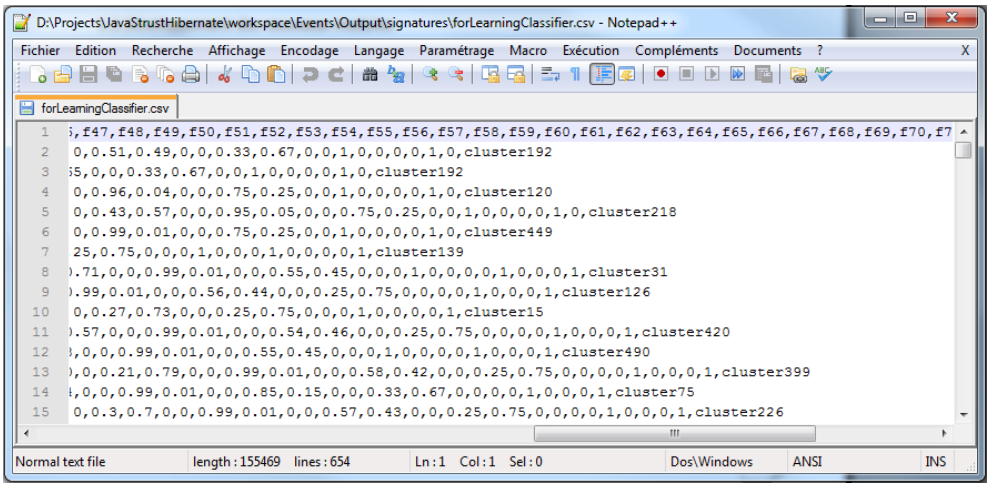


FIGURE 4.17 – Résultats du groupement des vecteurs

4.1.9 Apprentissage d'un classificateur

Le classificateur utilisé est de type **KNN (K-nearest neighbor)** de la librairie Weka. il permet par la suite, pour une nouvelle signature, de trouver la classe correspondante. Le classificateur est stocké dans un fichier *classifier.ser* pour qu'il soit utilisé par la suite dans la phase de la classification.

4.1.10 Indexation

La phase d'indexation consiste à sauvegarder dans une base de données l'ensemble des informations liées à chaque signature : sa classe, le nom du fichier GXL (d'où elle vient) et ces coordonnées 4.18 . La base de données utilisée est une base de donnée *MySQL*.

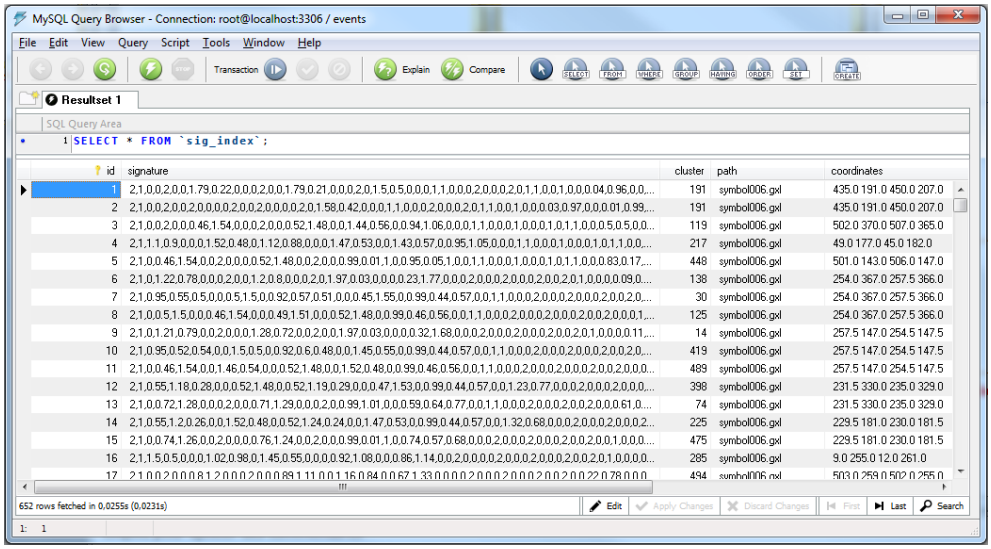


FIGURE 4.18 – La base d'indexation

4.1.11 Résumé du mode « Apprentissage »

Le schéma suivant 4.5 résume les différentes fonctions implémentées concernant le mode « Apprentissage » ainsi que les résultats qu'elles génèrent .

4.2 Le mode « Localisation » (Spotting)

La deuxième principale fonctionnalité du système est de pouvoir sélectionner un logo par l'utilisateur et de lancer une recherche dans notre collection d'images afin de repérer les documents qui contiennent ce logo. Les traitements qui seront effectués sur la zone d'image sélectionnée (le logo) seront presque les mêmes cités dans la phase d'apprentissage avec quelques différences qu'on va citer par la suite.

4.2.1 Entrée

L'entrée du mode « Localisation (Spotting) » sera une partie d'image qui contient un logo, donc le système doit offrir à l'utilisateur une interface graphique qui permet de sélectionner avec la souris une zone dans le document affiché, et c'est grâce au package **JCrop** de la librairie **JQuery** qu'on a pu intégrer cette fonctionnalité comme le montre la figure 4.20.

Ensuite, le système récupère les coordonnées de la zone sélectionnée, découpe le logo, et l'enregistre dans dans un fichier nommé « logo ».

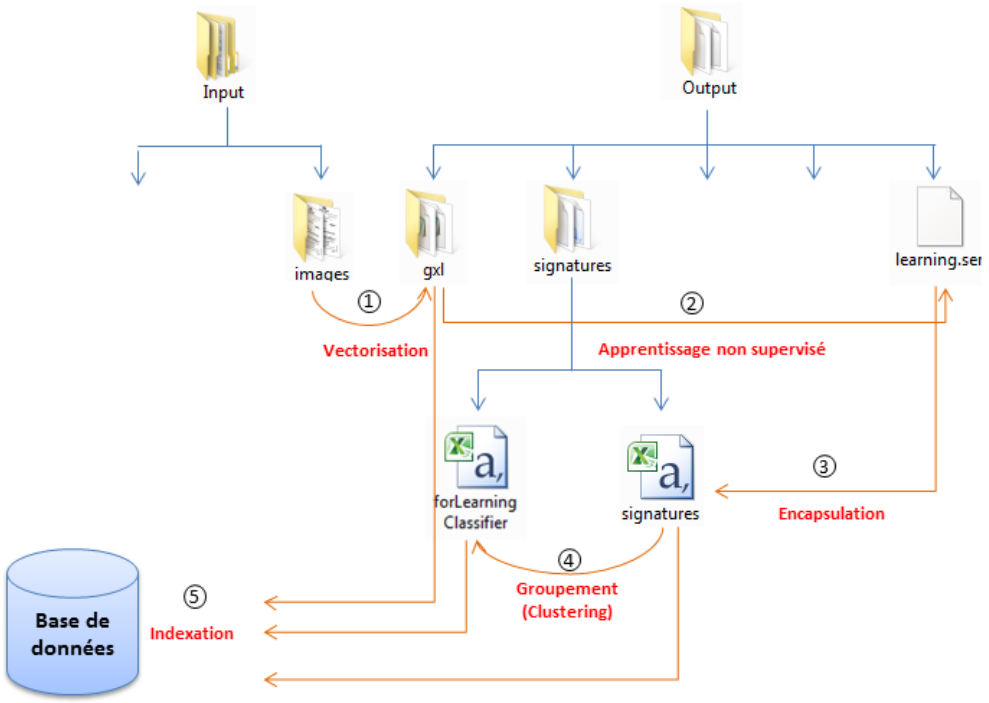


FIGURE 4.19 – Les étapes de l'Apprentissage

FIGURE 4.20 – Sélection d'un logo

4.2.2 Pré-traitement de la zone sélectionnée

Cette partie est similaire au pré-traitement des images dans le mode « Apprentissage » 4.1.2 sauf que cette fois le traitement à effectuer sera seulement sur le logo sélectionné.

4.2.3 Vectorisation

Idem (cf. 4.1.3)

4.2.4 Pré-traitement des graphes

Idem (cf. 4.1.5)

4.2.5 Extraction des cliques d'ordre 2

Idem (cf. 4.1.6)

4.2.6 Encapsulation des graphes

Cette étape consiste à encapsuler les cliques d'ordre 2 extraites du logo sélectionné dans des vecteurs flous de caractéristiques en utilisant les intervalles flous trapézoïdaux et intervalles discrets déjà calculés lors du mode « Apprentissage ».

4.2.7 Classification des VFCs

Cette étape consiste à classifier les VFCs (les signatures) en utilisant le classificateur générer lors de la phase d'apprentissage, ce qui va permettre d'affecter, aux signatures requête, les mêmes classes que les signatures qui leurs ressemblent dans la base d'indexation.

4.2.8 Recherche dans l'index

Après la classification des VFCs, on va chercher dans la base de données, les documents qui contiennent toutes les classes trouvées dans la requête, ensuite, pour chacun de ces documents, on va récupérer sa matrice d'adjacence puis en modifie les valeurs telles que pour chaque case (i, j) qui représente un arc entre les nœuds i et j, aura comme valeur :

⇒ 0, s'il n'y a pas d'arc entre i et j
⇒ 1, s'il y a un arc entre i et j
⇒ 2, s'il y a un arc entre i et j et il a la même classe que celles de la requête

L'étape suivante consiste à calculer le score pour chaque arc d'une valeur égale à 2 en utilisant la formule suivante :

$$score = \sum_{Z=0}^{2} (Z \times \frac{|Z|}{W})$$

avec :

⇒ Z, une valeur dans la matrice d'adjacence (0, 1 ou 2)
⇒ |Z|, sa fréquence dans les case voisines
⇒ W, la taille de la fenêtre choisie pour le calcul du score

La figure 4.21 montre un exemple pour le calcul du score avec une fenêtre de taille 3×3

Au final, on trie les cliques selon leurs score pour en sélectionner les meilleurs, ensuite on récupère ses coordonnées pour les afficher avec une couleur différente afin de les localiser dans les documents résultat.

4.2.9 Post-traitement

Les premiers résultats obtenus ont montré qu'il fallait ajouter une autre étape afin d'éliminer les faux positifs (voir la figure ?), c'est-à-dire qu'il y a des documents qui contiennent l'ensemble des signatures qui ressemblent aux signatures requête, mais en calculant leurs scores, on en déduit qu'ils ne dépassent pas le seuil défini dans le fichier de paramétrage, donc ces documents seront éliminés et ile ne figureront pas parmi les résultats.

Une deuxième constatation qu'on a remarqué était la dispersion des zones repérées dans les documents résultat qui est dû à la structure bruitée de ces derniers, et pour remédier à cela, on a ajouté une étape dans le post-traitement qui consiste à repérer et n'afficher que les zones denses. Cette procédure se base

La matrice d'adjacence

1	0	0	1	1	0
0	0	O	0	0	0
0	O	1	O	0	0
0	1	2	2	0	0
0	0	2	0	0	1
0	0	0	0	1	2

Pas d'arc

Une clique qui n'a pas la même classe que celles trouvées dans la requête

Une clique qui a la même classe qu'une autre clique de la requête

La clique qu'on voudrait calculer son score

W = 3 × 3

$$score = \sum_{Z=0}^{2}(Z \times \frac{|Z|}{W}) = \left(0 \times \frac{4}{9}\right) + \left(1 \times \frac{2}{9}\right) + \left(2 \times \frac{3}{9}\right) = 0{,}88889$$

FIGURE 4.21 – Calcul de score

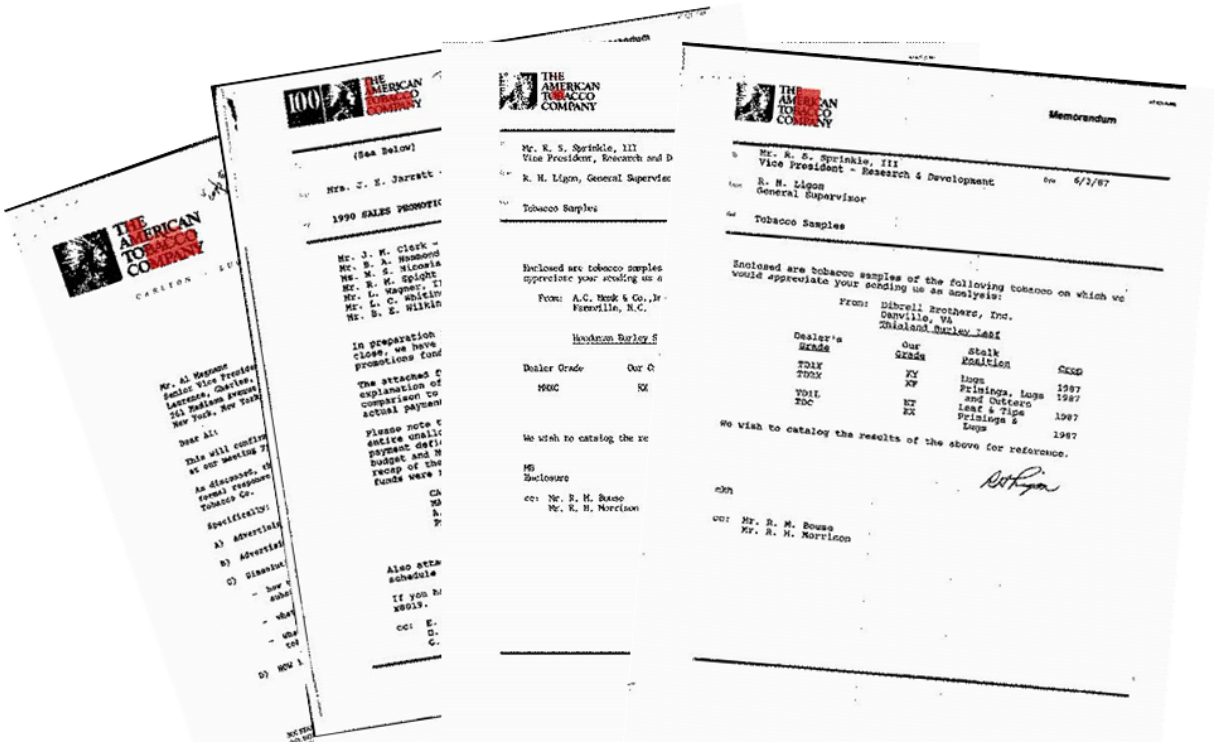


FIGURE 4.22 – Résultat de la localisation

sur un calcul pour chaque signature sélectionnée, du nombre de voisins selon un horizon préalablement défini (exemple : une fenêtre de même taille que l'image requête).

Spécification logicielle du système

5.1 Langage de développement

L'objectif principal de ce projet était de concevoir et développer un système de localisation des logos dans les images de documents, et contrairement aux projets précédents notamment ceux de *Adil MARKOUCHE* et de *Anass AYOUBI* qui ont été développés dans le cadre de leurs projets de fin d'études et qui utilisent à la fois des scripts MATLAB et le langage Java, ce système a la particularité d'être développé seulement en Java, et plus précisément en *Java EE* puisqu'il s'agit d'une application web Client-serveur.
D'autres langages sont utilisés dans la partie présentation tel que le **JQuery**, une librairie **javaScript** qui offre plusieurs fonctionnalités comme celle utilisée dans la sélection du logo, ou la barre de progression lors de la création des fichiers GXL, etc. L'autre langage utilisé est le JSP à côté du HTML et du CSS pour la création des pages web.

5.2 Outils de développement

Afin d'avoir un code source bien structuré, le choix d'un framework était indispensable pour bien séparer les données, le traitement et la présentation, pour cela, j'ai choisi **Struts** qui adopte l'architecture *Modèle-Vue-Contrôleur* et qui offre d'autres fonctionnalités telles que la validation des données saisies, le contrôle des actions déclenchées par l'utilisateur, etc. En ce qui concerne la couche accès aux données, j'ai utilisé **Hibernate**, un framework open source qui gère la persistance des objets Java en base de données.

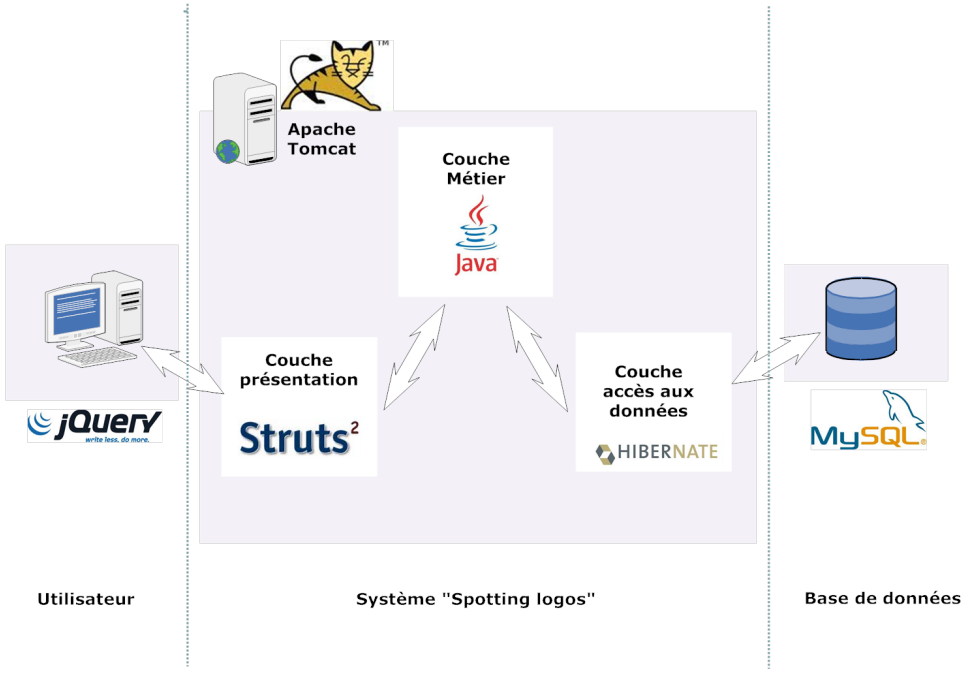


FIGURE 5.1 – Architecture logicielle du système

5.3 Diagrammes UML

5.3.1 Diagramme d'utilisation

Le diagramme 5.2 montre les différents cas d'utilisation du système et son interaction avec les acteurs externes.

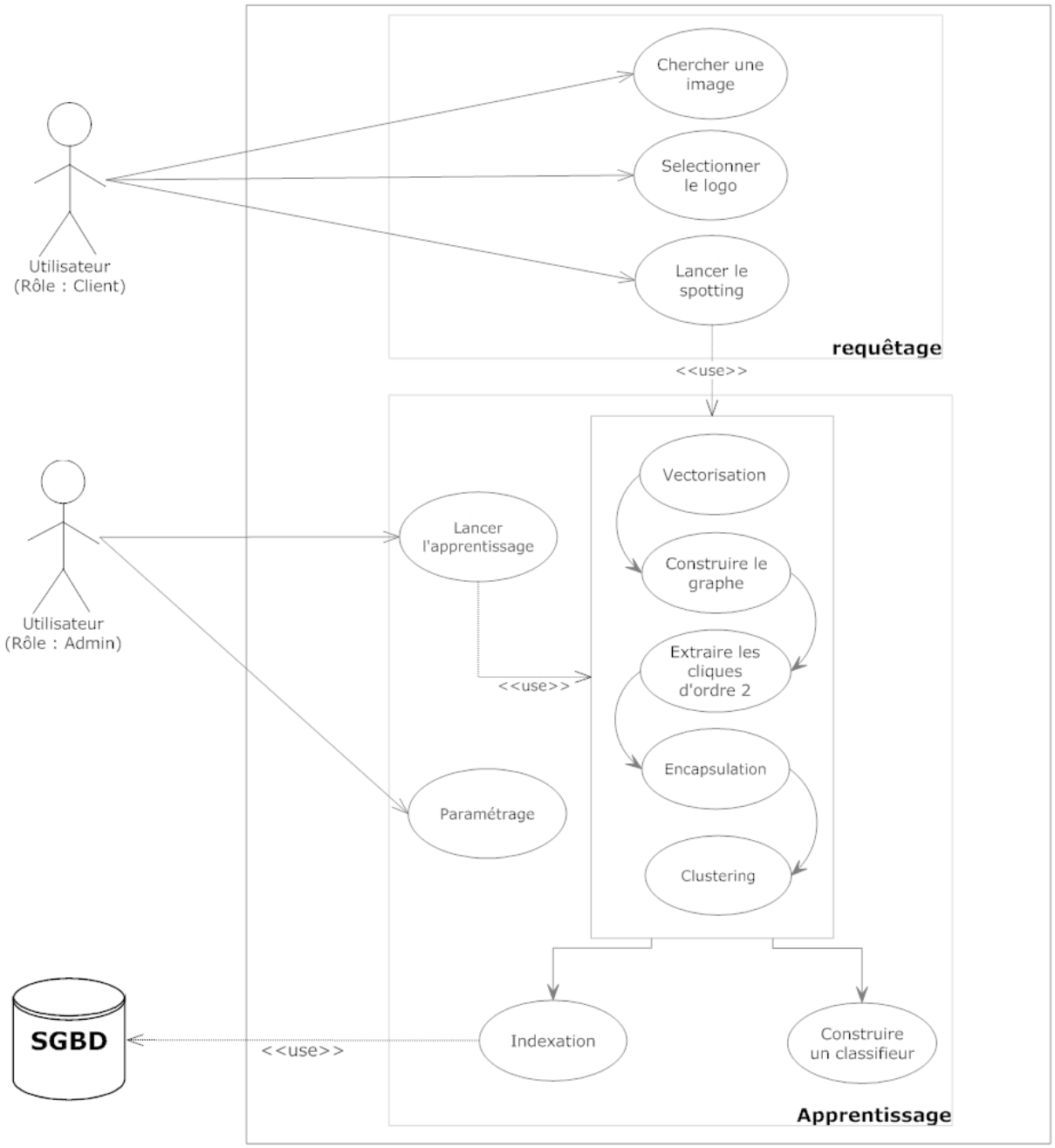


FIGURE 5.2 – Diagramme de cas d'utilisation

Sélectionner le logo

L'interface web développée offre à l'utilisateur la possibilité d'afficher, puis de sélectionner la partie qui contient le logo, ensuite le système est capable de récupérer la zone sélectionnée sous forme d'une image

puis il lance le Spotting là-dessus. 4.2

L'encapsulation explicite des graphes

Cette fonction consiste à passer d'une représentation descriptive d'un sous graphe d'ordre 2 (des fichiers XML) à une représentation numérique sous forme d'un vecteur (signature) appelé Vecteur de Flou de Caractéristiques (VFC). Cette transformation permettra un traitement rapide et un stockage plus facile. 4.1.7

Paramétrage

Dans certaines étapes et surtout dans la phase d'apprentissage, ils existent plusieurs paramètre à passer au système tel que le nombre limite de documents à retourner lors de la recherche, ou le seuil du score à définir pour la sélection des signatures, etc. Dans ce cas, le système doit offrir à l'utilisateur le choix d'utiliser une telle ou telle valeur selon ces besoins, et c'est pour cette raison qu'une partie de paramétrage est nécessaire afin d'ajouter une certaine souplesse au système.

La base d'indexation

Cette base d'indexation sera une table dans une base de données MySQL qui, lors du mode d'apprentissage, va stocker, pour chaque image, le graphe, les cliques, les signatures et les classes correspondants. Et pour le mode Spotting, cette table va servir une base de recherche des documents qui contiennent les mêmes régions d'intérêt sélectionnées par l'utilisateur.

Localisation des ROIs

Cette fonction intervient dans la phase finale du mode de localisation, elle permet de repérer les ROIs recherchées par l'utilisateur dans les documents trouvés.

5.3.2 diagramme de séquence

Le mode « Apprentissage »

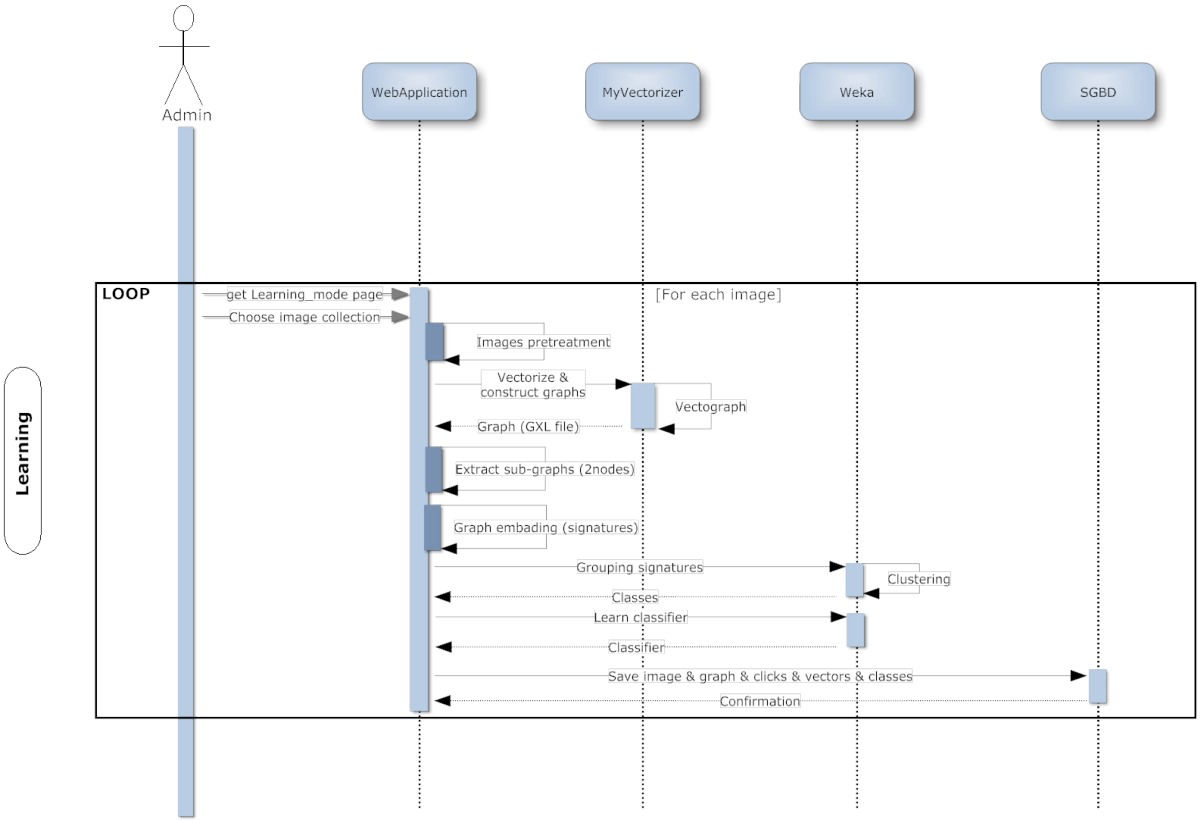


FIGURE 5.3 – Diagramme de séquence du mode « Apprentissage »

Comme vous pouvez le constater, lors de la phase d'apprentissage, le système interagit dans un premier temps avec l'utilisateur pour le choix de la collection d'images, ensuite il construit les fichiers GXL de chaque image en faisant appel à VectoGraph 3.2a, puis il encapsule les cliques d'ordre 2 dans des signatures qui seront regroupées à l'aide de Weka dans des classes pour les enregistrer dans une base de données en faisant référence pour chaque signature, au graphe à qui elle appartient, sa classe et ses coordonnées.

Le mode « Localisation »

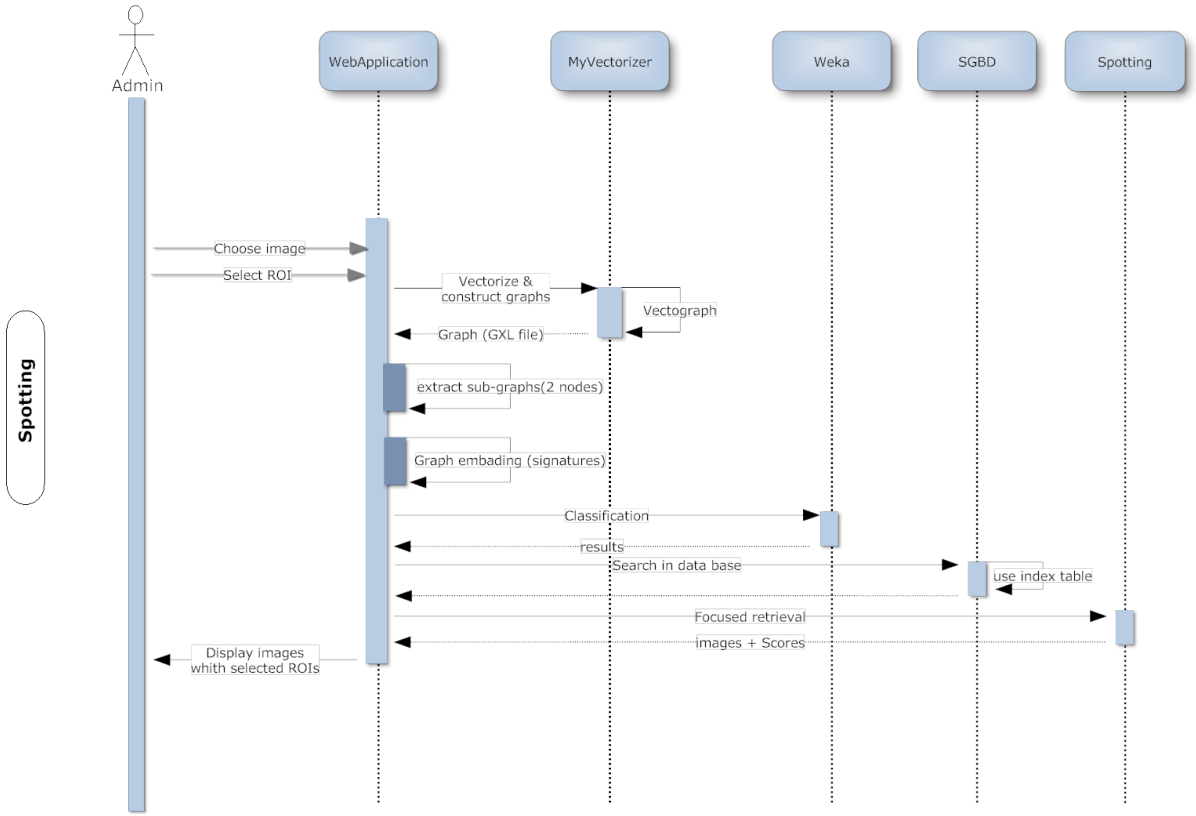


FIGURE 5.4 – Diagramme de séquence du mode « Localisation »

Lors de la phase de la localisation, l'utilisateur commence par choisir une image, puis il sélectionne le logo contenu dans cette image, ensuite le système récupère la zone sélectionnée et il va effectuer les mêmes traitements que la phase d'apprentissage, en commençant par la vectorisation, ensuite l'extraction des cliques d'ordre 2, puis l'encapsulation et enfin il va chercher dans la base de données les signatures qui ressemblent le plus à celles trouvées dans la requête utilisateur pour les imprimer avec une couleur différente dans les documents résultat.

5.3.3 diagramme de classes

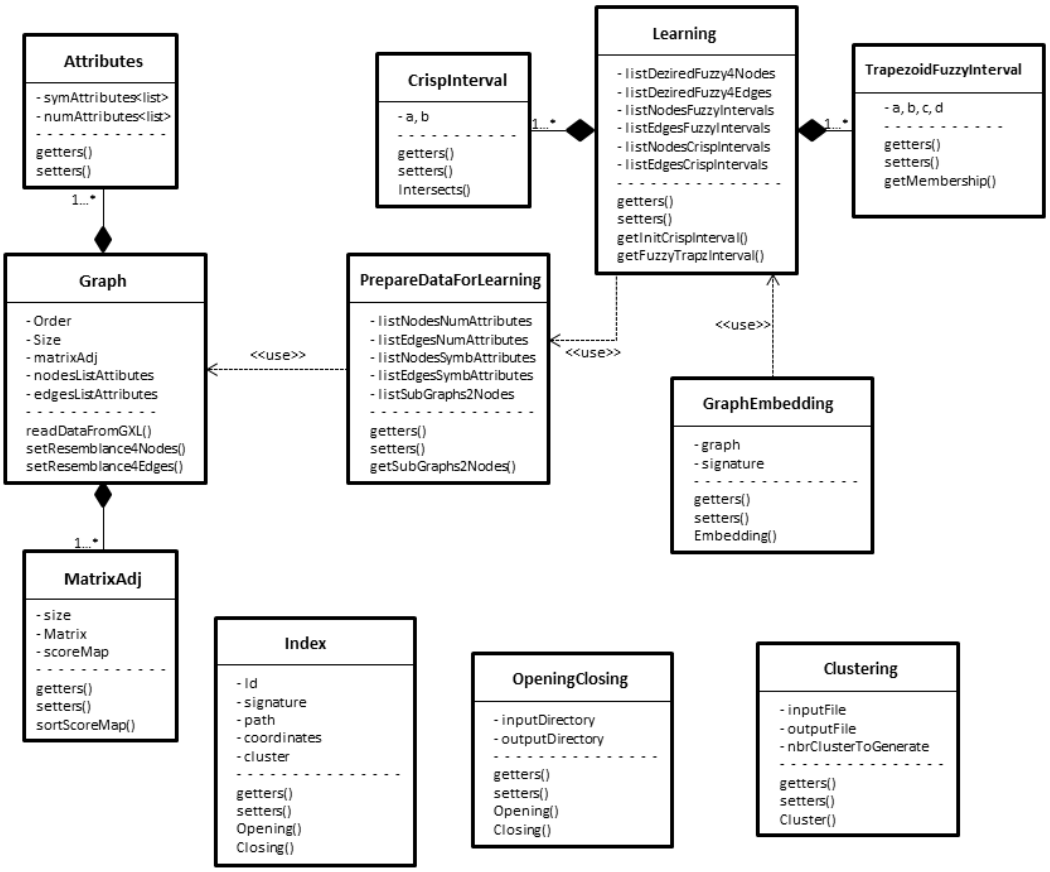


FIGURE 5.5 – Diagramme de classes

Ce diagramme montre les classes les plus importantes du système, dans un premier lieu, on trouve la classe *Graph* qui va représenter la structure contenue dans les fichiers GXL, elle contient principalement des attributs représentant l'ordre, la taille, la matrice d'adjacence et la liste des nœuds et des arcs qui sont eux même composés d'un ou plusieurs objets de type *attributes*, cette classe a comme fonctions, le parseur qui permet la lecture des fichiers GXL et l'initialisation des structures de données, les autres fonctionnalités permettent le calcul des nouveaux attributs de ressemblance pour les nœuds comme pour les arcs. Ensuite la classe *PrepareDataForLearning* qui va avoir comme objectif le parcours des objets *Graph* présents en mémoire afin d'enregistrer leurs attributs numériques et symboliques séparément dans des listes qui, à leurs tours, vont être utilisées dans la phase d'apprentissage. On trouve aussi la classe *Learning* avec des attributs de type *CrispInterval* pour les intervalles discrets, et des attributs de type *TrapezoidFuzzyInterval* pour les intervalles flous, les fonctions permettant l'initialisation de ces intervalles, notamment *GetInitCrispInterval()* et *GetFuzzyOverlapTrapzInterval()* se basent sur les attributs fournis par la classe *PrepareDataForLearning*, après il y a la classe *GraphEmbedding* qui va utiliser les intervalles, préalablement construits, pour encapsuler les graphes dans des Vecteurs Flous de Caractéristiques (VFC), une autre classe nommée *clustering* qui va prendre en charge le groupement des VFCs dans des classes en utilisant comme méthode le *K-means*, on a aussi une classe *OpeningClosing* pour le pré-traitement des images afin d'éliminer le bruit et la dernière classe sera la classe *Index* qui va assurer la sauvegarde dans la base de données les signatures (VFCs) avec leurs graphes, classes et leurs coordonnées.

Gestion de projet

6.1 Cycle de développement

La méthode adoptée pour le déroulement du projet est une approche par méthode agile (Figure 6.1). Ce choix est dû à la nature du projet qui demande un développement par modules. Chaque module sera développé en un ou plusieurs cycles (Sprints). Chaque Sprint commence par une phase de spécification, ensuite le développement, et il se termine par une phase de test. Un livrable est programmé pour la fin de chaque Sprint qui est validé par l ?encadrant lors des réunions.

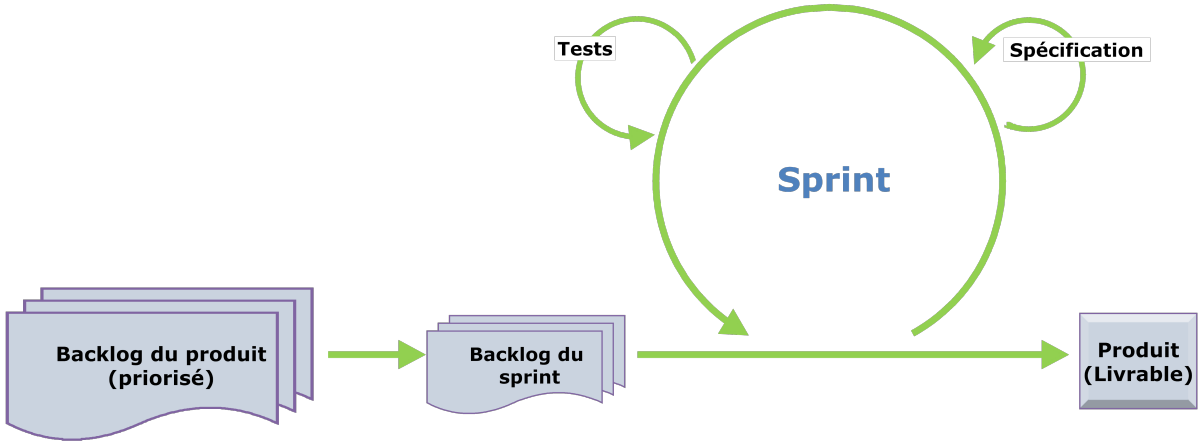


FIGURE 6.1 – Méthodologie de cycle de développement

6.2 Outils de gestion de projet

Les produits Google sont les principaux outils utilisés pour la gestion de ce projet, notamment *Google Docs* pour le partage des documents, *Google Calendar* pour la planification des réunions, *GMail* pour la communication et *Google Wave* pour planification des tâches, mais malheureusement ce dernier a été clôturé par Google, c'est pour cette raison j'ai choisi de basculer vers *Assembla* qui offre gratuitement un environnement de travail complet avec des outils intéressants tels que le dépôt SVN multi-utilisateurs de 200 Mo, un chat avec les membres de l'équipe du projet, un wiki, le Time tracking, File attachements, etc. Un autre outil *MicrosoftProject* est utilisé pour la planification des tâches est la construction du digramme de Gant, ainsi qu'un logiciel qui s'appelle *SmartDraw* pour le dessin des digrammes UML.

6.3 Planning de développement

La tâche de la planification est parmi les tâches les plus difficiles à effectuer, surtout dans le cas d'un type de projet qui est tout à fait nouveau et dont on ignore la charge de travail, notamment lors de la phase de spécification, pendant laquelle, on est amené à découper le projet en tâches, d'estimer le temps nécessaire pour leurs réalisations et en déduire un planning de développement.

FIGURE 6.2 – Outils de gestion de projet

Le premier planning a été effectué le 01/11/2011, lors de la rédaction du cahier de spécification système, ensuite il a été présenté lors de la soutenance de mi-parcours le 18/01/2012 avec une seule modification concernant la tâche de *Pré-traitement d'images* qu'on a déplacé vers la fin du projet à cause du manque d'informations sur les algorithmes à implémenter.
Les figures 6.3 et 6.4 montrent respectivement le planning prévisionnel et le planning respecté.

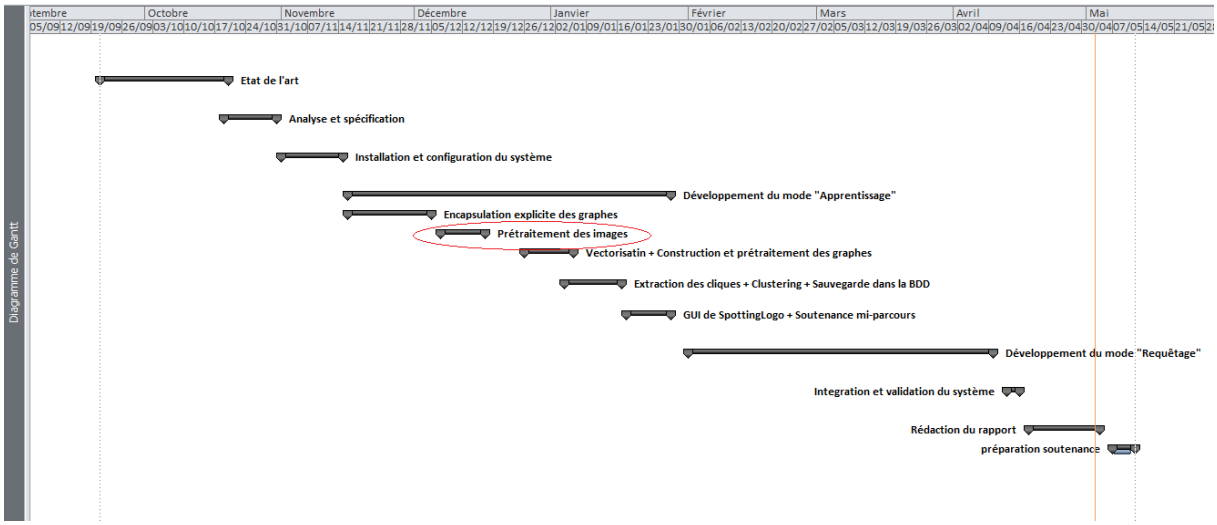


FIGURE 6.3 – Planning prévisionnel

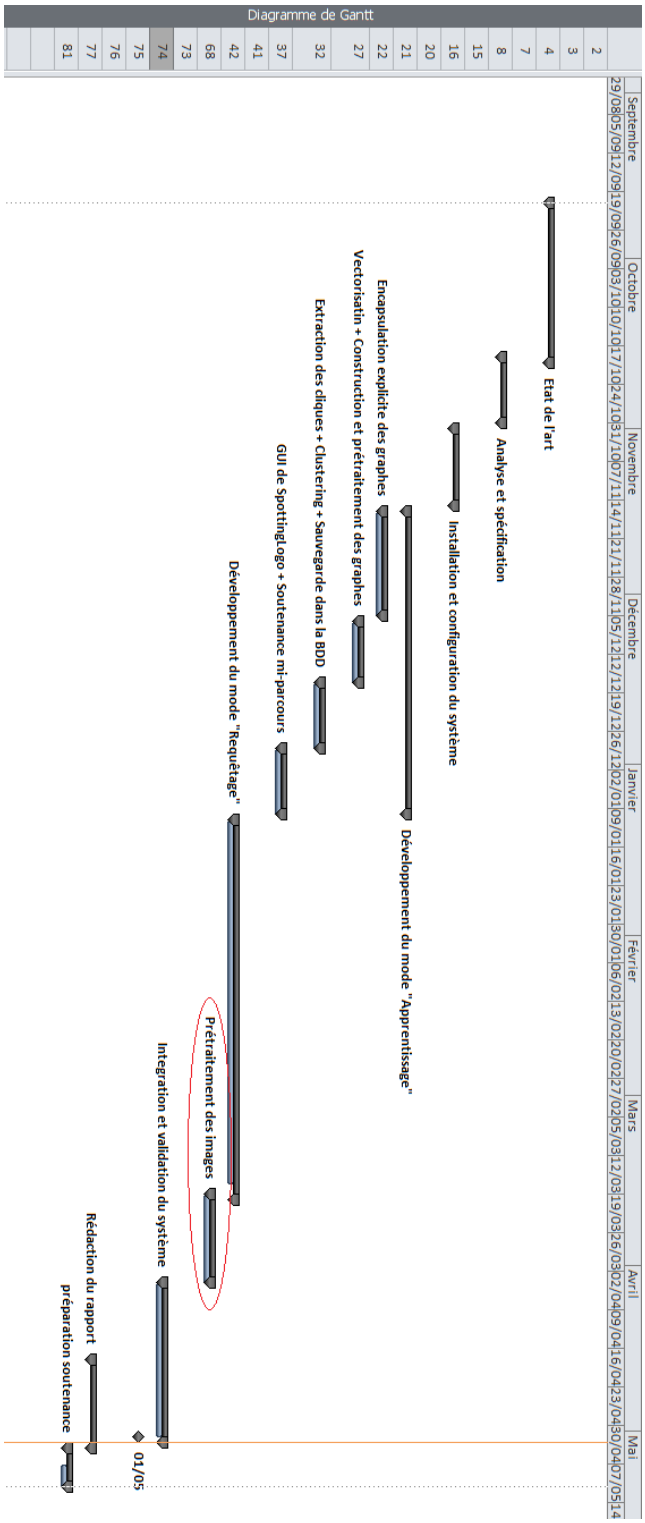


FIGURE 6.4 – Planning respecté

Problèmes rencontrés

Chaque projet, quelle que soit sa nature, a des moments pendant lesquels la rentabilité et la vitesse de développement ne sont pas constantes dans le temps, ce sont des moments où on rencontre des problèmes plus au moins difficiles, soit d'ordre conceptuel ou technique.

Le premier problème rencontré lors de la réalisation de ce projet était le développement du parseur des fichiers GXL, la première version utilisait la librairie *GXL (Graph eXchange Language)* et elle était fonctionnelle sur les premières versions des fichiers qui m'ont été fournis, par la suite, on voulait tester le parseur sur d'autres fichiers, mais malheureusement cela ne marchait pas, après quelques heures de debuggage on s'est redu compte que l'erreur vient de la structure de ces fichiers qui était différente, et pour résoudre ce problème, j'ai créé un autre parseur avec une autre librairie qui s'appelle *SAX (Simple API for XML)* et finalement j'ai obtenu deux types de parseurs adaptés aux différents type de fichiers sur lesquels on a exécuté le système.

Le deuxième problème rencontré était la sauvegarde des structures java tel que l'objet *Learning* qui contient la liste des intervalles construits pendant la phase d'apprentissage non supervisée, et aussi l'objet *KNN* qui représente le classificateur créé lors de la classification et qui sera utilisé dans le mode de localisation, à ce moment-là, on avait le choix entre deux méthodes de sauvegarde, soit un mapping de ces objets dans une base de données, ou la deuxième solution, qui a été adoptée, et qui utilise la sérialisation, ce concept implémenté en Java est facilement utilisable et ne nécessite que deux opérations, une sérialisation qui enregistre un objet présent en mémoire dans un fichier *.ser*, la deuxième opération est la dé-sérialisation qui récupère l'objet à partir d'un fichier. Ce processus de sauvegarde a donné par la suite une certaine indépendance entre les phases du système, par exemple, on peut lancer l'apprentissage non supervisé un jour j, éteindre le système, ensuite le reprendre un autre jour sans être obligé de refaire la phase d'apprentissage, et cela est valable pour toute les phases.

D'autres problèmes ont été rencontrés surtout d'ordre technique par exemple l'intégration d'une librairie développée en C++ dans un environnement Java, cette librairie s'appelle *OpenCV* pour le traitement d'images, dans un premier temps j'ai essayé quelques outils jusqu'à ce que j'ai trouvé *JavaCV* qui est une interface qui fournit l'accès aux fonctionnalités, non seulement celles d'OpenCV, mais aussi celles d'autres librairies de recherche telles que *FFmpeg, libdc1394, PGR,* etc.

Les problèmes rencontrés tout au long de ce projet m'ont permis d'acquérir des nouvelles techniques et des nouvelles méthodologies intéressantes qui vont me permettre de bien gérer, dans le futur, telles situations, et surmonter les défis à venir dans le monde du travail.

Perspectives

Le système développé est fonctionnel et il répond aux besoins du client cités dans le cahier des spécifications, et comme chaque projet est susceptible à des améliorations qui peuvent le rendre plus performant et plus robuste, on peut citer quelques perspectives qui peuvent intéresser celui qui va prendre le relais de développement du système.

Une première perspective concerne la phase de clustering pendant laquelle j'ai utilisé le *K-means* de la librairie *Weka*, cette méthode peut donner des résultats plus intéressants si les paramètres passés, tel le que le nombre de classes sont calculés dynamiquement en s'adaptant à la nature des documents.

Une deuxième perspective serait d'utiliser plusieurs méthodes pour le clustering comme pour la classification et proposer à l'utilisateur une interface de paramétrage avec laquelle il peut configurer le système selon sa collection d'images.

Une autre proposition serait, au lieu d'extraire des cliques d'ordre 2 ensuite les encapsuler dans des vecteurs numériques, on peut extraire des cliques d'ordre plus important ensuite procéder de la même manière afin d'obtenir des vecteurs plus significatifs et plus riches d'informations.

Une autre remarque, que je suppose, pourrait améliorer les résultats, est l'intégration dans les signatures, une autre couche d'informations liées au graphe initial, parce que , au moment de découpage de ce dernier en cliques d'ordre 2, il y a une perte d'informations concernant leurs origines (c'est-à-dire le graphe source). Ces informations numériques peuvent être plus utiles et elles peuvent rendre les signatures plus fidèles aux éléments qu'elles représentent (notamment les graphes).

Conclusion

Ce projet de fin d'études, intitulé *« Spotting automatique des logos dans les images de documents »* s'inscrit dans le domaine de traitement d'images, il traite comme problématique la recherche, dans un ensemble de graphes, les sous-graphes similaires à la requête. Le système accomplit le repérage à travers une technique d'encapsulation dite *Fuzzy Multilevel Graph Embedding* qui produit, pendant la phase d'apprentissage non supervisée hors-ligne, une base d'indexation de la collection des graphes. Ensuite la phase de la localisation se fait en temps réel.

La réalisation d'un tel projet, qui couvre à la fois, le domaine de la recherche et la technique, m'a permis d'améliorer mes capacités et mes réflexions au niveau de la conception comme au niveau de la programmation, d'autant plus qu'il s'agit d'un système qui implémente des méthodes et des approches très récentes issues des travaux de recherche de l'équipe *RFAI*, et en même temps un système qui s'est basé dans son développement sur les nouvelles technologies notamment le *Java EE* et le panier qui vient avec (*Struts, Spring, Hibernate,* etc), sans oublier les situations critiques et les problèmes rencontrés qui ont développé en moi la patience, la responsabilité et l'engagement à fournir un travail qui correspond aux attentes de mon encadrant.

En fin, ce projet était une expérience très intéressante d'autant plus que qu'il ne s'agit pas de la réalisation d'une partie d'un programme ou la participation dans un projet collectif dans lequel on se contente d'exécuter les tâches confiées, mais sa particularité réside dans la prise de responsabilité de toutes les phases quelle soit l'état d'art,la conception, le développement, les tests, ainsi de suite jusqu'au déploiement, et cela n'était pas le fruit d'un travail individuel, mais aussi grâce à la supervision, les idées et les consignes de mon encadrant.

Bibliographie

[1] D. Lewis, G. Agam, S. Argamon, O. Frieder, D. Grossman, and J.Heard, "Tabacco800, building a test collection for complex document information processing," in *Proc. 29th Annual Int. ACM SIGIR Conference*, pp. 665–666, 2006.

[2] MM.Luqman, JY.Ramel, J.Lladós, and T.Brouard, *Fuzzy Multilevel Graph Embedding*. Tours : Laboratoire d'Informatique, Université François Rabelais de Tours, August 18, 2011.

[3] MM.Luqman, T.Brouard, JY.Ramel, and J.Lladós, *Recherche de sous-graphe par encapsulation explicite de graphes : Application à la localisation de contenu dans les images de documents graphiques*. Tours : Laboratoire d'Informatique, Université François Rabelais de Tours, 2011.

[4] MM.Luqman, T.Brouard, JY.Ramel, and J.Lladós, *A Content Spotting System For Line Drawing Graphic Document Images*. International Conference on Pattern Recognition, 2010.

[5] MM.Luqman, T.Brouard, JY.Ramel, and J.Lladós, *Vers une approche floue d'encapsulation de graphes : application à la reconnaissance de symboles*. Colloque International Francophone sur l'Ecrit et le Document, pp. 169-184, 2010.

[6] A. MARKOUCHE, *Content Spotting : recherche de contenu dans les images de documents*. Projet de fin d'études au département informatique de Polytech'Tours, 2010.

[7] Y. AYOUBI, *Apprentissage pour détection des régions d'intérêt dans les graphes*. Projet de fin d'études au département informatique de Polytech'Tours, 2011.

[8] J. . S. . H. . Tutorial, "http://www.scribd.com/doc/25244173/Java-Struts-Hibernate-Tutorial,"

[9] JQuery, "http://docs.jquery.com,"

[10] JavaCV, "http://code.google.com/p/javacv/,"

[11] Weka, "http://weka.wikispaces.com/Use+WEKA+in+your+Java+code,"

[12] GXL, "http://gxl.sourceforge.net/documentation_editor.html,"

Spotting automatique des logos dans les images de documents

Département Informatique
5e année
2011 - 2012

Rapport de Projet de Fin d'Etudes

Résumé : L'objectif de ce PFE est de développer un moteur de spotting de logos contenus dans une collection d'images (Tabacco800). L'application fonctionne sous deux modes, un mode dit « *Apprentissage* » qui analyse et exploite les informations contenues dans les graphes représentant les images et un deuxième dit « *Spotting* » qui, à partir d'une image requête, génère le graphe correspondant qui sera comparé aux autres via une base d'indexation afin de repérer les documents qui contiennent le logo recherché.

Mots clefs : repérage de logo, encapsulation de graphe, groupement, classification, intervalle flou

Abstract: The aim of this project is the development of a system for spotting logos in a collection of documents called Tabacco8000. The application works in two modes, a mode called « *Learning* » which analyzes and processes information in the graphs representing images, and a second mode, called « *Spotting* » that, from an image query, generates the corresponding graph that will be compared to others via an indexation base to retrieve documents that contain the desired logo.

Keywords: spotting logo, graph embedding, clustering, classification, flou interval

---

Encadrants
Muhammed Muzzamil LUQMAN
muhammadmuzzamil.luqman@etu.univ-tours.fr
Jean-Yves RAMEL
ramel@univ-tours.fr
Dimosthenis Karatzas
dimos@cvc.uab.es

Étudiant
Zakaria MELK
zakaria.melk@etu.univ-tours.fr

DI5 2011 - 2012

Université François-Rabelais, Tours