N° d'ordre : 2011-XX-XXX

THÈSE

présentée

à l'Université de Cergy-Pontoise
École Nationale Supérieure de l'Électronique et de ses Applications

pour obtenir le grade de :

Docteur de l'Université de Cergy-Pontoise
Spécialité : Informatique

Par

Justine Lebrun

Équipes d'accueil :
Équipes Traitement des Images et du Signal (ETIS) - UMR 8051

Titre de la thèse :

Appariement inexact de graphes appliqué à recherche d'image et d'objet 3D

Soutenue le 16 mai 2011 devant la commission d'examen

M.	Remy	Mullot	Président
M.	Alain	Rakotomamonjy	Rapporteur
Mme.	Christine	Solnon	Rapportrice
M.	Philippe-Henri	Gosselin	Co-Directeur de thèse
Mme.	Sylvie	Philipp-Foliguet	Directrice de thèse

A ma famille et à Jia Lin

"On se lasse de tout, sauf de comprendre."

Virgile.

Résumé

Les graphes sont des modèles de représentation qui permettent de modéliser un grand nombre de type de documents. Dans cette thèse, nous nous intéressons à leur utilisation pour la recherche dans des bases de données multimédia. Nous commençons par présenter la théorie autour des graphes ainsi qu'un aperçu des méthodes qui ont été proposées pour leur mise en correspondance. Puis, nous nous intéressons plus particulièrement à leur utilisation pour la reconnaissance des formes et l'indexation multimédia. Dans le but de répondre de la manière la plus générique possible aux différents problèmes de recherche, nous proposons de travailler dans le cadre des fonctions noyaux. Ce cadre permet de séparer les problèmes liées à la nature des documents de ceux apportés par les différents types de recherche. Ainsi, toute notre énergie est consacrée à la conception de fonctions de mise en correspondance, mais en gardant à l'esprit qu'elles doivent respecter un certain nombre de propriétés mathématiques. Dans ce cadre, nous proposons de nouvelles solutions qui permettent de mieux répondre aux caractéristiques particulières des graphes issus de primitives et descripteurs visuels. Nous présentons aussi les algorithmes qui permettent d'évaluer rapidement ces fonctions. Enfin, nous présentons des expériences qui mettent en lumière ces différentes caractéristiques, ainsi que des expériences qui montrent les avantages qu'offrent nos modèles vis à vis de la littérature.

Abstract

Graphs are models of representation that can model many type of documents. In this thesis, we focus on their use for research in multimedia databases. We begin by presenting the theory of graphs and around an overview of methods that have been proposed for matching. Then, we are particularly interested in their use for recognition forms and multimedia indexing. In order to respond in the most generic possible different research problems, we propose to work within the framework of kernel functions. This framework allows to separate the problems related to the nature of the documents those introduced by the different types of research. Thus, all our energy is devoted to the design of mapping functions, but bearing in mind that they must meet a number mathematical properties. In this context, we propose new solutions that better meet the specific graphs from primitive and visual descriptors. We also present algorithms to quickly assess these functions. Finally, we present experiments that highlight these different characteristics and experiences that show advantages of our models with respect to the literature.

Remerciements

J'aimerais remercier en premier lieu ma famille et surtout ma mère qui m'ont soutenue pendant ces années de thèse. Je remercie mes encadrants Sylvie et Philippe de m'avoir guidée pendant ces années thèse et apporté une expérience humaine et scientifique qui m'est utile quotidiennement. Je remercie mes amis qui malgré mon faible temps libre de ces dernières années sont toujours là. Je remercie Jia Lin de m'avoir soutenue et encouragée. Je remercie les anciens doctorants et les nouveaux doctorants d'ETIS d'avoir animé le laboratoire de discussions intéressantes et plus particulièrement Sonia, Shuji, David P.,David G,Thomas et Jean-Emmanuel.

Je remercie Fréderic Précioso et Inbar Fijalkow de leurs précieux conseils, Michel Jordan pour m'avoir hébérgée dans son bureau et tous les permanents d'ETIS de m'avoir accueillie, fait profiter des séminaires et des pots de laboratoire.

Je remercie mes rapporteurs de thèse d'avoir accepté la relecture de ce manuscrit.

Les travaux présentés dans ce mémoire ont été réalisés au sein du laboratoire ETIS [1] de l'ENSEA [2].

1. Équipes Traitement de l'Information et des Systèmes
2. École Nationale Supérieure de l'Électronique et de ses Applications

Table des matières

Chapitre 1

Introduction

Présentation du problème

Approche par le contenu

Avec la démocratisation des appareils multimédia, de plus en plus d'images numériques sont générées chaque jour. Le problème de la recherche d'images se retrouve alors dans divers domaines tels que :

- Médical (aide au diagnostic) ;
- Télédétection
- Internet (recherche d'images dans de vastes ensembles) ;
- Robotique (reconnaissance d'un lieu ou d'une situation) ;
- Vidéo (recherche de séquences, de personnes, de lieux, ou d'objets similaires).

Différentes problématiques doivent alors être résolues, du stockage des données jusqu'à la phase d'interrogation du système. Dans la problématique de recherche de documents multimédia, il existe deux grandes familles : les techniques par mots clefs, et celles par le contenu.

La première famille suppose qu'il existe un ensemble de mots clefs associé à chaque document de la base. Des techniques de recherche textuelle sont alors utilisées pour permettre de retrouver l'information. Le principal avantage de cette famille est qu'elle permet d'obtenir de très bons résultats pour peu que la base soit bien annotée.

La deuxième famille, à laquelle nous nous intéressons dans cette thèse, s'appuie directement sur le contenu des documents ; par exemple, pour les images, seul l'ensemble des pixels la constituant est utilisé. L'avantage de cette famille réside dans sa capacité à pouvoir être utilisée sans passer par une phase manuelle d'annotation des images. Il est cependant nécessaire de passer par une phase d'extraction de primitives et caractéristiques visuelles, étant donné que le document dans son état brut est rarement exploitable. Pour le cas des images, on s'intéresse par exemple aux régions ou aux points d'intérêts qui sont décrits par leurs couleurs, gradients et textures. Pour les vidéos, on peut prendre d'autres informations comme le mouvement. Pour les objets en trois dimen-

sions, il est aussi possible d'extraire des régions et points d'intérêts en trois dimensions, qui sont ensuite décrits par des mesures tels que les courbures ou les distances au centre de gravité.

De nombreuses propositions existent pour combiner des techniques issues de ces deux familles. Une approche est de combiner tardivement deux techniques : chacune fonctionne séparément, puis dans une phase finale les résultats sont fusionnés. Une autre possibilité est de choisir un cadre formel commun aux techniques, et qui permet d'utiliser la même méthode de décision finale. Notons que dans le cadre de cette thèse, bien que nous ne nous intéressons pas à l'utilisation conjointe de plusieurs types d'approche, nous avons fait le choix de travailler dans un cadre formel qui le permet.

Types de recherche

Il existe différentes manières d'effectuer des recherches. Une des plus connues est la recherche de cible. Par exemple, le document recherché est connu, mais sa localisation ne l'est pas. Dans ce cas, le système est interrogé à l'aide d'un morceau ou d'un croquis du document, puis une similarité est calculée entre la requête et les documents de la base. Un type de recherche similaire est celui de la recherche de copie, où l'objectif est de retrouver tous les documents qui sont une déformation d'un document original.

Un autre type de recherche est la recherche d'objets ou de catégories. Dans cette situation, l'utilisateur souhaite retrouver un ensemble de documents avec des propriétés communes. Par exemple, l'ensemble des images qui contiennent une voiture, ou l'ensemble des vases en trois dimensions. Ce type de recherche est généralement plus difficile étant donné qu'il n'existe pas nécessairement de corrélations visuelles élémentaires qui regroupent les documents. Par conséquent, une solution courante est d'utiliser des techniques d'apprentissage qui vont effectuer un tri ou une classification de la base en fonction d'exemples ou de paramètres fournis par l'utilisateur.

Il existe aussi différentes manières d'interroger le système. La plus simple est de former une unique requête, à la manière de Google qui à quelques mots clefs fait correspondre un résultat. Dans le cas des approches par le contenu, nous retrouvons aussi des requêtes simples, comme la recherche de copie à partir d'un document original. Pour des problèmes plus complexes, il est nécessaire de former des requêtes plus riches, comme un ensemble d'exemples et de contre-exemples.

Ce dernier cas peut être particulièrement contraignant lorsque plusieurs milliers d'exemples pris au hasard sont nécessaires pour obtenir un bon résultat. Dans le but de résoudre ce problème, une alternative consiste à interagir avec le système de recherche. L'objectif est alors de permettre à l'utilisateur d'enrichir ou raffiner sa requête en fonction des résultats. De nombreux types d'interaction sont possibles via des interfaces graphiques. Par exemple, les techniques de "browsing" permettent à l'utilisateur de se "déplacer" dans l'ensemble des solutions. D'autres techniques posent des questions à l'utilisateur. Ces techniques déterminent les questions qui permettent d'améliorer au mieux le résultat. Les questions posées ne sont pas nécessairement textuelles, et peuvent être graphiques. Par exemple, le système peut demander si un document est pertinent ou non, ou si un document est plus intéressant qu'un autre.

Modèle général

Dans cette thèse, nous avons choisi de nous intéresser à des modèles généralistes, qui peuvent s'adapter à plusieurs types de document et de recherche.

Tout d'abord, nous avons fait le choix de nous intéresser aux modèles de représentation des documents par graphes. Un graphe représente l'ensemble des primitives du document (régions, points d'intérêts, ...) sous la forme de sommets (décrits par leurs caractéristiques visuelles) et d'arêtes (décrites par la nature des relations entre les primitives). Ce type de modèle permet de représenter différents types de média, comme les images, les vidéos et les objets en trois dimensions. Ainsi, nous tentons de répondre à différents problèmes avec les mêmes modèles. La figure 1 montre un exemple d'une image segmentée en régions (ici des régions floues) puis représentée par un graphe.

Notons toutefois que nous nous intéressons à des graphes valués où les sommets et les arêtes portent aussi des informations numériques et pas seulement symboliques. En effet, les descripteurs visuels ne sont généralement pas à valeur dans un ensemble fini, et leur signification sémantique est toute relative. Ce dernier point est très prononcé dans le cas des bases généralistes, au contraire des bases spécialisées, comme les bases d'empreintes digitales par exemple. Ce point est très important pour la conception des solutions ; nous y reviendrons à plusieurs reprises pour expliquer la raison de certains choix.

Ensuite, nous avons fait le choix de travailler dans le cadre formel des fonctions noyaux. Ce cadre, que nous décrivons plus en détails dans les chapitres suivants, permet en quelques mots de séparer les problèmes liés à la nature des documents de ceux apportés par les problèmes d'apprentissage. Ainsi, nous concentrons toute notre énergie dans la production de modèles pour la comparaison de graphes, qui peuvent par la suite être utilisés par les nombreuses méthodes à noyau, du clustering à la classification en passant par le browsing.

Notons toutefois que, dans le but d'évaluer les modèles, nous nous sommes surtout intéressés à la recherche interactive. Nous suivons un modèle général assez classique, dont un résumé est présenté dans la figure 1.1. Nous trouvons deux grandes phases. Une première phase dite hors-ligne est effectuée à l'initialisation de la base, puis éventuellement mise à jour si de nouveaux documents sont ajoutés à la base. Lors de cette phase, le maximum de pré-calculs sont effectués, comme l'extraction des primitives et caractéristiques visuelles, la création des graphes et le stockage sur disque. Une fois ceci fait, on peut passer à la seconde phase dite en-ligne qui se répète pour chaque session de recherche. Dans ce scénario, un utilisateur vient avec une requête initiale qui est utilisée pour donner un premier résultat (sous la forme d'un tri de la base). Puis, le système sélectionne des questions à poser à l'utilisateur. Dans notre cas, les questions sont des images de l'utilisateur et la réponse de celui-ci consiste en une annotation de ces images. Une fois ceci fait, le résultat peut être mis à jour, et de nouvelles questions sont sélectionnées. Ce processus est répété jusqu'à satisfaction, ou stoppé si l'utilisateur est las.

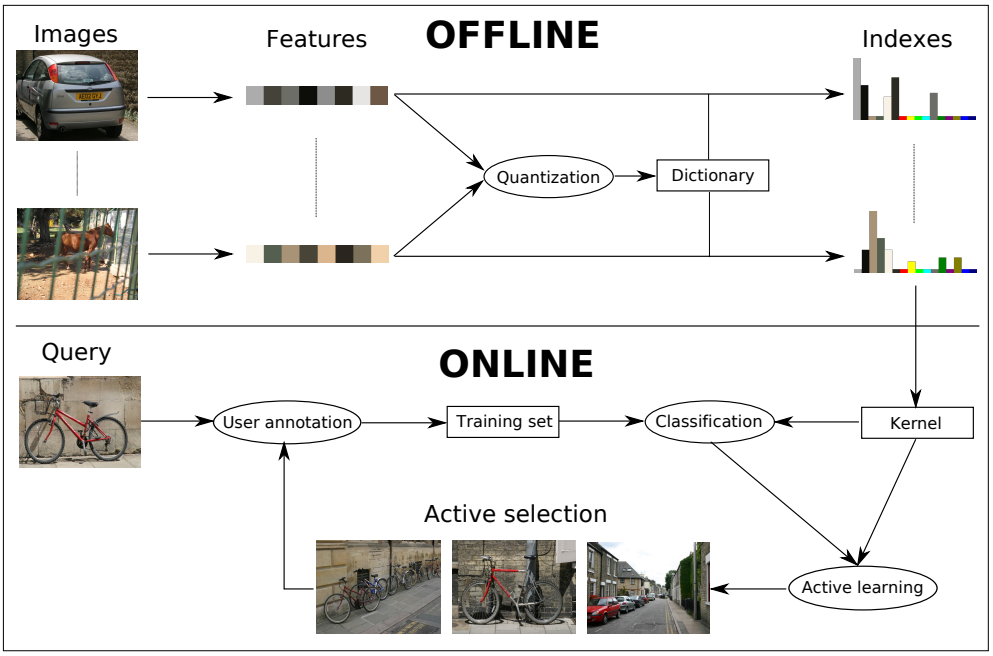
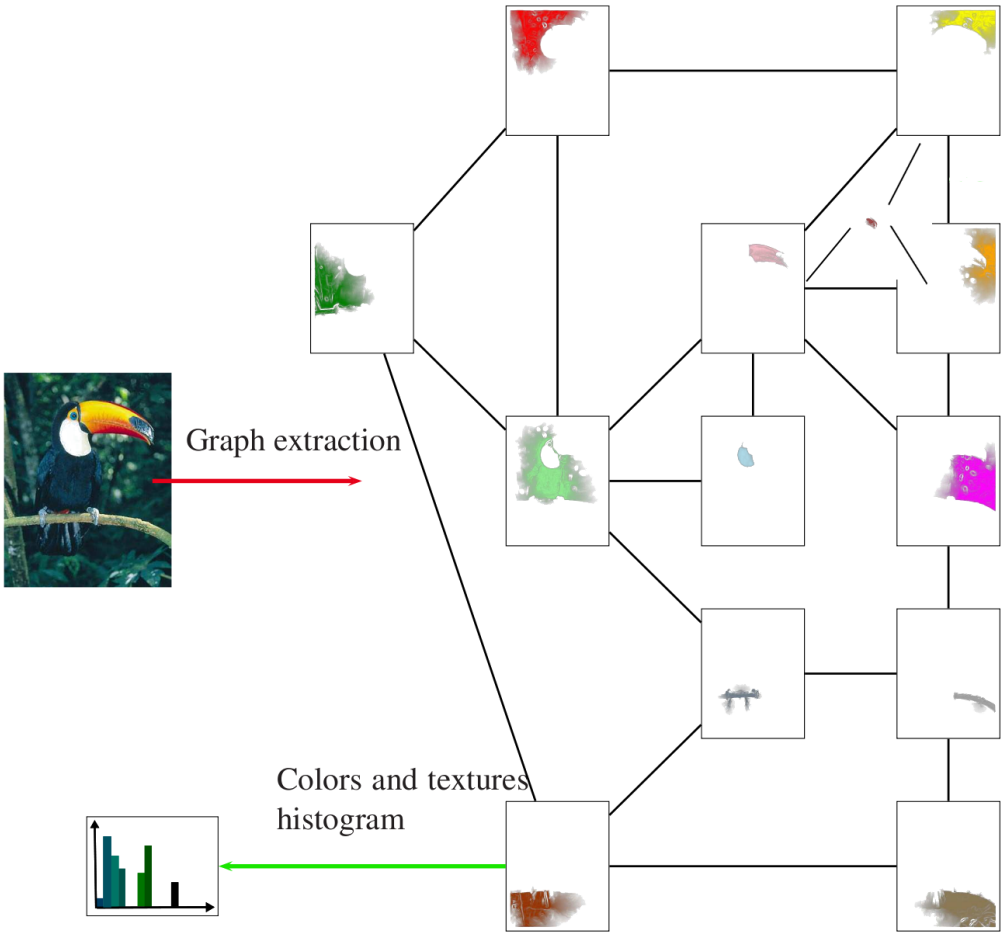


FIGURE 1.1 – Architecture pour la recherche interactive de documents

(a) (b)



(c)

FIGURE 1.2 –

Plan de thèse et contributions

Mise en correspondance de graphes

Nous démarrons ce mémoire par un rappel des notations et notions autour des graphes qui seront utilisées dans les différents chapitres. Puis, nous présentons une vue d'ensemble des techniques basées sur les graphes pour la reconnaissance des formes et l'indexation des documents. Un très grand nombre de méthodes sont proposées dans la littérature à ce sujet. Il existe cependant un certain nombre de constantes qui permettent de classifier les méthodes. De ces classes naissent beaucoup de ramifications, qui se distinguent souvent par les améliorations apportées et le type de problématique auxquelles elles s'attaquent. Enfin, nous concluons ce chapitre sur une présentation des différentes raisons qui nous poussent à nous orienter vers le modèle général que nous considérons. Ce modèle vise à traiter les différentes particularités des graphes issus de primitives et descripteurs visuels, avec autant de généricité qu'il est possible.

Noyaux sur graphes

Nous poursuivons ce mémoire en présentant un rappel sur le cadre formel des fonctions noyaux. Nous y présentons les définitions et notations, ainsi que ses différents avantages, puis, un aperçu des techniques basées sur les graphes qui ont été proposées dans ce cadre en insistant sur des méthodes basées sur des noyaux de chemins aléatoires. Nous revenons plus en détails sur deux méthodes qui sont à la base de celles que nous proposons.

Nous présentons ensuite une famille de fonctions noyaux que nous proposons pour travailler sur les graphes issus de primitives et descripteurs visuels. Cette famille s'inscrit dans celle des fonctions sur sacs de chemins aléatoires parcourant les graphes. L'idée générale est de s'intéresser aux fonctions qui somment les similarités entre les chemins mis en correspondance entre deux graphes comparés. Dans le but de mieux gérer ces similarités si particulières, nous proposons de ne sommer que les similarités les plus pertinentes. Notons que nous présentons aussi une fonction noyau particulière qui nous semble être la plus adaptée à cette problématique. Nous proposons aussi d'autres nouvelles fonctions qui permettent de comparer les chemins entre eux.

Algorithmes

Ce chapitre est consacré aux algorithmes qui permettent de calculer rapidement les similarités de graphes. D'une manière générale, nous nous intéressons aux méthodes qui représentent le calcul sous la forme d'un arbre d'appariement. La recherche des solutions dans cet arbre est alors effectuée à l'aide de l'algorithme du branch-and-bound. Nous présentons et justifions les différentes hypothèses que doivent vérifier les fonctions noyaux pour pouvoir utiliser ces algorithmes.

Expériences

Enfin, nous présentons dans un dernier chapitre les expériences que nous avons menées en fonction de la nature des graphes que nous souhaitons classifier. En effet, nous verrons que ces derniers possèdent des caractéristiques qu'il faut gérer au mieux, et qui sont à la base de certains choix que nous avons faits dans nos modèles. D'autres expériences s'intéressent à l'influence des différents choix en matière de paramétrage. Ainsi, certains paramètres ont une influence énorme sur la qualité des résultats, alors que d'autres produiront des résultats beaucoup plus stables. Enfin, les dernières expériences présentent une vue d'ensemble de la qualité des recherches, obtenue en utilisant des méthodes de la littérature ainsi que celles que nous proposons.

Publications de l'auteur

REVUES INTERNATIONALES

1. J.Lebrun, Philippe-Henri Gosselin, Sylvie Philipp-Foliguet *Inexact graph matching for online object retrieval in image databases*, en révision majeure à IVC.

CONFÉRENCES INTERNATIONALES

5. J. Lebrun, S. Philipp-Foliguet, P.-H. Gosselin, *Image retrieval with graph kernel on regions*, 19th ICPR International Conference on Pattern Recognition - dec 2008.
6. J.-E. Haugeard, S. Philipp-Foliguet, F. Precioso, J. Lebrun *Extraction of Windows in facade using Kernel on Graph of Contours*, 16th Scandinavian Conference on Image Analysis, Volume 5575, page 646–656 - June 2009 .

CONFÉRENCES NATIONALES

8. P.-H. Gosselin,J.Lebrun, S. Philipp-Foliguet, *Recherche d'images par noyaux sur graphes de régions*, 8èmes journées francophones Extraction et Gestion des Connaissances (EGC) - Jan 29th - Fev 1st 2008.

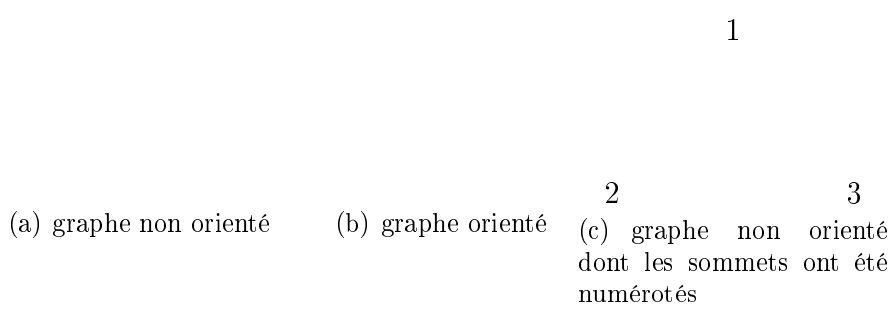
Chapitre 2

Mise en correspondance de Graphes

Les graphes sont des outils mathématiques utilisés dans de nombreux domaines pour modéliser les problèmes à résoudre. S'il l'on devait donner une origine à cette théorie, ce serait celle introduite par Euler au 18ème siècle avec le problème des ponts de Königsberg. Depuis, elle a donné naissance à de multiples techniques dans de nombreux domaines, en mathématique, automatique, algorithmique, biologie, chimie, etc. et plus récemment en reconnaissance des formes.

Cette théorie a deux principaux objectifs. Le premier est d'offrir un modèle pour représenter le problème, généralement l'ensemble des possibilités. Dans notre contexte, le problème est la mise en correspondance d'éléments dans les images. Dans ce cas, le graphe est une représentation des différents éléments et de leurs relations binaires entre eux, par exemple l'ensemble des régions de l'image et les relations d'adjacence. Le deuxième objectif de cette théorie est d'offrir des outils pour permettre de trouver les meilleures possibilités dans le but de résoudre un problème. Dans notre contexte, nous nous intéressons aux outils qui permettent de comparer deux ensembles de possibilités (i.e. les graphes). Par exemple, rechercher les combinaisons de régions les plus similaires entre deux images.

Dans ce chapitre, nous commencerons par présenter les différentes définitions et notations qui sont nécessaires pour la bonne compréhension des techniques présentées dans cette thèse. Puis, nous présenterons deux grandes familles de méthodes de comparaison de graphes. Enfin, nous nous appuierons sur ces présentations afin de motiver nos choix pour la recherche d'images.



(a) graphe non orienté (b) graphe orienté (c) graphe non orienté dont les sommets ont été numérotés

FIGURE 2.1 – Exemples de différents types de graphe

2.1 Théorie des Graphes

2.1.1 Définitions

Nous présentons dans cette section des définitions et notations couramment utilisées dans la littérature sur la base des travaux de Berge[1].

Définition 2.1.1. ***Graphe non orienté.*** *Un graphe non orienté $G = (V, E)$ est constitué d'un ensemble de sommets V et d'un ensemble d'arêtes $E \subseteq V \times V$. Une arête est une paire non ordonnée de deux sommets.*
Un exemple de graphe non orienté est présenté en Figure 2.1(a).

Définition 2.1.2. ***Graphe orienté.*** *Un graphe orienté $G = (V, E)$ est constitué d'un ensemble de sommets V et d'un ensemble d'arcs $E \subseteq V \times V$. Un arc est un couple ordonné de deux sommets.*
Un exemple de graphe orienté est présenté en Figure 2.1(b).

Les arcs d'un graphe orienté, sont représentés sous forme de flèche et ne permettent le déplacement que dans un seul sens. On les appelle arcs pour les distinguer des arêtes non orientées. On utilise les parenthèses (,) pour écrire le couple de sommet.

Définition 2.1.3. ***Ordre.*** *L'ordre d'un graphe G est le nombre de sommets de ce graphe.* L'ordre du graphe de la Figure 2.1(b) est 3.

Définition 2.1.4. ***Complétude.*** *Un graphe est complet si tous ses sommets sont reliés deux à deux.*

Définition 2.1.5. ***Boucle.*** *Une arête ou un arc (v,v) est appelée boucle.*
L'arête en bas à gauche du graphe en Figure 2.1(b) est une boucle.

Définition 2.1.6. ***Adjacence.*** *Deux sommets sont dits adjacents s'il existe dans le graphe une arête (ou un arc) les reliant.*
Les sommets 1 et 2 de la figure 2.1(c) sont adjacents.

Définition 2.1.7. ***Chaîne.*** *Une chaîne est définie par une suite de sommets adjacents.*
La suite de sommets $(2, 1, 3)$ du graphe de la figure 2.1(c) constitue une chaîne. La chaîne $(3, 1, 2)$ est une écriture différente de la même chaîne.

Définition 2.1.8. ***Chemin.*** *Un chemin est une chaîne orientée.*
Les chemins $h = (2, 1, 3)$ et $h' = (3, 1, 2)$ du graphe de la figure 2.1(c) sont deux chemins différents. On notera $|h|$ la longueur d'un chemin h, i.e son nombre d'arêtes, par exemple $|h'| = 2$.

Définition 2.1.9. ***Cycle.*** *Un cycle est une chaîne dont les deux sommets extrémités sont identiques.*
La chaîne $h = (2, 1, 3, 2)$ et $h' = (2, 1, 2)$ sont des cycles.

Définition 2.1.10. ***Connexité.*** *Un graphe est connexe lorsque, pour tout couple de sommets, il existe une chaîne qui les relie.*
Les graphes des figures 2.1.1 sont connexes.

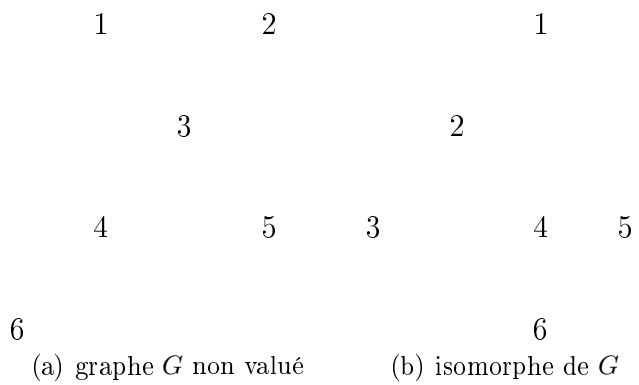
Définition 2.1.11. ***Arbre.*** *Un arbre est un graphe non orienté, connexe et sans cycle.*
Les graphes des figures 2.1(b) et 2.1(c) sont des arbres.

Définition 2.1.12. ***Graphe valué (ou étiqueté).*** *Les graphes valués ont leurs sommets ou leur arêtes qui portent de l'information.*

Les sommets ou arêtes ainsi étiquetés peuvent être différenciés par l'information qu'ils portent. Une étiquette peut être un nombre, un vecteur, un symbole, une image, etc. Notons que les sommets des graphes peuvent être numérotés pour faciliter la manipulation des graphes, mais cela ne constitue pas une étiquette pour autant. On peut changer ou permuter à loisir ce type d'étiquettes. G' de la fig.2.2(c) est valué alors que G de la fig 2.2(a) ne l'est pas.

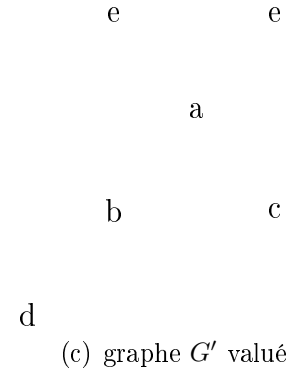
Définition 2.1.13. ***Isomorphisme.*** *Deux graphes sont isomorphes s'ils ont la même structure : $G = (V, E)$ et $G' = (V', E')$ sont isomorphes ssi ils ont le même ordre $|V| = |V'|$ et s'il existe une bijection $f : V \to V'$ t.q. $(u, v) \in E$ ssi $(f(u), f(v)) \in E'$. Autrement dit, ces deux graphes sont identiques quelle que soit la façon dont ils sont dessinés. Plus formellement, cela signifie qu'il existe un morphisme bijectif entre les deux graphes.*

L'isomorphisme dans les graphes étiquetés est contraint par la ressemblance des sommets entre eux. Les figures 2.2(a) et 2.2(b) montrent des graphes non valués isomorphes (rappelons que les numéros sont figuratifs). Les figures 2.2(c) et 2.2(d) montrent des graphes valués isomorphes. Les graphes des figures 2.2(e) et 2.2(f) montrent des graphes valués non isomorphes à ceux des figures 2.2(c) et 2.2(d). Les graphes isormorphes d'un graphe étiqueté sont moins nombreux que ceux des graphes non étiquetés. La détection de l'isomorphisme entre deux graphes étiquetés est de ce fait plus aisée que celle des graphes non porteurs d'information.

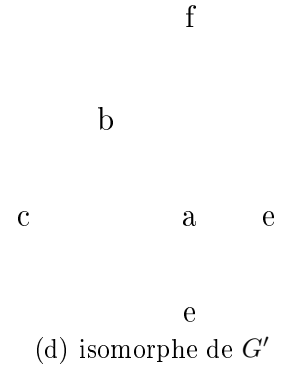


(a) graphe G non valué

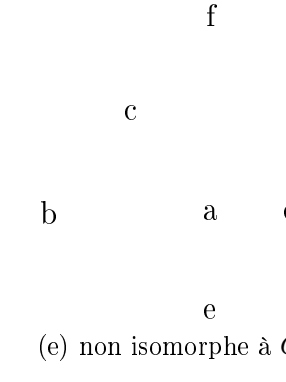
(b) isomorphe de G



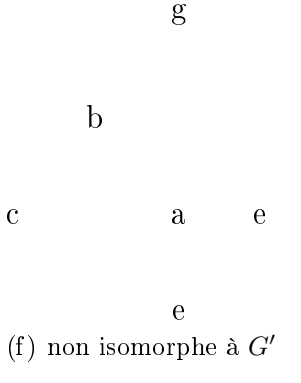
(c) graphe G' valué



(d) isomorphe de G'



(e) non isomorphe à G'



(f) non isomorphe à G'

FIGURE 2.2 – Exemples de graphes et illustration des isomorphes.

Taille chemin	$H_{\Omega l}$	H_{El}	$H_{\bar{c}l}$	H_{Ω}	H_{E}	$H_{\bar{c}}$
$\|h\| = 0$	5	5	5	5	5	5
$\|h\| = 1$	25	25	20	20	20	20
$\|h\| = 2$	125	100	100	90	60	60
$\|h\| = 3$	625	380	320	320	180	120

FIGURE 2.3 – Nombre de chemins dans un graphe complet d'ordre 5 en fonction de différentes génératrices.

2.1.2 Ensemble de chemins

Les fonctions de similarités de graphes que nous aborderons au chapitre 3, se calculent à partir de similarités de chemins appartenant aux graphes considérés. On va s'intéresser à des ensembles particuliers des chemins qui peuvent être extraits d'un graphe. Pour définir ces ensembles de chemins, nous introduisons la notion de fonction génératrice.

Définition 2.1.14. ***Génératrice.*** *Une fonction génératrice de chemin H est une fonction qui à un graphe G fait correspondre un ensemble de chemins issus de ce graphe.*

Voici différents types d'ensembles de chemins que l'on peut tracer sur un graphe G, couramment utilisés dans la littérature :

- $H_{\Omega}(G)$: tous les chemins sans boucles ;
- $H_{\Omega l}(G)$: tous les chemins avec ou sans boucles ;
- $H_{E}(G)$: chemins Eulériens : tous les chemins qui passent une et une seule fois par chaque arête ;
- $H_{El}(G)$: chemins Eulériens avec boucles ;
- $H_{\bar{c}}(G)$: tous les chemins qui ne sont pas des cycles et sans boucles ;
- $H_{\bar{c}l}(G)$: tous les chemins qui ne sont pas des cycles.

Le choix de la génératrice produit des ensembles de chemins plus ou moins grands, et a une incidence très importante sur la comparaison des graphes ainsi que sur les temps de calculs. Dans le but d'illustrer ces différences, nous présentons dans la Figure. 2.3 le nombre de chemins de longueur comprise entre 0 et 3, pour chaque type d'ensembles de chemins sur un graphe complet d'ordre 5.

Dans certaines situations, comme avec les algorithmes rapides, nous aurons aussi besoin de disposer d'ensembles de chemin récursifs. Nous introduisons la fonction génératrice récursive.

Définition 2.1.15. ***Génératrice récursive.*** *Une fonction génératrice H est récursive si pour tout graphe G elle produit des chemins récursifs. Un chemin h est dit récursif si il est composé d'un seul sommet, ou bien s'il existe un chemin $h' \in H(G)$, un sommet $v \in V$ et une arête $e \in E$ reliant h' et v tels que $h = h'ev$. Autrement dit s'il est le prolongement d'un chemin $h' \in H(G)$ et qu'il est lui aussi dans $H(G)$.*

Les génératrices H_E,$H_{\Omega l}$ produisent des ensembles de chemins récursifs.

2.2 Comparaison exacte

Dans cette section, nous présentons différentes techniques proposées dans la littérature pour comparer deux graphes G et G' de manière exacte. Une première forme de comparaison exacte est de déterminer si les deux graphes sont isomorphes. Plus précisément, on cherche à déterminer s'il existe un morphisme f entre les deux graphes tel que, pour toute chaîne $v_1 e v_2$ dans G, il existe une unique image $v'_1 e' v'_2$ dans G' via f.

Une forme moins contrainte de comparaison consiste à rechercher les isomorphismes entre les sous-graphes des deux graphes. D'autres encore moins contraintes vont en relâcher certaines, comme la nécessité de mettre en correspondance chaque nœud d'un graphe avec un et un seul nœud de l'autre graphe.

Ces différents problèmes sont presque tous NP-difficiles (pour certains cas cela reste à déterminer), et ont par conséquent des temps de résolution exponentiels de manière générale. Il reste cependant possible de réduire cette complexité, par exemple en ne considérant que certaines catégories de graphes. Or en se limitant à des graphes dont le nombre de sommets est faible, les algorithmes peuvent être lancés pour un temps raisonnable compte tenu de la puissance des machines actuelles.

2.2.1 Arbre de recherche

La plupart des algorithmes de comparaison exacte sont basées sur une représentation par arbre de recherche. L'idée est de représenter chaque mise en correspondance sous la forme d'une chaîne dans un arbre de recherche. Pour ce faire, on représente chaque mise en correspondance entre deux sommets sous la forme d'un nœud dans l'arbre de recherche, et chaque mise en correspondance entre deux arêtes sous la forme d'un lien entre deux nœuds de l'arbre de recherche. Cela permet de diviser les calculs de mise en correspondance dans le but d'éviter de refaire plusieurs fois le même calcul. Cela permet aussi d'éviter certains calculs en élaguant certaines branches de l'arbre. Notons qu'une des conditions pour obtenir une solution exacte est que toutes les solutions soient atteignables en explorant l'arbre de recherche. Les algorithmes sont généralement basés sur des recherches en profondeur d'abord ou sur le branch-and-bound.

L'une des premières méthodes qui a rencontré un franc succès dans la littérature est celle d'Ullman [2]. Cette méthode peut résoudre les problèmes d'isomorphisme et d'isomorphisme de sous-graphe. Des améliorations ont été proposées, comme [3], mais avec un coût en mémoire qui limite leur usage aux petits graphes. Des propositions plus récentes utilisent des heuristiques plus efficaces[4, 5], ainsi qu'une utilisation mémoire linéaire avec le nombre de sommets dans le graphe, ce qui permet de traiter des graphes de plus grande taille [6].

Certaines méthodes s'appuient sur la notion de distance de graphe, comme [7, 8]. L'idée est de calculer l'heuristique du branch-and-bound en fonction du nombre d'opérations d'insertion ou de suppression qui permettent de rendre un graphe isomorphe à un autre. Une bonne partie de la complexité réside alors dans le calcul de cette édition de graphes, dont des méthodes rapides ont été proposées récemment, comme [9] qui proposent d'utiliser les graphes biparties pour représenter ce calcul.

Enfin, notons que des propositions ont aussi été faites pour paralléliser les calculs [10] et plus particulièrement [11] qui propose une parallélisation du branch-and-bound que nous utilisons dans cette thèse. Ces techniques sont plus que d'actualité compte tenu du nombre grandissant de cœurs au sein des processeurs actuels.

2.2.2 Autres techniques

D'autres techniques de comparaison de graphes se basent sur des outils mathématiques, comme la théorie des groupes. Un algorithme très populaire pour sa rapidité est celui de McKay[12]. Cet algorithme, qui permet de déterminer l'isomorphisme entre deux graphes, est basé sur un autre algorithme qui permet d'obtenir une représentation canonique des graphes. Déterminer si deux graphes sont isomorphes revient donc à comparer les deux représentations canoniques. Ce type de méthode est par conséquent très intéressant pour comparer un nouveau graphe avec un grand nombre de graphes dont les représentations canoniques ont été pré-calculées.

L'idée d'effectuer des pré-calculs sur un ensemble de graphes a ensuite été reprise dans plusieurs méthodes. Par exemple, Messmer [13, 14] forme un dictionnaire de sous-graphes prototypes à partir d'une base de graphes. Puis, il exprime chaque graphe en fonction de ces prototypes, et propose un algorithme qui évitera de comparer plusieurs fois les mêmes sous-graphes. Des améliorations seront proposées par la suite en utilisant des arbres de décision [15].

2.3 Comparaison inexacte

L'objectif des méthodes de comparaison inexacte est de trouver une solution satisfaisante pour un critère donné. Le critère le plus utilisé est naturellement l'erreur commise vis à vis de la solution optimale, mais d'autres critères sont aussi utilisés comme un temps de calcul maximal, ou des combinaisons de différents critères.

Cette approche a deux principaux intérêts. Le premier est qu'elle permet de trouver une solution intéressante en un temps raisonnable, et par conséquent de traiter des graphes de plus grande taille [7]. Le deuxième est qu'elle permet de résoudre des problèmes bruités, dont la solution exacte ne peut être identifiée clairement. Ces méthodes sont plus souples, et permettent de trouver des correspondances partielles là où les méthodes exactes ne trouvent aucune solution.

Ce type de comparaison est par conséquent bien adapté aux images, dont les graphes issus des primitives et descripteurs visuels forment un modèle imparfait.

2.3.1 Arbre de Recherche

Les arbres de recherche peuvent aussi être utilisés pour la comparaison inexacte de graphes. Dans ce cas, la recherche est dirigée par le coût de la mise en correspondance obtenue jusqu'à présent, et par une heuristique qui estime le coût des futures mises

en correspondance. Cette information est utilisée pour élaguer l'arbre mais aussi pour déterminer l'ordre de traitement des nœuds de l'arbre.

Beaucoup de méthodes vont ainsi se distinguer par le choix de la fonction coût et l'heuristique associée. Parmi celles-ci on retrouve assez souvent la notion de distance d'édition de graphes[16]. L'idée est de calculer un coût en fonction du nombre d'opérations d'insertion ou de suppression qui permettent de rendre un graphe isomorphe à un autre. Autour de cette idée, de nombreuses techniques sont proposées [17, 18, 19].

Une autre distance de graphe utilisée pour estimer le coût est celle basée sur la mise en correspondance par graphe bipartie [20]. L'idée ici est de représenter le calcul de distance sous la forme d'un graphe bipartie. On recherche alors la meilleure mise en correspondance entre deux jeux de sommets, avec la contrainte que chaque sommet ne peut être mis en correspondance qu'une et une seule fois. Ce problème peut être résolu rapidement à l'aide d'un algorithme $A^\star$. Notons aussi que des propositions plus récentes permettent de déterminer les valeurs de l'heuristique très rapidement [21, 22].

Des méthodes pour paralléliser les calculs on aussi été proposées dans ce domaine, comme [23], qui propose une version parallélisée du branch-and-bound pour calculer une distance de graphe.

2.3.2 Méthodes spectrales

Les méthode spectrales s'appuient sur les vecteurs et valeurs propres de la matrice d'adjacence des graphes pour les comparer. L'intérêt d'une telle approche repose sur le fait que les vecteurs et valeurs propres sont invariants aux permutations des sommets du graphe. Ainsi, deux graphes isomorphes ont le même spectre. Notons toutefois que la réciproque n'est pas vraie, deux graphes de même spectre ne sont pas nécessairement isomorphes. En terme de complexité, cette approche est intéressante puisque le principal calcul réside dans la détermination du spectre, qui peut être pré-calculé.

Les premières méthodes basées sur ce principe sont assez contraintes, par exemple la méthode de [24] ne fonctionne que sur des graphes de même nombre de sommets et tous les sommets doivent être mis en correspondance deux à deux. Des méthodes ont été proposées par la suite pour réduire ces problèmes [25]. Une autre catégorie de méthodes vont s'appuyer sur le spectre pour effectuer un clustering des sommets. Par exemple, [26] utilise ces clusters pour effecteur une mise en correspondance hiérarchique. Un premier niveau compare les clusters, puis une mise en correspondance plus fine est effectuée au sein des clusters. La méthode de [27] construit un espace vectoriel défini par les vecteurs propres de la matrice d'adjacence, puis projette les sommets dans cet espace. Un clustering est alors utilisé pour trouver les sommets à mettre en correspondance.

Des méthodes proposées dans le domaine de la vision par ordinateur s'appuient sur des techniques spectrales pour trouver les mises en correspondance. Dans ce cas, des matrices plus généralistes que celle d'adjacence sont considérées,comme par exemple la matrice des similarités entre tout élément structurel du graphe. Le principal vecteur propre de cette matrice est alors utilisé pour déterminer les éléments mis en correspondance. Parmi ces méthodes, nous pouvons citer [28] qui permet la mise en correspondance entre points d'intérêts, ainsi qu'une amélioration invariante aux rotations [29]. Des versions

capables de gérer les voisinages autour des points d'intérêts ont été proposées [30], et tout particulièrement [31] qui utilisent les tenseurs pour résoudre des problèmes plus généraux, mais pour un temps de calcul très élevé pour les mises en correspondance de chemin de longueurs supérieures à 3.

2.3.3 Autres techniques

Certaines méthodes initialement proposées pour la mise en correspondance exacte ont été étendues pour traiter les cas inexacts. Par exemple l'approche par décomposition de [13, 14] a été étendue pour permettre une certaine erreur dans les mises en correspondance [32]. D'autres extensions et améliorations ont aussi été proposées par la suite, comme [33, 34].

Les réseaux de neurones sont aussi utilisés pour effectuer la comparaison de graphes. Par exemple, [35] propose d'apprendre via une carte de Kohonen modifiée les correspondances entre les noeuds d'un graphe et ceux d'un modèle. D'autres méthodes "bio-inspirées" ont été proposées, comme les méthodes génétiques. Par exemple, on peut trouver des techniques pour la mise en correspondance de graphes pondérés [36], pour la recherche d'homomorphisme flou [37], ou encore [38]. Enfin, des techniques sont inspirées du fonctionnement des fourmis[39].

D'autres méthodes s'appuient sur une recherche locale pour résoudre le problème de mise en correspondance [40]. L'idée est d'explorer l'espace des combinaisons en modifiant légèrement à chaque itération la combinaison courante. Cette exploration s'effectue en se déplaçant vers les voisins de la combinaison courante. Diverses approches ont été proposées en ce sens, comme le recuit simulé [41], la montée de gradient [42], ou les methodes dites "taboues" [43, 44].

Enfin, une approche plus récente consiste à produire une fonction noyau entre graphes, en quelques mots une fonction de similarité qui se comporte comme un produit scalaire. Cette approche est plus difficile à classer étant donné que les méthodes vont souvent reposer sur l'une des techniques précédentes plus ou moins modifiées pour répondre aux critères mathématiques. Parmi celles-ci, nous pouvons citer celles basées sur les chemins aléatoires (*random walks*) [45, 46, 47, 48], qui s'intéresse à un ensemble de chemins issus du graphe.

2.4 Discussion

Dans les sections précédentes, nous avons présenté une vue d'ensemble des techniques de comparaison de graphes. Ce domaine est étudié depuis de nombreuses années, et le nombre d'approches proposé est très élevé. Nous présentons ici les caractéristiques particulières de la recherche d'images qui nous ont poussés à nous orienter vers l'approche que nous présentons en détail dans les chapitres suivants.

Nature des graphes. Une première catégorie de méthode de comparaison de graphes s'intéresse davantage à la structure même des graphes. Le meilleur exemple est celui de la recherche de graphes isomorphes. Ce type de méthode est utilisé par exemple en chimie, où les sommets et les arêtes ont des étiquettes symboliques, comme "carbone" ou "hydrogène" pour les sommets, et "au-dessus" ou "en-dessous" pour les arêtes. Dans le contexte de la recherche des images, où les sommets et arêtes sont généralement décrits par des vecteurs, ce type de méthode ne semble pas pertinent. En effet, il est difficile de déterminer clairement si deux éléments sont similaires ou non, ce qui rend caduque les méthodes qui "parient" sur un élagage franc des possibilités.

Nature de la similarité. Beaucoup de méthodes vont reposer sur le fait qu'il existe une notion de similarité pleinement pertinente entre les graphes. Par exemple, déterminer si deux graphes sont isomorphes peut être formulé sans erreur. Par contre, dans notre contexte, la notion de similarité entre images est un problème qui n'est pas soluble. En effet, lorsqu'on compare deux caractéristiques visuelles, la valeur retournée par une fonction de similarité est toute relative, et dépend entièrement de ce qui est recherché. Nous nous intéressons donc avant tout aux valeurs relatives des similarités. Autrement dit le fait qu'un graphe G soit davantage similaire à un graphe G' qu'à un graphe G'' est la première information qui nous intéresse. De plus, il n'est pas nécessaire de produire une méthode qui effectue un calcul précis de la similarité entre graphes. Une imprécision dans le résultat est tout à fait acceptable, étant donné que le calcul exact d'une similarité entre graphes de primitives visuelles est par nature inexact.

Apprentissage. Dans le but de classifier au mieux des images, il faut impérativement passer par l'apprentissage, comme nous l'avons motivé en introduction. Il existe de nombreuses techniques pour ce faire, et nous avons choisi de nous orienter vers les méthodes à noyaux. Ainsi, une grande partie des problèmes liés à l'apprentissage est confiée à des méthodes éprouvées dans ce domaine. En d'autres termes, nous avons choisi un cadre qui permet de séparer les problèmes liés à la mise en correspondance de graphes et ceux liés à la recherche elle-même. Ce cadre a aussi d'autres avantages que nous présentons dans le chapitre suivant. Concernant le choix de méthodes pour la mise en correspondance de graphes, cela implique que seule la valeur de la similarité nous intéresse. Cela signifie que les méthodes qui permettent d'énumérer les éléments mis en correspondance n'ont pas d'intérêt particulier.

Approche. Dans le but de correspondre aux mieux à ces différentes caractéristiques, nous avons choisi de nous intéresser aux techniques qui comptent ou somment les similarités entre les différents éléments des graphes. Plus précisément, nous nous intéressons aux techniques à noyaux basées sur les chemins aléatoires. Ce type de technique nous semble bien adapté étant donné qu'elle se concentre directement sur le calcul de la

similarité. Le caractère aléatoire de cette approche produit un résultat qui a un sens statistique ou probabiliste, comme la valeur moyenne des similarités entre les similarités d'un sous-ensemble des chemins issus des graphes.

Algorithme. Il existe différentes manières d'implanter les techniques par chemins aléatoires, et parmi celles-ci nous avons choisi le branch-and-bound pour les raisons suivantes. Premièrement, cet algorithme permet de calculer rapidement une valeur inexacte de la similarité. Néanmoins, l'erreur commise peut être contrôlée, ce qui permet d'assurer par exemple que deux graphes globalement similaires auront une valeur de similarité élevée. Deuxièmement, il n'est pas nécessaire d'énoncer l'ensemble des chemins pour pouvoir obtenir un résultat, ce qui est intéressant pour réduire les coûts mémoire, contrairement à des méthodes basées sur la matrice des similarités entre chemins. Enfin, cet algorithme se parallélise très bien, aspect qui n'est pas négligeable à l'heure où les cœurs de processeurs sont de plus en plus nombreux.

Chapitre 3

Noyaux sur Graphes

Dans le chapitre précédent, nous avons présenté différentes manières de comparer des graphes. Dans ce chapitre, nous nous intéressons à une approche particulière via le cadre formel des fonctions noyaux. Ce choix nous permet d'utiliser les multiples techniques d'apprentissage disponibles dans la littérature, du "clustering" à la classification en passant par les techniques de "browsing". Ce cadre offre aussi de nombreux avantages comme une meilleure maîtrise des capacités d'apprentissage qu'offre une similarité entre graphes.

Nous commençons par présenter quelques rappels sur ce cadre formel. Puis, nous verrons les techniques proposées dans la littérature avant de présenter nos contributions dans ce domaine.

3.1 Rappels sur les fonctions noyaux

Nous présentons ici quelques éléments fondamentaux du cadre formel des fonctions noyaux. Pour plus de détails et de preuves, nous invitons le lecteur à s'intéresser aux livres de Vapnik[49], de Smola et al.[50] et de Shawe Taylor et al.[51].

3.1.1 Définition

Le premier objectif de ce cadre est de déplacer le problème initialement exprimé dans un espace quelconque $\mathcal{X}$ (dans notre cas, l'espace des graphes) dans un espace hilbertien $\mathcal{H}$. Ce choix repose sur l'idée de ramener tout espace à un espace vectoriel bien connu, et muni de métriques elles aussi bien connues : le produit scalaire et la distance Euclidienne. Pour ce faire, on considère une fonction ϕ, qui à un élément $x \in \mathcal{X}$ fait correspondre une image $\phi(x) \in \mathcal{H}$:

$$\begin{array}{rccc} \phi & : & \mathcal{X} & \rightarrow & \mathcal{H} \\ & & x & \mapsto & \phi(x) \end{array}$$

On dit alors que l'espace $\mathcal{H}$ est l'espace *induit* par la fonction ϕ.

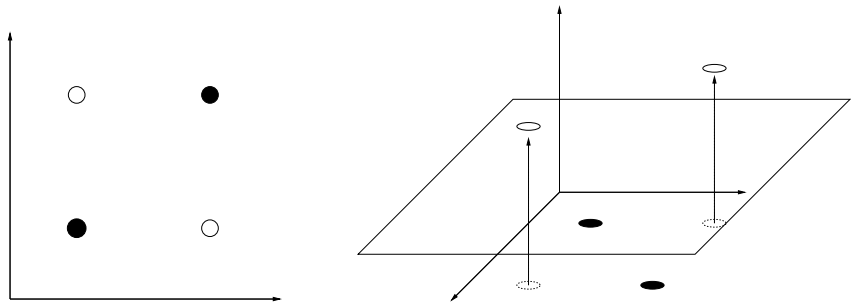


FIGURE 3.1 – Dimension et pouvoir discriminant des classifieurs. Dans le cas à deux dimensions (a), il est impossible de séparer les points noirs et blancs à l'aide d'une droite, alors que dans le cas à trois dimensions (b), cela le devient.

Plus particulièrement, on s'intéresse à une fonction ϕ *injective*, qui donne une image unique à tout élément de $\mathcal{X}$, mais aussi qui injecte les éléments dans un espace $\mathcal{H}$ de plus grande dimension. L'intérêt d'un espace de plus grande dimension est de pouvoir résoudre des problèmes avec des outils linéaires. Par exemple la classification par hyperplan s'en trouve simplifiée, comme l'illustre la figure 3.1. Ainsi, il existe des liens entre une famille de classifieurs et la dimension de l'espace où la classification est effectuée. Davantage d'informations à ce sujet peuvent être trouvées au sein de la théorie de Vapnik[49].

Dans le contexte de la recherche d'informations, les éléments $x \in \mathcal{X}$ généralement très nombreux sont pourvus d'un grand nombre de dimensions. Par exemple, les représentations par histogrammes ont généralement plusieurs centaines de valeurs, voir plusieurs dizaines de milliers pour le cas du texte. Dans le but de pouvoir résoudre les problèmes de manière linéaire, il nous faut souvent travailler dans un espace induit de très grande dimension, voir de dimension infinie. Cela pose naturellement de sérieux problèmes en terme d'implantation, étant donné que les ordinateurs ont une capacité finie de stockage et de traitement. C'est à ce niveau que le cadre des fonctions noyaux va se distinguer des autres méthodes de changement d'espace initial. L'idée est de ne plus travailler directement sur les images de la fonction ϕ, mais uniquement sur les produits scalaires entre les images. Ce produit scalaire s'exprime alors sous la forme d'une *fonction noyau* k définie comme suit :

$$\begin{array}{rccl} k & : & \mathcal{X} \times \mathcal{X} & \to & \mathbb{R} \\ & & x, y & \mapsto & k(x,y) = \langle \phi(x), \phi(y) \rangle \end{array}$$

On dira aussi que l'espace $\mathcal{H}$ est l'espace *induit* par la fonction noyau k.

Ainsi, tout algorithme pouvant s'exprimer en utilisant uniquement un produit scalaire peut être utilisé pour travailler sur des données quelconques et dans un espace induit de n'importe quelle dimension, dès lors que l'on dispose d'une fonction noyau. Prenons l'exemple des algorithmes basés sur la distance euclidienne comme les K-Means.

Sachant que :

$$\begin{aligned} d(\phi(x),\phi(y))^2 &= \langle \phi(x)-\phi(y), \phi(x)-\phi(y)\rangle \\ &= \langle \phi(x),\phi(x)\rangle + \langle \phi(y),\phi(y)\rangle - 2\langle \phi(x),\phi(y)\rangle \end{aligned}$$

Nous pouvons en déduire une expression uniquement basée sur des évaluations d'une fonction noyau :

$$d(\phi(x),\phi(y))^2 = k(x,x)+k(y,y)-2k(x,y)$$

3.1.2 Fonctions noyaux classiques

Voici quelques noyaux classiques qui servent souvent de base dans des noyaux plus complexes :

– **Linéaire** :

$$k(x,y) = \langle x,y\rangle$$

– **Polynomial de degré d** :

$$k(x,y) = \langle x,y\rangle^d$$

– **Gaussien avec une distance Euclidienne** (Gaussien L^2) :

$$k(x,y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

– **Gaussien avec une distance du χ^2** (Gaussien χ^2) :

$$k(x,y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}$$

avec d une distance au sens du χ^2 :

$$d(x,y) = \sum_{r=1}^{p} \frac{(x_r-y_r)^2}{x_r+y_r}$$

– **Triangulaire** :

$$k(x,y) = 1 - \frac{1}{\sigma^2}||x-y||$$

Le choix du noyau de base peut-être guidé par le type de données manipulées. Les fonctions polynomiales représentent des combinaisons de fonctions. Par exemple, dans le cas de la modélisation des occurrences de paires de régions d'image pour une classification de paysage, celle-ci n'est pas obtenue par les exemples d'un seul lieu. Ceci implique le besoin d'une fonction noyau quadratique. Si les occurrences de triplets de régions donnent des informations distinctes, il est intéressant d' utiliser des noyaux cubes. Une fonction radiale de base vous permet d'avoir des caractéristiques qui choisissent cercles (hypersphères). A contrario les limites de décisions des classifications deviennent beaucoup plus complexes.

3.1.3 Techniques de construction

Pour construire de nouvelles fonctions noyaux, une méthodologie courante est de combiner des fonctions noyaux sur vecteurs en utilisant les propriétés suivantes :

- $\forall \lambda > 0$, k est une fonction noyau $\Rightarrow \lambda k$ est une fonction noyau.
- $\forall p \geq 1$, k est une fonction noyau $\Rightarrow k^p$ est une fonction noyau.
- k et k' sont des fonctions noyaux $\Rightarrow k + k'$ est une fonction noyau.
- k et k' sont des fonctions noyaux $\Rightarrow kk'$ est une fonction noyau.

Dans le but d'illustrer ces techniques, prenons le cas des représentations par sacs d'attributs, où un document x_i est représenté par un ensemble non ordonné de vecteurs $B_i = \{b_{ri}\}_r$. S'il existe une fonction noyau k sur les éléments b_{ri}, alors il est possible de construire une fonction noyau K sur sacs B_i en posant :

$$\Phi(B_i) = \sum_r \phi(b_{ri}) \tag{3.1}$$

avec ϕ la fonction injective correspondant à k et Φ celle correspondant à K.

Il s'en suit que :

$$\begin{aligned} K(B_i, B_j) &= \langle \Phi(B_i), \Phi(B_j) \rangle \\ &= \sum_r \sum_s \langle \phi(b_{ri}), \phi(b_{sj}) \rangle \\ &= \langle \sum_r \phi(b_{ri}), \sum_s \phi(b_{sj}) \rangle \\ &= \sum_r \sum_s k(b_{ri}, b_{sj}) \end{aligned} \tag{3.2}$$

La fonction K est une fonction noyau et son expression ne dépend que des valeurs de la fonction k.

Notons que k est généralement appelée fonction noyau *mineure* et K fonction noyau *majeure*.

3.1.4 Semi-définie positivité

Les fonctions noyaux les plus populaires sont les fonctions dites de Mercer, autrement dit qui sont semi-définies positives (*sdp*). De telles fonctions $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ doivent satisfaire la condition suivante, quelque soit les éléments $x_1, \ldots, x_n$ et les réels $c_1, \ldots, c_n$:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i, x_j) c_i c_j \geq 0 \tag{3.3}$$

Cette condition est équivalente à avoir uniquement des valeurs propres positives pour la matrice de Gram associée(notée M_{GRAM}) :

$$M_{GRAM}(K, \mathcal{X}) = \begin{bmatrix} K(x_1, y_1) & \ldots & K(x_1, y_m) \\ \vdots & \ddots & \vdots \\ K(x_m, y_1) & \ldots & K(x_m, y_m) \end{bmatrix} \tag{3.4}$$

avec x et $y \in \mathcal{X}$.

Cependant, lorsque l'on souhaite utiliser certains opérateurs, cette condition ne peut plus être vérifiée pour n'importe quel ensemble d'éléments $x_1, \ldots, x_n$. Prenons par exemple le cas du noyau de Wallraven et al. [52], qui s'appuie sur un calcul de maximum :

$$K(B_i, B_j) = \frac{1}{|B_i|} \sum_r \max_s k(b_{ri}, b_{sj}) + \frac{1}{|B_j|} \sum_s \max_r k(b_{ri}, b_{sj}) \tag{3.5}$$

avec $B_i = \{b_{ri}\}_r$ des ensembles d'éléments.

Outre sa pertinence pour la mise en correspondance, ce noyau est intéressant car l'utilisation d'un calcul de maximum permet d'utiliser des algorithmes rapides, comme nous le verrons dans le prochain chapitre.

Contrairement à ce qui est annoncé par leurs auteurs, cette fonction n'est pas de Mercer. Une contre-preuve est présentée dans [53]. Cependant, il est encore possible de travailler avec ce noyau lorsque sa matrice de Gram associée à la base donnée sur laquelle on travaille est définie positive.

Quand on a la possibilité de travailler sur l'ensemble des données du problème, on peut alors calculer le signe des valeurs propres de la matrice de Gram $\mathbf{K}$. Cette matrice regroupe l'ensemble des valeurs de la fonction noyau sur une base finie X :

$$\forall x_i, x_j \in X^2, \quad \mathbf{K}_{ij} = k(x_i, x_j)$$

Si toutes les valeurs propres de la matrice de Gram sont positives ou nulles, alors il existe une fonction de Mercer k' telle que $\forall x_i, x_j \in X^2,\ k(x_i, x_j) = k'(x_i, x_j)$. Utiliser la fonction k sur la base X revient donc à utiliser k' qui est valide.

Si la matrice de Gram possède des valeurs propres de signes opposés, on peut alors se tourner vers des méthodes pour les noyaux non-*sdp*, par exemple les SVM [54] ou le Discriminant de Fisher [55]. Une des idées de ces noyaux est de travailler avec un autre type d'espace(espace Krein) qui induit un noyau construit à partir de la différence de deux noyaux définis positifs.

Toutefois nous nous sommes restreints aux noyaux de Mercer et à quelques noyaux non de Mercer dont nous n'avons pas cherché à trouver leur pendant avec les noyaux précédemment cités. Pour toutes nos expériences avec des noyaux non Mercer(le maximum ou somme de maximums) que nous présentons dans le chapitre 4, les matrices de Gram qui ont été calculées se sont révélées définies positives.

3.2 Etat de l'art

3.2.1 Méthodes de Noyaux sur Graphes

Le noyau de convolution de Haussler[56] est un modèle très généraliste. Il a permis la proposition de nouveaux noyaux sur des éléments structurés à partir des sous-éléments les constituant. Dans le cadre des graphes les sous-éléments pouvant être utilisés sont les sommets, les arêtes, les chemins et les sous-graphes. Gärtner[57] a donc proposé des noyaux sur graphes se basant sur ce principe. D'autres noyaux ont été proposés et peuvent se regrouper par famille en fonction du type de structure qu'ils considèrent.

Parmi ces familles de noyaux, l'une d'entre-elles se base sur des similarités entre chemins. D'après Gartner et al [45], si l'on prend tous les parcours aléatoires d'un graphe, on obtient le graphe lui-même. Cet ensemble de parcours aléatoires revient à considérer un ensemble de chemins puisque le parcours aléatoire définit un chemin. Cette famille s'appuie sur ce principe pour obtenir la similarité entre deux graphes à partir de la somme de toutes les similarités possibles entre leurs ensembles de chemins. Ces ensembles pouvant être infinis, en pratique on ne considère pas l'intégralité des chemins. Diverses approches sont possibles : par le tirage aléatoire des parcours [58, 59, 60] ou par le produit direct entre graphes [61, 48].

Une autre famille considère des sous-graphes élémentaires (*graphlets*) pour calculer les similarités entre deux graphes [62]. Ces méthodes sont motivées par les limites des parcours aléatoires (ou chemins) de ne pas discerner certains graphes [63] entre eux. Ces problèmes de discernement n'arrivent que lorsque les similarités entre sommets (et arêtes) ont des valeurs binaires. Dans certains travaux, les graphlets sont spécifiés comme des arbres [63, 64].

De nombreux noyaux de la littérature ont été construits pour les applications bio-informatiques ou chimiques. Ces applications manipulent des données qui possèdent une faible information sur les sommets et arêtes généralement représentée par un petit vecteur (moins de quatre dimensions) ou une étiquette. De plus, ces méthodes ont été proposées pour des graphes de petite taille, ou alors elle ne fonctionnent que sur des graphes etiquetés [62]. Une adaptation des méthodes est nécessaire dans le contexte de la recherche multimédia. Plusieurs techniques ont été proposées pour l'effectuer : nous présenterons ici, deux approches différentes pour injecter les éléments structurels dans un espace Hilbertien.

La première est l'injection explicite qui travaille à partir des attributs du graphe (nombre de sommets, spectre, chemins) sélectionnés. Afin de choisir ces attributs, des prototypes sont construits en utilisant des techniques comme K-Means, l'ACP [65] , "Multiple instance learning"[66] ou encore les forêts aléatoires[67]. Une fois ceux-ci définis, l'injection explicite va, par exemple, calculer la distance dans le cadre de chaque prototype, puis l'utiliser dans un noyau classique sur vecteur ou histogramme[68]. Dans [69], le résultat d'une fonction d'appartenance binaire ou non binaire au prototype est stocké dans une valeur du vecteur correspondant à un graphe.

L'approche explicite limite la taille des vecteurs dans l'espace induit, réduisant ainsi leur coût de stockage en mémoire. Elle requiert des paramètres globaux (comme le

nombre de prototypes) réglés pour chaque base de données ou chaque requête. Une solution possible est d'effectuer le calcul en ligne des paramètres et des prototypes au cours de la requête[70, 71].

La deuxième approche est l'injection implicite. Celle-ci permet la manipulation de représentations de grande dimension sans jamais avoir à les calculer. Les méthodes [72, 53, 73] ont été appliquées à des sacs (pouvant être vus comme des graphes sans arêtes). Certaines méthodes[59, 60] ont été appliquées à de la recherche multimédia. D'autres techniques ont été utilisées comme les spectrales sur des paires de sommets [28] ou les tenseurs pour des ordres de mise en correspondance supérieurs[31].

Dans cette thèse, nous nous sommes basés sur la dernière approche et aussi sur les chemins de graphes issus de parcours aléatoires. Dans le cadre de la comparaison de graphes avec une richesse d'informations sur les nœuds et arêtes, il semble que ce type d'approche soit plus adapté. Afin d'introduire au mieux nos contributions dans la section suivante, nous présentons en détails les méthodes de Kashima [58] et Suard [59] dans les sous-sections suivantes.

3.2.2 Méthode de Kashima

Cette méthode[46] fait partie des méthodes qui s'appuient sur des parcours aléatoires pour modéliser puis comparer les graphes. Elle fait suite aux travaux de Tsuda[74], Vishwanathan[75] et Gärtner[45].

Soit $G = (V, E)$ et $G' = (V', E')$ deux graphes. On considère l'ensemble $H(G)$ (resp. $H'(G)$) des chemins h de G (resp. h' de G') composés d'une séquence de n sommets $v_0, \ldots, v_n$ (resp. $v'_0, \ldots, v'_n$), chaque sommet v_i étant relié au sommet suivant v_{i+1} par l'arête e_i (resp. v'_i, v'_{i+1} et e'_i).

Le noyau $K_{kashima}(G, G')$ est la somme pondérée des comparaisons entre tous les chemins h de G et h' de G' de même longueur :

$$K_{Kashima}(G, G') = \sum_{n=1}^{\infty} \sum_{\substack{h \in H(G) \\ |h|=n}} \sum_{\substack{h' \in H(G') \\ |h'|=n}} K_C(h, h') p(h|G) p(h'|G') \tag{3.6}$$

La fonction noyau $K_C(h, h')$ compare deux chemins de même longueur n, et s'appuie sur un noyau $K_V(v, v')$ qui compare les descripteurs de deux sommets $v \in V$ et $v' \in V'$, ainsi que sur un noyau $K_E(e, e')$ qui compare les descripteurs de deux arêtes $e \in E$ et $e' \in E'$:

$$K_C(h, h') = K_V(v_1, v'_1) \prod_{i=2}^{n} K_V(v_i, v'_i) K_E(e_i, e'_i)$$

Kashima utilise des noyaux sur sommets et arêtes qui renvoient des valeurs entre 0 et 1. Cette condition, associée à celle d'un noyau sur chemin composé de produit et par conséquent à valeur décroissant en fonction de la longueur de chemin, lui permet d'obtenir une convergence sur une suite récursive réexprimant le noyau sur graphe K_G par rapport à la longueur des chemins. Il en déduit ainsi une résolution par système linéaire du noyau grâce à cette suite et à sa convergence. Lorsque les descripteurs sont

des vecteurs, l'utilisation de noyau Gaussien L^2 est préconisée. Cependant, il reste tout à fait possible d'utiliser d'autres noyaux dès lors qu'ils renvoient des valeurs entre 0 et 1.

Les probabilités $p(h|G)$ associées à un chemin h sont calculées de la manière suivante :

$$p(h|G) = p_s(v_1) \prod_{i=2}^{n} p_t(v_i|v_{i+1}) p_q(v_n)$$

avec $p_s(v)$ la probabilité (uniforme) de trouver le sommet v dans G, $p_t(v_i|v_{i+1})$ la probabilité de transition de v_i à v_{i+1} et $p_q(v)$ la probabilité que le chemin s'arrête au sommet v sont contraintes de la manière suivante :

$$\forall v_i, \sum_{j=1}^{|V|} p_t(v_j|v_i) + p_q(v_i) = 1$$

En théorie, le noyau proposé doit comparer tous les chemins jusqu'à une longueur infinie. Cependant, le caractère récursif de l'expression permet d'exprimer le problème sous la forme d'un système linéaire fini. De plus, les valeurs des probabilités et celles des noyaux sur sommets et arêtes étant inférieures à 1, la convergence du calcul est assurée.

Cependant, comme le montre Suard[59], cette méthode a une forte tendance à considérer de nombreuses fois les mêmes arêtes. Cet aspect est peut-être peu gênant pour des graphes de molécules, mais dans le contexte des graphes issus d'images, un très grand nombre de petites valeurs de similarité sont ainsi sommées, et finissent par diluer les similarités les plus fortes. Cela pose aussi des problèmes en terme de complexité calculatoire.

3.2.3 Méthode de Suard

Cette méthode proposée dans [59] est une version approximée de la méthode de Kashima. Son principal objectif est de réduire la complexité calculatoire. Le noyau proposé est le suivant :

$$K_{Suard}(G, G') = \frac{1}{2} \left(\sum_{h \in H(G)} \max_{\substack{h' \in H(G') \\ |h'|=|h|}} K_C(h, h') + \sum_{h' \in H(G')} \max_{\substack{h \in H(G) \\ |h|=|h'|}} K_C(h, h') \right) \tag{3.7}$$

Le noyau sur chemin $K_C(h, h')$ est similaire à celui proposé par Kashima.

Cette fonction est composée de deux parties pour assurer la symétrie. La partie gauche (et de même la partie droite) somme les similarités entre un chemin h de G et son meilleur appariement de même longueur dans G'. Cela permet d'éviter de sommer trop de petites valeurs de similarités.

La méthode propose aussi de considérer un ensemble de chemins moins vaste, à savoir l'ensemble des plus courts chemins entre deux sommets du graphe. Ainsi, il y a au plus $|V|^2$ (resp. $|V'|$) chemins considérés dans G (resp. G'), et donc au plus $|V|^2 \times |V'|^2$ comparaisons de chemins.

Au final, pour des graphes d'ordre n, cette méthode permet de passer d'une complexité $O(n^6)$ à $O(n^4)$. Les expériences menées montrent des performances légèrement inférieures à celles de Kashima, mais pour un temps de calcul bien meilleur.

3.3 Contributions

3.3.1 Noyau sur sacs de chemins

Notre objectif dans cette section est de proposer des fonctions noyaux adaptées aux particularités des graphes issus des images. En effet, dans ce contexte, les valeurs des similarités entre les sommets et arêtes évoluent de manière assez uniforme dans un intervalle. Prenons l'exemple des noyaux mineurs basés sur des Gaussiens χ^2, dont les valeurs possibles sont entre 0 et 1. Dans la pratique, il est très rare de trouver deux sommets dont la similarité est égale à 1, et de même pour les valeurs proches de 0. Avec un paramétrage classique, deux sommets visuellement proches vont avoir une similarité de l'ordre de 0.9, et deux sommets visuellement différents vont avoir une similarité de l'ordre de 0.1. Puis, pour toutes les autres possibilités, la notion de similitude est toute relative. Par conséquent, suivre une stratégie où l'on accumule un très grand nombre de similarités n'est pas efficace dans ce domaine. Les problèmes dont souffrent certains noyaux comme celui de Kashima s'en trouvent donc amplifiés.

Dans le but de répondre à cette problématique, nous proposons de nous intéresser à une famille de fonctions noyaux qui va effectuer une sorte de filtrage sur les similarités à accumuler. Plus précisément, nous proposons de ne considérer que les couples de chemins issus des deux graphes dont la similarité entre les premiers sommets est élevée. Cela peut s'exprimer de la manière suivante :

$$K(G,G') = \sum_{v\in V}\sum_{v'\in V'} K_A(v,v')\hat{K}\left(H_v(G), H_{v'}(G')\right) \tag{3.8}$$

avec :

- $G=(V,E)$ et $G'=(V',E')$ les deux graphes à comparer ;
- $K_A(v,v')$ une fonction qui va déterminer si le couple de sommets (v,v') doit être pris en compte ;
- $\hat{K}(H,H')$ une fonction qui compare deux ensembles de chemins H et H' ;
- $H_v(G)$ une fonction qui renvoie un ensemble de chemins issus de G dont le premier sommet est v.

L'idée est alors de jouer sur la fonction $K_A(v,v')$ pour ne prendre en compte que les similarités pertinentes.

Notons que cette formulation intègre les formules précédentes. Par exemple, si on choisit $K_A = 1$, mises à part les probabilités, on peut retrouver une formule similaire à l'équation (3.6) avec :

$$\hat{K}_{somme}(H,H') = \sum_{n=1}^{\infty}\sum_{\substack{h\in H\\|h|=n}}\sum_{\substack{h'\in H'\\|h'|=n}} K_C(h,h') \tag{3.9}$$

Dans le cas où l'on peut déterminer via un seuil θ si deux sommets sont similaires ou non (comme dans le cas des points d'intérêts), on peut utiliser la fonction K_A suivante :

$$K_{A_{seuil}}(v,v') = \begin{cases} 1 & \text{si } K_V(v,v') < \theta \\ 0 & \text{sinon} \end{cases} \tag{3.10}$$

Si on a la possibilité de construire un dictionnaire de sommets, on peut alors utiliser la fonction suivante :

$$K_{A_{dict}}(v, v') = \sum_r a_r(v) a_r(v') \tag{3.11}$$

avec $a_r(v)$ une fonction qui renvoie 1 si v est dans le cluster r, 0 sinon.

Enfin, si on ne peut ni fixer un seuil ni calculer un dictionnaire, on peut alors se tourner vers une fonction qui ne sélectionne que les couples les plus proches :

$$K_{A_{ppv}}(v, v') = \frac{1}{|V|} p_{v'}(v) + \frac{1}{|V'|} p_v(v') \tag{3.12}$$

avec

$$p_v(v') = \begin{cases} 1 & \text{si } v' \in \text{ppv}_k(v) \\ 0 & \text{sinon} \end{cases} \tag{3.13}$$

et $\text{ppv}_k(v)$ une fonction qui renvoie les k plus proches voisins de v dans G'.

Toujours dans le but de réduire le nombre de similarités accumulées, il est aussi possible de jouer sur la fonction $\hat{K}(H, H')$ qui compare deux ensembles de chemins. Pour ce faire, nous proposons de calculer la valeur de la plus grande similarité entre les chemins des deux ensembles :

$$\hat{K}(H, H') = \max_{n \in [1,N]} \max_{\substack{h \in H \\ |h|=n}} \max_{\substack{h' \in H' \\ |h'|=n}} K_C(h, h') \tag{3.14}$$

Avec une telle fonction, il y autant de valeurs de similarité accumulées dans l'équation (3.8) que de couples de sommets sélectionnées par la fonction $K_A(v, v')$. Un autre avantage de cette formule est la recherche d'une valeur maximale. En effet, avec des noyaux sur chemins $K_C(h, h')$ récursifs, il est possible d'utiliser des algorithmes de calcul rapide. Ces algorithmes et leur mise en œuvre sont présentés dans le chapitre suivant.

La formulation finale que nous expérimentons dans les chapitres suivants peut être écrite de la manière suivante :

$$\begin{aligned} K_{new}(G, G') &= \frac{1}{|V|} \sum_{\substack{v \in G \\ v' \in \text{ppv}_k(v)}} \max_{h \in H_v(G)} \max_{\substack{h' \in H_{v'}(G') \\ |h'|=|h|}} K_C(h, h') \\ &+ \frac{1}{|V'|} \sum_{\substack{v' \in G \\ v \in \text{ppv}_k(v')}} \max_{h' \in H_{v'}(G)} \max_{\substack{h \in H_v(G) \\ |h|=|h'|}} K_C(h', h) \end{aligned} \tag{3.15}$$

Cette fonction n'est pas une fonction de Mercer, cependant elle peut être utilisée comme une fonction noyau (cf. section 3.1.4).

3.3.2 Noyaux sur chemins

Pour compléter un noyau sur un ensemble de chemins, il est nécessaire de définir un noyau sur ces chemins : $K_C(h, h')$. Dans cette section, nous nous intéresserons à ces noyaux et nous proposerons diverses solutions possibles. Ces noyaux sont basés sur des

noyaux mineurs des sommets K_V et des arêtes K_E. Nous supposons que ce sont des noyaux gaussiens qui retournent des valeurs comprises entre 0 et 1.

Un noyau sur chemins très utilisé dans la littérature effectue le produit entre toutes les similarités des sommets et arêtes composant les deux chemins :

$$K_{C_{mul}}(h, h') = K_V(v_0, v'_0) \times \prod_{i=1}^{|h|} K_E(e_i, e'_i) K_V(v_i, v'_i) \tag{3.16}$$

Lorsque les noyaux mineurs retournent des valeurs entre 0 et 1 la fonction décroît toujours avec l'augmentation de la taille du chemin, et par conséquent les similarités des longues chaînes ont de très faibles valeurs. Lorsqu'il est utilisé avec un noyau sur sacs de chemins qui somme l'ensemble des similarités (comme dans l'équation (3.6)), cette fonction ne pose pas de problème. Cependant, lorsqu'elle est utilisée avec un noyau sur sacs de chemins qui recherche les meilleures similarités (comme les équations (3.7) ou (3.15)), seules sont accumulées les similarités des chemins les plus courts.

Un autre type de noyau sur chemin a été proposé dans la littérature, comme celui qui somme les similarités entre sommets/arêtes [76] :

$$K_{C_{som}}(h, h') = K_V(v_0, v'_0) + \sum_{i=1}^{|h|} K_E(e_i, e'_i) K_V(v_i, v'_i) \tag{3.17}$$

Cette fonction augmente de façon systématique avec la taille des chemins si les noyaux mineurs sont positifs. Les similarités des chemins courts seront inférieures à celles des mêmes chemins prolongés. Cependant, cela n'est pas le plus gênant étant donné qu'il est de plus en plus difficile de trouver des chemins de grande longueur dont tous les sommets sont très similaires. Le principal reproche que l'on pourrait faire est que, contrairement au noyau précédent, quelques sommets/arêtes très similaires suffisent pour considérer que les deux chemins sont similaires.

Dans le but de pallier aux problèmes de ces fonctions noyaux, mais aussi de mieux répondre aux particularités des graphes issus des images, nous proposons les deux fonctions suivantes.

Nous proposons tout d'abord le noyau suivant, dont le premier but est d'améliorer celui de l'équation (3.17) :

$$K_{C_{new1}}(h, h') = K_V(K_0, v'_0) \times \prod_{i=1}^{|h|} (1 + K_E(e_i, e'_i) \times K_V(v_i, v'_i)) \tag{3.18}$$

A l'aide d'une telle formule, nous obtenons une augmentation de la similarité avec la longueur des chemins comme dans (3.17). Cependant, en utilisant un produit, nous pénalisons fortement tout couple de chemins qui a au moins un couple de sommets ou d'arêtes non similaires.

Étant donné que les descripteurs sur arêtes sont généralement de plus faibles dimensions (les similarités sont par conséquent plus franches), il peut être intéressant de les

traiter différemment :

$$K_{C_{new2}}(h, h') = K_V(v_0, v'_0) \times \prod_{i=1}^{|h|} K_E(e_i, e'_i) \times (1 + K_V(v_i, v'_i)) \tag{3.19}$$

Contrairement aux autres, cette fonction n'est pas monotone avec la taille des chemins. Dans le cas où les chemins h et h' représentent des régions disposées dans l'image de la même manière, cette fonction va augmenter avec la taille des chemins. Dans l'autre cas, où au moins une des régions est disposée différemment, la similarité va chuter de manière significative. Ainsi, seuls les ensembles de régions disposées de manière similaire seront similaires. En plus de donner un poids important aux relations spatiales, cette fonction va aussi permettre des calculs plus rapides étant donné que davantage de comparaisons de chemins seront ignorées par les algorithmes rapides.

Chapitre 4

Algorithme de mise en correspondance de chemins

Dans le chapitre précédent, nous avons présenté des fonctions noyaux comme fonctions de similarité entre graphes. Bien que ces fonctions possèdent des propriétés intéressantes, leur évaluation est coûteuse en temps de calcul. Nous allons présenter dans ce chapitre un cadre permettant de limiter le temps de calcul de ces fonctions.

Le temps de calcul de ces fonctions est lié au temps nécessaire pour apparier deux chemins de deux graphes et calculer leur similarité. Par exemple, la fonction noyau de Kashima utilise la somme des similarités de tous les appariements possibles. Pour résoudre ce problème, on recense différentes méthodes dans la littérature pour :

- réduire le nombre de chemins (et donc le nombre d'appariements) ,
- réduire le nombre de calculs par une approximation de la fonction de similarité,
- réduire la redondance dans les chemins par une construction récursive,
- réduire la redondance de calcul via le stockage de résultats partiels communs à divers calculs.

La première méthode réduit le nombre de chemins appariés. Par exemple, Suard a fortement réduit ce nombre en n'utilisant que les chemins représentant des plus courts chemins entre 2 sommets donnés. Outre une réduction, ce choix permet l'utilisation d'algorithmes performants pour la génération des chemins. De plus, la formule du noyau considéré reste inchangée en utilisant une méthode de réduction du nombre de chemins. Néanmoins ce genre de réduction peut poser des problèmes quant à la conservation des propriétés mathématiques de la fonction noyau sur graphe. Enfin, il impose un choix qui peut se révéler important selon la morphologie structurelle du graphe.

La méthode d'approximation est le plus souvent utilisée lorsque le nombre de chemins est infini ou très grand. Il s'agit d'approximer les valeurs des fonctions en effectuant soit des calculs de fonctions proches mais plus simples, soit en supprimant certains calculs ayant une faible contribution sur la valeur finale, soit en arrêtant, avant terminaison, le calcul en utilisant des propriétés de convergence. Kashima l'utilise pour pallier une

possibilité de temps de calcul infini (si le nombre de chemins est infini, le nombre d'appariements le sera. Par conséquent, le calcul de la somme sur toutes les similarités des appariements ne se finit jamais) . Néanmoins, celle-ci pose des problèmes de non conservation des propriétés mathématiques des fonctions noyaux utilisées dans la similarité.

La limitation de la redondance dans les chemins s'effectue le plus souvent à l'aide de création récursive des chemins. Celle-ci nécessite de considérer des ensembles de chemins ayant des propriétés de récursivité.

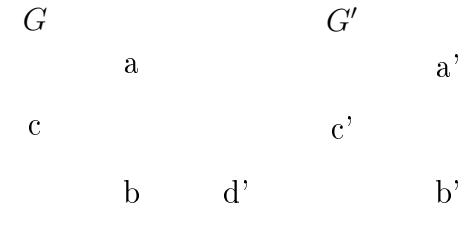
Le stockage d'éléments s'arrête le plus souvent au stockage des similarités sur les arêtes et les sommets. La similarité des chemins peut s'écrire le plus souvent comme une combinaison de la similarité de ses sous-éléments arêtes et sommets. La similarité des graphes peut elle-même combiner la similarité de nombreux chemins. Ainsi ces similarités d'arêtes et de sommets interviennent de nombreuses fois dans le calcul des similarités de chemins et de graphes.

Notre méthode exploite la redondance et la récursivité de l'expression des appariements de chemins dans une représentation sous forme d'arbre. Dans cette représentation, un chemin entre la racine et la feuille de l'arbre est une solution d'appariement de chemins. Cette représentation permet de se placer dans le cadre d'exploration d'un arbre de recherche. Par conséquent des algorithmes de type $A*$, par exemple le "branch and bound" et d'autres algorithmes d'exploration d'arbre de recherche peuvent être employés pour résoudre l'appariement de deux graphes par similarité d'ensembles de chemins.

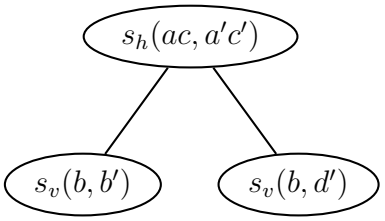
Nous présentons la mise en représentation sous forme d'arbre d'appariements pour une catégorie de fonctions de mise en correspondance de graphe. Puis nous montrons l'application de cette représentation d'arbre avec des ensembles de chemins pour construire un arbre de recherche parcourant l'ensemble des possibilités. Enfin nous présentons l'utilisation algorithmique de ces représentations pour calculer les similarités noyaux sur graphes.

4.1 Représentation arborescente du calcul de similarité de chemins

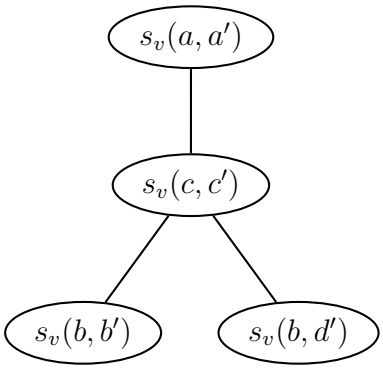
Les chemins ainsi que leur fonctions de similarité basées sur la similarité entre sommets et arêtes sont calculables ou constructibles récursivement. Les éléments possédant cette propriété récursive peuvent facilement se représenter sous forme d'arbre en regroupant les parties communes. La littérature a détaillé l'utilisation d'arbres de recherche sur des appariements de chemins entre deux graphes. Ce qui nous intéresse est d'obtenir un arbre de recherche dans le cadre d'ensemble de chemins particuliers. Et de plus, il faut que la fonction de similarité entre les chemins soit calculable lors de l'exploration de l'arbre. Pour cela il faut vérifier que le calcul peut lui aussi se représenter sous forme d'arbre. Nous allons dans un premier temps montrer au travers d'exemples comment on peut effectuer cette représentation arborescente puis dans un deuxième temps, formaliser l'écriture de ces similarités au travers de l'arbre. Enfin nous mettrons en place une structure arborescente qui permettra la mise en place dans la prochaine



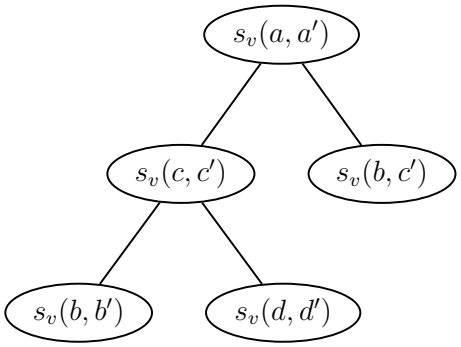
(a) Exemple de deux graphes G et G'



(b) Arbre T_1 où la similarité commune des appariements de chemins (issues de G et G') est compactifiée sur le nœud racine



(c) Arbre T_2 avec une séparation de la racine de T_1 en deux nœuds



(d) Ajout d'un nœud portant la similarité de $s_v(b, c')$

FIGURE 4.1 – On calcule des valeurs de similarité pour des appariements de chemin entre deux graphes, G et G', à l'aide d'arbre.

section, d'un arbre de recherche valide pour des ensembles de chemins.

4.1.1 Un exemple simple

Nous proposons d'illustrer la représentation du calcul de la similarité de graphes sous forme d'arbre au travers d'un exemple. Dans celui-ci, nous voulons comparer deux graphes G et G' (figure 4.1(a)) à l'aide des chemins de longueur deux (soit deux arêtes voir tableau 4.1). Pour cela nous considérons une fonction de similarité de graphes, S_G, qui renvoie la valeur du meilleur appariement parmi l'ensemble des appariements de chemins de longueur deux, issus des graphes comparés. Autrement dit une fonction qui sélectionne l'appariement avec la plus haute valeur de similarité entre chemins, noté s_h, dans l'ensemble des appariements de chemins. Ce qui revient à la formule suivante :

$$S_G(G, G') = \max_{h \in H^2(G)} \max_{h \in H^2(G')} s_h(h, h')$$

avec $H^2(G)$ l'ensemble des chemins de longueur 2 de G.

	liste des chemins
$H^2(G)$	abc,acb,bac,bca,cab,cba
$H^2(G')$	$a'c'b'$,$a'c'd'$,$b'c'a'$,$b'c'd'$,$d'c'b'$,$d'c'a'$

TABLE 4.1 – Tableau récapitulatif des chemins de longueur 2 issus des deux graphes G et G' de la figure 4.1(a)

Pour rappel un chemin est une séquence ordonnée de sommets et d'arêtes, on le note h. On peut écrire la séquence de sommets et d'arêtes d'un chemin h de la manière suivante :

$$h = v_0 e_1 v_1 ... e_n v_n$$

ou bien se contenter d'énumérer les sommets $h = v_0 v_1 ... v_n$.

Pour définir la ressemblance entre deux chemins, on peut simplement sommer les similarités s_v entre chaque paire de sommets des deux chemins appariés :

$$s_h(h = v_0 \dots v_n, h' = v'_0 \dots v'_n) = \sum_{i=0}^{n} s_v(v_i, v'_i)$$

En reprenant les graphes d'exemple (figure 4.1(a)), pour l'appariement du chemin abc de G et du chemin $a'c'b'$ de G' on a :

$$s_h(acb, a'c'b') = s_v(a, a') + s_v(c, c') + s_v(b, b')$$

Le caractère récursif de la fonction de similarité nous permet de réécrire cette similarité entre chemins de longueur deux en fonction de celle de l'appariement de chemins de longueur une :

$$s_h(acb, a'c'b') = s_h(ac, a'c') + s_v(b, b')$$

Nous allons, maintenant, comparer la similarité entre ces deux chemins avec celle de acb de G et $a'c'd'$ de G'.

$$s_h(acb, a'c'b') = s_v(a, a') + s_v(c, c') + s_v(b, b') = s_h(ac, a'c') + s_v(b, b')$$

et

$$s_h(acb, a'c'd') = s_v(a, a') + s_v(c, c') + s_v(b, d') = s_h(ac, a'c') + s_v(b, d')$$

On remarque :

- qu'ils ont des valeurs communes ($s_h(ac, a'c')$),
- qu'ils peuvent se réécrire à partir de similarité de chemins de longueur inférieure (chemin ac de G apparié avec le chemin $a'c'$ de G').

La représentation de la figure 4.1(c) se rapproche plus d'un arbre représentant les appariements. La figure 4.1(b) montre une représentation de ces similarités sous forme d'arbre : la similarité commune est encore plus visible (premier nœud). La valeur de similarité portée par le premier nœud de l'arbre de la figure 4.1(b) est égale à la somme des similarités portées par les deux premiers nœuds de l'arbre de la figure 4.1(c).

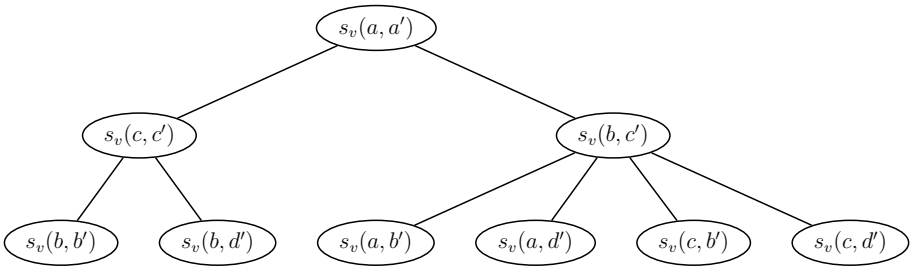


FIGURE 4.2 –

En sommant chaque similarité le long du parcours du premier nœud de l'arbre à une feuille de l'arbre on obtient la similarité de l'appariement de chemin représenté par ce parcours. Dans le cas de la figure 4.1(c) les similarités des appariements acb avec $a'c'd'$ et acb avec $a'c'b'$.

Si l'on considère de nouveaux appariements de chemins commençant par ab et $a'c'$ qui comparent non plus c avec c' mais b avec c', on développe dans un premier temps le nœud $s_v(a, a')$ pour lui rajouter le fils $s_v(b, c')$ (figure 4.1(d)). Pour obtenir toutes les feuilles de profondeur 2 qui portent les solutions d'appariements de longueur 2, on développe ce nœud. On obtient de nouvelles feuilles à l'arbre (figure 4.2) qui sont autant de nouvelles solutions d'appariements.

La fonction de similarité peut être vue comme la fonction d'évaluation(idéalement calculable pendant l'exploration de l'arbre) pour des algorithmes de type $A*$. Pour un arbre de recherche représentant les diverses solutions d'appariements entre les chemins de deux graphes. Un arbre de recherche est un arbre représentant un espace de solutions finies.

Pour utiliser ce genre de représentation et ce type d'algorithme, il est nécessaire de s'assurer des contraintes qui se posent sur les fonctions de similarités de chemins et de la complète représentation de l'espace par l'arbre de recherche considéré. Dans la prochaine section nous traitons le premier point.

4.1.2 Similarités sur chemins pour arbre de recherche

Le processus introduit dans la section précédente peut se généraliser à d'autres fonctions de similarités de chemins. Dans cette section, nous allons présenter les catégories de fonctions qui sont utilisables dans le cadre d'arbre de recherche. Lors de la création des arbres pour l'exemple (section 4.1.1), nous avons utilisé cette fonction de similarité sur chemins :

$$s_h(h = v_0 \dots v_n, h' = v'_0 \dots v'_n) = \sum_{i=0}^{n} s_v(v_i, v'_i)$$

Cette fonction peut être calculée via des méthodes récursives :

$$\begin{aligned} s_h(h = v_0 \dots v_n, h' = v'_0 \dots v'_n) = & \sum_{i=0}^{n} s_v(v_i, v'_i) \\ = & \sum_{i=0}^{n-1} s_v(v_i, v'_i) + s_v(v_n, v'_n) \\ = & s_h(h = v_0 \dots v_{n-1}, h' = v'_0 \dots v'_{n-1}) + s_v(v_n, v'_n) \end{aligned}$$

La valeur de similarité de notre appariement (h, h') peut être calculée à partir de la valeur d'un sous appariement de celui-ci $(v_0 \dots v_{n-1}, v'_0 \dots v'_{n-1})$. Cette propriété est nécessaire pour un calcul de la similarité pendant l'exploration de l'arbre de recherche d'appariement.

La fonction précédente peut se réécrire sous une forme récursive :

$$s(h_n, h'_n) = \begin{cases} s_v(v_0, v'_0) \text{ si } h = v_0 \text{ et } h' = v'_0 \\ s_h(h_{n-1}, h'_{n-1}) + s_v(v_n, v'_n) \text{ si } n > 1 \end{cases}$$

avec $h_n = h_{n-1} e_n v_n$ et $h'_n = h'_{n-1} e'_n v'_n$.

La fonction prise comme exemple possède la particularité supplémentaire d'être une fonction noyau si la fonction s_v l'est. De nombreuses fonctions noyaux pour les similarités de chemins sont comme la fonction précédente des fonctions récursives. Plus particulièrement, les similarités de chemins utilisées dans les noyaux sur graphes sont souvent récursives. Car elles sont généralement des combinaisons de fonctions noyaux sur les sous-éléments qui composent un chemin : les sommets et les arêtes.

Par la suite, nous ne considérons que les fonctions récursives (notées s_h) que l'on pourra réécrire sous cette forme à l'aide de trois opérateurs op_v,op_e et op_{noeud} :

$$s_h(h_n, h'_n) = \begin{cases} op_v(v_0, v'_0) \text{ si } h_n = v_0 \text{ et } h'_n = v'_0 \\ op_e(v_0 e_1 v_1, v'_0 e'_1, v'_1) \text{ si } h_n = v_0 e_1 v_1 \text{ et } h'_n = v'_0 e'_1 v'_1 \\ op_{noeud}(e_n, v_n, e'_n, v'_n, s_h(h_{n-1}, h'_{n-1})) \text{ sinon} \end{cases}$$

Les opérateurs op_v et op_e similarité de sommets et d'arêtes :

$$op_v : V \times V \to \Re$$

$$op_e : E \times E \to \Re$$

On utilise ces opérateurs pour les premiers nœuds de l'arbre. Dans l'arbre T_1 de notre section exemple (4.1(b)), on utilise $op_e(ac, a'c') = s_v(a, a') + s_v(c, c')$ pour la racine. Dans l'arbre T_2(4.1(c)), on utilise $op_v(a, a') = s_v(a, a')$ pour la racine. Le choix entre ces deux opérateurs dépend de la fonction de similarité. Notre fonction exemple utilise uniquement les sommets ; par conséquent, il vaut mieux travailler avec l'opérateur op_v.

L'opérateur op_{noeud} ($op_{noeud} : E \times V \times E \times V \times \Re \to \Re$) correspond aux opérations à effectuer à partir de la valeur de similarité du sous-appariement (h_{n-1}, h'_{n-1}) et des nouveaux éléments (sommets v_n et v'_n et arêtes e_n et e'_n) pour obtenir celle de l'appariement (h_n, h_n). C'est ce dernier opérateur qui est utilisé dans nos arbres de recherche pour mettre la valeur de similarité à jour à chaque création de nœud.

Pour la suite, nous supposons que toutes les fonctions de similarité sur les chemins sont récursives. Ainsi dans le cadre d'arbre de recherche sur des espaces d'appariement

de chemins, on pourra calculer ou estimer le calcul de la similarité pendant l'exploration de l'arbre. Ainsi ces fonctions pourront être utilisées dans les algorithmes d'exploration d'arbre de recherche comme fonction de référence.

4.1.3 Arbre d'appariements sur chemins de longueur fixe

Nous allons dans cette section définir un arbre qui représente l'espace des appariements de chemins de longueur fixe et de manière exhaustive : un arbre d'appariement. Cet arbre nous permettra de représenter complètement et uniquement cet espace des appariements entre $H(G)$ et $H(G')$ (H une fonction générant un ensemble de chemins à partir d'un graphe). De plus les fonctions de similarité des chemins de cet arbre sont calculables pendant d'exploration de l'arbre.

Nous allons dans un premier temps définir l'arbre d'appariement de façon à obtenir un arbre de recherche valide pour l'espace constitué de tous les appariements possibles entre les chemins de longueur fixe entre deux ensembles. La définition d'arbre d'appariement, nous assure que l'arbre contient toutes les solutions d'appariement entre les chemins de deux ensembles A et B. Puis on s'intéresse à ne considérer que l'espace de solution des appariements de chemins de longueur fixe.

Un ***arbre d'appariement*** $T(A, B)$ *pour deux ensembles de chemins A et B est un arbre de recherche dont tous les parcours entre la racine et une feuille représentent des appariements entre deux chemins de même longueur issus de A et B. De plus, tout appariement de deux chemins de même longueur issus de A et B est représenté par un parcours entre la racine et un nœud de l'arbre.*

On note $App(A, B)$ l'ensemble des appariements entre deux chemins de A et de B.

Le dernier arbre de recherche présenté (Fig. 4.2) lors de l'exemple de la section 4.1.1 est un arbre d'appariement entre les ensembles de chemins $A = \{ac, acb, aba, abc\}$ et $B = \{a'c', a'c'b', a'c'd'\}$. En effet, quelque soit l'appariement de chemins choisi, on trouve un chemin ξ dans l'arbre qui le représente. De plus tous les chemins entre la racine et les feuilles de l'arbre sont bien des appariements entre deux chemins de A et B (ex : le chemin en rouge dans la fig.4.3 correspond à un appariement entre les chemins acb de A et *a'c'b'* de B) :

$$\begin{aligned} App(A,B) = \quad & \{(ac, a'c'), \\ & (acb, a'c'b'), (acb, a'c'd'), (aba, a'c'b'), (aba, a'c'd'), \\ & (abc, a'c'b'), (abc, a'c'd')\} \end{aligned}$$

On remarque que les ensembles A et B contiennent des chemins avec une seule arête et des chemins avec deux arêtes. L'appariement des chemins d'une arête est représenté dans un arbre de ce type par des parcours de la racine au nœud de profondeur 1. Par exemple pour l'arbre de la figure 4.2, l'appariement de l'arête ac avec l'arête $a'c'$ (noté $\alpha(ac, a'c')$), correspond au parcours entre la racine ($s_v(a, a')$) et le nœud le plus à gauche parmi les fils de la racine : ($s_v(c, c')$).

N'importe quel parcours dans un arbre d'appariement ne représente pas forcément un des appariements des deux ensembles de chemins (A et B dans notre exemple). Dans

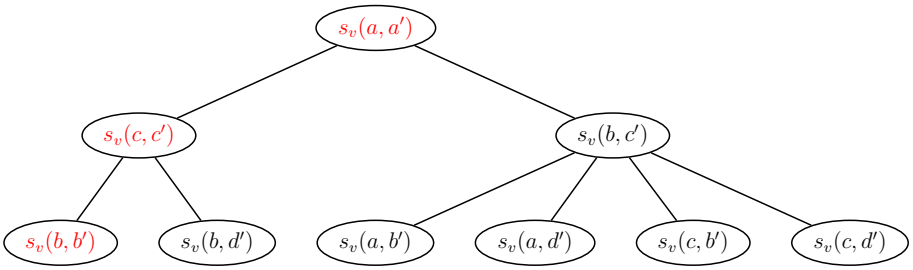


FIGURE 4.3 –

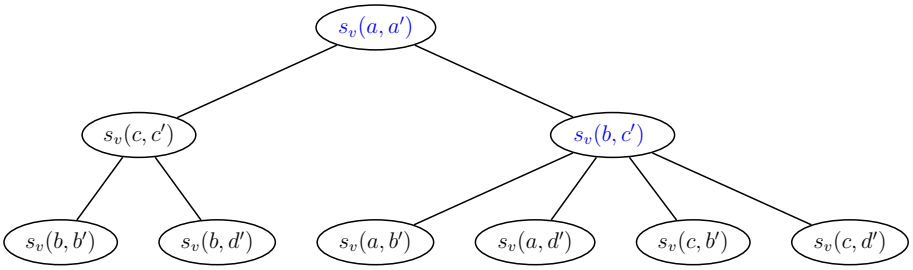


FIGURE 4.4 –

l'arbre de la figure 4.4, le chemin bleu représente l'appariement (ab,$a'c'$) qui n'est pas contenu dans l'ensemble $App(A,B)$.

Par contre, comme on peut le voir sur notre arbre d'exemple, tous les parcours de la racine jusqu'au nœud définissent un appariement appartenant à l'ensemble App (pour notre exemple $App(A,B)$). Il est plus intéressant de travailler uniquement sur les chemins entre la racine et les feuilles car nous sommes certains qu'ils définissent un appariement entre deux chemins issus des ensembles. Dans un cadre de construction de ces arbres de manière récursive, il est pertinent de travailler avec des ensembles qui nous assurent que les seuls appariements possibles entre les deux ensembles soient tous contenus dans les parcours de la racine aux feuilles de l'arbre (et uniquement dans ces chemins).

Soit A^k le sous-ensemble de A ne contenant que les chemins de longueur k dans A (la longueur k étant le nombre d'arêtes constituant le chemin). Cette notation est aussi utilisée dans le cadre des fonctions H (génèrent un ensemble de chemins pour un graphe donné) : H^k est une fonction qui ne donne que des chemins de k arêtes pour un graphe donné. Cette notation nous permet de simplifier l'écriture de propriétés par la suite.

Revenons à nos deux ensembles A et B ainsi qu'à l'arbre(fig.4.4) de l'exemple, pour les ensembles A^2 et B^2 nous avons :

$$\begin{aligned} App(A^2,B^2) = \quad & \{(acb,a'c'b'),(acb,a'c'd'),\\ & (aba,a'c'b'),(aba,a'c'd'),\\ & (abc,a'c'b'),(abc,a'c'd')\} \end{aligned}$$

On peut vérifier que tous les appariements entre A^2 et B^2 sont représentés par les

parcours de la racine aux feuilles de cet arbre et uniquement par ceux-ci. Dans les arbres de recherche, les solutions sont représentées par les feuilles de l'arbre ; le parcours de la racine jusqu'à une feuille est forcément unique et représente dans notre cas une solution d'appariement pour des chemins de longueur liée à la profondeur de la feuille dans l'arbre. Il nous reste à considérer le cas des ensembles issus de fonctions H^k pour s'assurer que l'espace de solution est bien complet et unique.

Nous allons voir que pour deux ensembles de chemins $H^k(G)$ et $H^k(G')$:

à chacun de leur appariement $\alpha(h, h')$ il existe un unique parcours $p(racine, feuille)$ le représentant dans l'ensemble $P(T(H^k(G), H'^k(G')))$.

$P(T(H^k(G), H'^k(G')))$ est l'ensemble des parcours de la racine aux feuilles de l'arbre d'appariement $T(H^k, H'^k)$.

Le parcours p peut se noter comme une séquence de couples de sommets puisqu'il représente un appariement de chemins. Les notations pouvant être utilisées sont :

$$p = (racine)(n_0)(n_k)$$

où n est un nœud l'arbre,

$$p = (racine)\binom{v_0}{v'_0} \cdots \binom{v_k}{v'_k},$$

$$p = \binom{h}{h'}.$$

Théorème 4.1.1. *Soit H une fonction générant un ensemble de chemins à partir de graphe.*
Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes.
Soit $T(H^k(G), H^k(G'))$ un arbre d'appariement
On note l'ensemble des parcours de la racine à la feuille d'un arbre T : $\Xi(T)$
Alors on a : $App(H^k(G), H^k(G')) = \Xi(T(H^k(G), H^k(G'))$.
Autrement dit :
Quelque soit l'appariement entre deux chemins h de $H^k(G)$ et h' de $H^k(G')$ il existe un parcours p de la racine à la feuille de l'arbre T tel que : $p = (racine)\binom{h}{h'}$
Quelque soit le parcours $p = (racine)\binom{v_0}{v'_0} \cdots \binom{v_p}{v'_p}$ de la racine à une feuille de l'arbre T, $\alpha = (v_0 ... v_p, v'_0 ... v'_k)$ est un appariement entre deux chemins de $H^k(G)$ et $H^k(G')$.

Ce lemme se démontre simplement(voir annexe A), il permet de nous assurer de l'exploration complète et unique des solutions(voir annexe A).

Il est possible de construire récursivement les arbres de comparaison, si les ensembles associés à ces arbres sont constructibles récursivement.

Nous avons donc une définition d'un arbre qui, pour une longueur de chemins donnée, va produire tous les appariements possibles de cette longueur. Par la suite nous représenterons uniquement le couple de sommets au lieu de leur similarité dans les arbres.

Notre but est d'obtenir un arbre de recherche pour les fonctions H qui produisent des ensembles de chemins à partir de graphes. Puisque ces fonctions sont couramment

utilisées dans les cadre de fonctions noyaux sur graphes. Pour cela nous allons introduire le terme de famille d'arborescences liées à une fonction H qui nous permettra de définir un arbre de recherche appliqué à l'espace des appariements entre chemins issus des ensembles $H(G)$ et $H(G')$ pour deux graphes G et G'.

4.2 Ensembles de chemins et arbres

Dans la section précédente nous avons travaillé sur une représentation d'arbre de recherche ne comprenant que des chemins de même longueur. Pour travailler sur l 'espace des appariements engendré par le triplet : graphe G , graphe G'et la fonction générative de chemins H, nous allons généraliser notre représentation et introduire des familles d'arbres. Les ensembles de chemins sont des éléments importants qui peuvent influer sur l'apprentissage et la complexité du calcul. Nous allons tout d'abord revoir les ensembles de chemins issus des fonctions génératives introduites à la section 2.1.2. Puis nous nous appuierons sur le théorème des arbres d'appariements pour définir des familles de forêts d'arbres d'appariements liées à un type d'ensembles de chemins particulier. Nous montrerons que ces familles, pour une fonction donnée et un couple de graphe donné, nous permettent d'obtenir un arbre de recherche représentant l'espace des appariements de chemins possibles sur lequel travaillent les fonctions noyaux présentées dans le chapitre précédent. Puis nous montrerons les applications d'exploration de l'arbre de recherche définies par cette famille d'arbres.

4.2.1 Ensembles de chemins

Nous avons besoin de manipuler tous les appariements possibles entre deux ensembles. On doit s'assurer en effet que nos arbres de recherche permettent d'explorer l'espace complet des appariements entre les deux ensembles de chemins $H(G)$ et $H(G')$.

Par exemple, la fonction $H_{sansccycle}$ renvoie un ensemble contenant tous les chemins sans cycle du graphe G. On utilise les notations suivantes :

- H^k fonction générant des chemins de longueur k,
- $H^{\leq k}$ fonction générant des chemins de longueur inférieure ou égale à k.

Une condition suffisante pour obtenir l'arbre de recherche couvrant l'espace complet des appariements entre deux chemins(de même longueur) issus des ensembles $H(G)$ et $H(G')$, est d'avoir des ensembles dits récursifs(voir chapitre 1 section 2.1.2). Autrement dit, il faut un ensemble A, dont tout chemin de longueur supérieure ou égal à k peut être construit à partir d'un chemin de taille inférieure appartenant à A. Cette propriété a déjà été exploitée dans la section précédente.

Ces ensembles peuvent être par conséquent divisés en sous-ensembles, ne contenant que des chemins de même longueur, ce qui donne pour un ensemble de chemins A dit récursif : $A = \cup_{k=0}^{\infty} A^k$

Une fonction générative H est dite récursive si elle produit pour tout graphe G un ensemble $H(G)$ qui est lui même récursif. En considérant l'ensemble d'arbres (noté F^k) , pour deux graphes G et G', représentant chaque appariement de chemins issus de $H^k(G)$

et $H^k(G')$, on s'aperçoit que pour tout arbre T^k(solutions d'appariement de longueur k), $k > 0$ il existe T^{k-1} à partir duquel on peut construire T^k grâce au caractère récursif de H. De ce fait l'arbre de recherche final peut être construit totalement ou partiellement en se basant sur des opérations récursives.

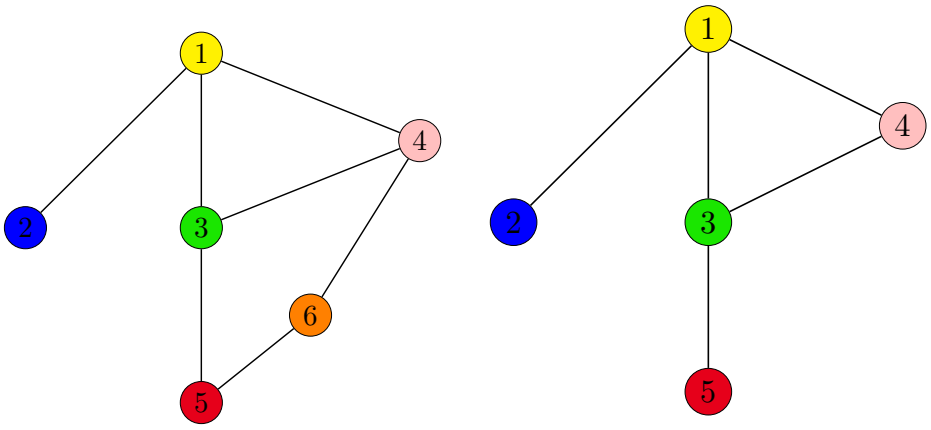


FIGURE 4.5 – Deux graphes pris pour illustrer les motifs induits par les noyaux sur graphes lors de leur calcul.

4.2.2 motifs d'appariement dans les graphes induits par les noyaux

Nous allons regarder pour quelques noyaux de graphe sur un exemple de deux graphes le poids des petits éléments dans le calcul du noyau et les motifs d'appariements produits si il y en a.

La figure 4.5 présente les deux graphes utilisés dans les exemples. Ces deux graphes possèdent un sous-graphe commun composé des sommets 1,2,3,4 et 5.

4.2.2.1 K_{max}

Ce noyau a bien pour équivalence un appariement de chemins. Par conséquent il couvre peu le graphe et ne garantit pas de passer par les sommets et les arêtes constituant l'objet.

4.2.2.2 K_{sum}

Ce noyau somme tous les appariements possibles entre les chemins de même longueur. Il recouvre avec seulement les arêtes pratiquement tout le graphe.

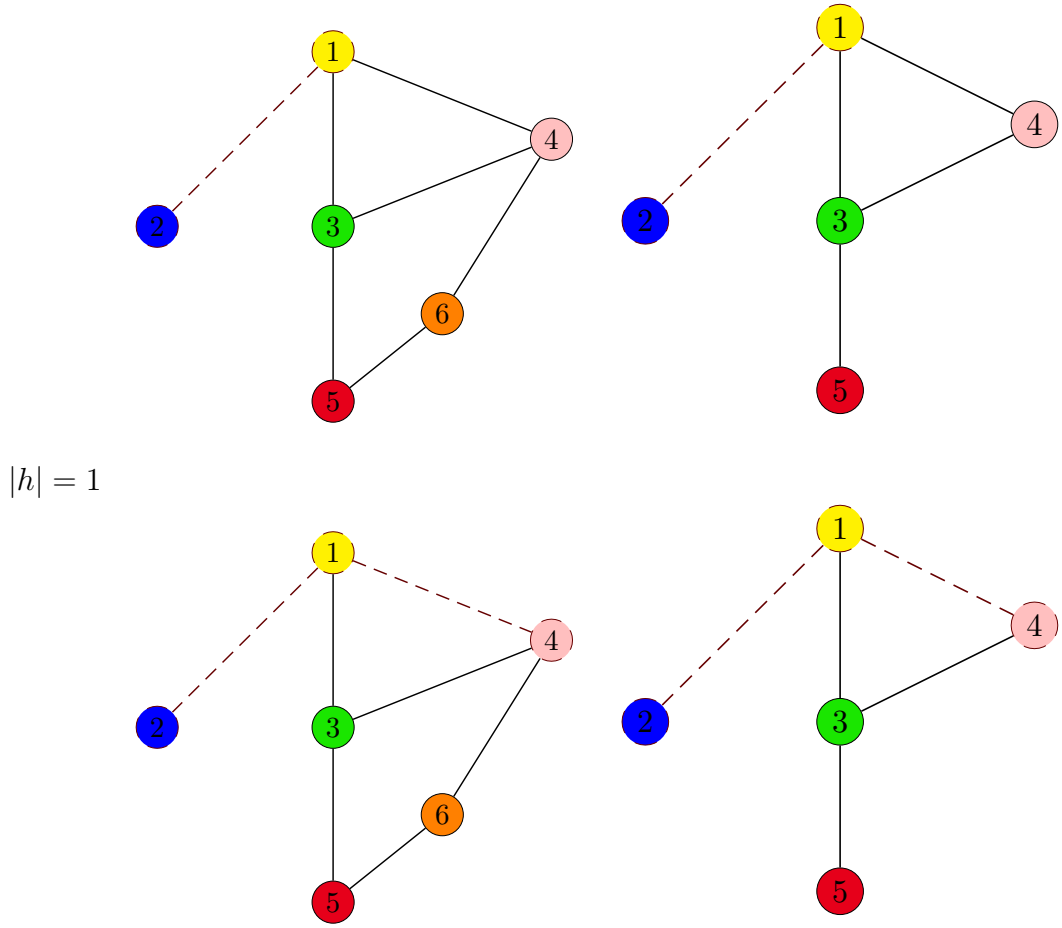
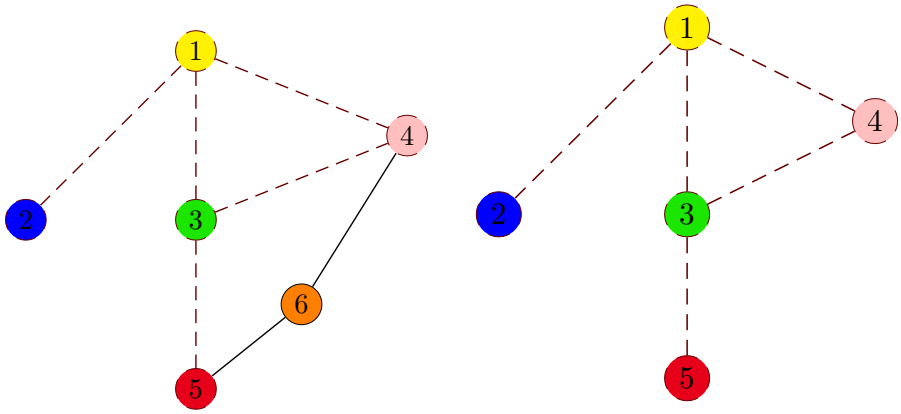


FIGURE 4.6 – K_{max}. Représentation des arêtes et sommets contribuant au calcul du noyau en trait non continu rouge foncé dans le cadre d'une hypothèse de noyau mineur se comportant de manière binaire. Plus les traits sont larges, plus les arêtes ou sommets ont contribué à ce calcul

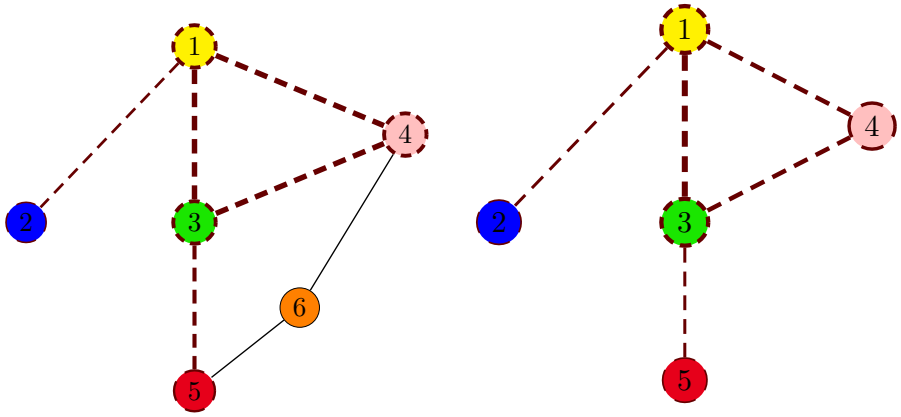
La figure 4.7 montre pour une hypothèse de comportement binaire des noyaux (0 si les éléments sont différents, 1 s'ils sont identiques). On s'aperçoit que les éléments non communs aux deux graphes ne comptent pas dans le calcul.

La figure 4.8montre pour une hypothèse de comportement non binaire des noyaux (strictement entre 0 et 1 si les éléments sont différents, 1 s'ils sont identiques). Dans ce cas tous les éléments des graphes participent au calcul.

Dans les deux figures on remarque que le nombre de voisins possibles des sommets influent sur leur participation au calcul. Plus le sommet est entouré de voisins plus il aura de poids dans le noyau.



$|h| = 1$



$|h| = 2$

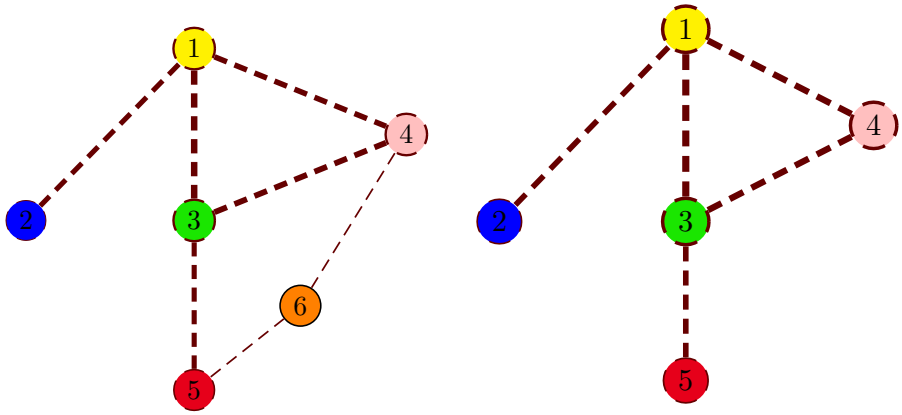
FIGURE 4.7 – K_{sum}. Représentation des arêtes et sommets contribuant au calcul du noyau en trait non continu rouge foncé dans le cadre d'une hypothèse de noyau mineur se comportement de manière binaire. Plus les traits sont larges, plus les arêtes ou sommets ont contribué à ce calcul

4.2.2.3 K_{new}

Contrairement au noyau précédent le sommet de plus forte valence n'est pas forcément celui qui compte le plus dans le noyau. Le noyau exploite bien une grande majorité du graphe dès la longueur 1.

4.2.3 Arbre de recherche pour ensembles de chemins

Les arbres d'appariements nous permettent d'explorer par des algorithmes l'espace des appariements entre deux ensembles de chemins. Par conséquent, il faut vérifier que nous pouvons explorer totalement cet espace d'appariements or on peut difficilement



$|h| = 1$

FIGURE 4.8 – K_{sum}. Représentation des arêtes et sommets contribuant au calcul du noyau en trait non continu rouge foncé dans le cadre d'une hypothèse de noyau mineur se comportement de manière non binaire. Plus les traits sont larges, plus les arêtes ou sommets ont contribué à ce calcul.

le garantir en n'utilisant qu'un seul arbre ; il est préférable de travailler sur une forêt d'arbres d'appariements cohérente pour les graphes comparés et la fonction générative d'un ensemble de chemins utilisée. Chaque arbre de cette forêt aura une profondeur donnée et on ne considère que les appariements définis par un chemin entre une feuille et la racine de l'arbre. Ainsi ils représentent tous les appariements liés à une longueur donnée de chemins. De ce fait, la forêt doit contenir tous les appariements avec ces arbres. De plus, l'hypothèse de récurrence sur les chemins nous permet la construction récursive des arbres de cette forêt. Suite à cette hypothèse, les appariements sont eux aussi récursifs et permettent cette récursivité au niveau des arbres de la forêt.

Une fois les forêts définies, nous passerons à la construction récursive de ces arbres dans ces familles.

4.2.3.1 Définition des familles d'arbres

En théorie, nous travaillons sur ces familles d'arbres pour calculer les similarités. En pratique, nous n'utilisons qu'un seul arbre auquel nous augmentons la profondeur quand c'est nécessaire grâce aux propriétés récursives. La famille d'arborescences doit donner pour deux graphes un ensemble d'arbres de différentes profondeurs deux à deux. De plus, on doit les construire par récursivité.

Définissons d'abord la famille d'arborescences de comparaison :

Définition 4.2.1. *Soit une fonction générative H .*
Soit F une forêt d'arborescences d'appariements.
On dit que F est une famille d'arbres d'appariements de H si et seulement si elle contient

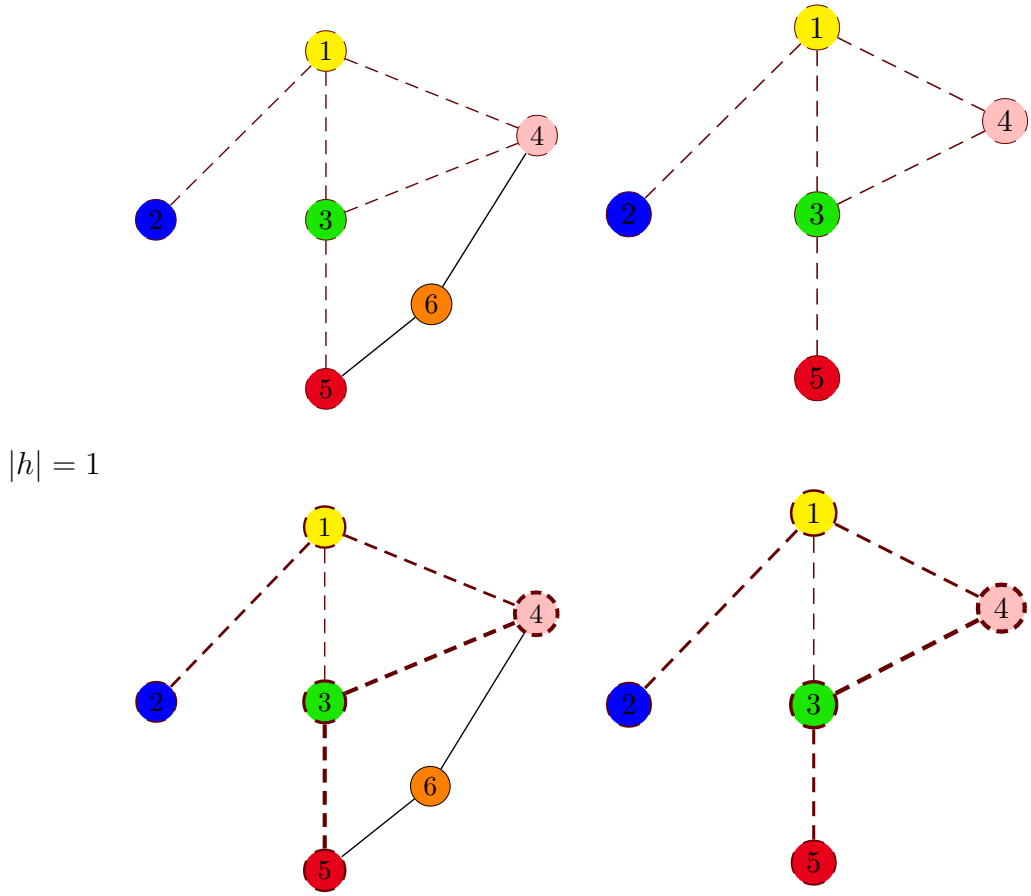


FIGURE 4.9 – K_{new} Représentation des arêtes et sommets contribuant au calcul du noyau en trait non continu rouge foncé dans le cadre d'une hypothèse de noyau mineur se comportement de manière binaire. Plus les traits sont larges, plus les arêtes ou sommets ont contribué à ce calcul

toutes les arborescences T liées à H et exclusivement celles-ci quelque soit les graphes considérés.

4.2.3.2 Construction récursive

Nous allons aborder la construction récursive d'un arbre dans une famille d'arbres donnée pour un couple de graphes donné. La validation de cette construction et l'exhaustivité de cette représentation seront détaillées dans la section suivante.

Dans un premier temps, nous reprenons les graphes de l'exemple de la section 4.1.1 pour illustrer la mise en place de cette construction récursive. Nous restreignons les chemins à ceux contenus dans l'ensemble des chemins sans cycle (fonction H_{sc}). Le

	G	G'
H^1_{sc}	ab, ba, bc, cb, ac, ca	$a'c', c'a', c'd', d'c', c'b', b'c'$
H^2_{sc}	abc, bac, bca cba, acb, cab	$a'c'b', d'c'a', b'c'a'$ $a'c'd', d'c'b', b'c'd'$

TABLE 4.2 – Tableau récapitulatif des chemins de longueur 1 et 2 contenu dans $H_{sc}(G)$ et $H_{sc}(G')$

tableau (4.2) récapitule les chemins de longueur 1,2 appartenant à cet ensemble.

Nous supposons que l'arbre d'appariements $T_1(G, G')$(figure 4.10) de la famille $F_{H_{sc}}$ est déjà construit. Nous construisons ensuite $T_2(G, G')$ de cette famille à partir de $T_1(G, G')$. Puis cette construction sera généralisée.

On prend la première feuille en partant de la gauche (b, c') qui correspond au couple de chemins : $ab, a'c'$. On cherche tous les voisins de b dans G et tous les voisins de c' dans G', a et c sont les voisins de b. a', b'etd' sont les voisins de c'. Ce qui nous donne comme chemins pour G : abc, aba ; et pour G' on a : $a'c'a', a'c'b, a'c'd'$. On élimine les sommets donnant des chemins n'appartenant pas à $H^2_{sc}(G)$ ou $H^2_{sc}(G')$: a et a'. On rajoute les nœuds à la feuille représentant tous les appariements possibles entre c et b',d' : c, b' et c, d'.

Pour toutes les autres feuilles, on applique le même fonctionnement :

1 On récupère les voisins des sommets de la feuille
2 On élimine ceux qui donnent des chemins cycliques (n'appartenant pas à H^2_{sc})
3 On rajoute les nœuds à la feuille représentant tous les appariements possibles entre les deux ensembles de sommets (vidés des sommets donnant des cycles).

Au final on obtient $T_2(G, G')$ de la famille $F_{H_{sc}}$. Cette construction est valable pour tout $T_{k+1}(G, G')$ à partir d'un $T_k(G, G')$ existant appartenant à la même forêt F.

4.2.3.3 Obtention d'un arbre de recherche pour les ensembles

La famille d'arborescence précédemment définie nous permet d'une part de la manipuler en arbre de recherche représentant tous les appariements entre deux ensembles de sommets issus d'une fonction générative et d'autre part de pouvoir le construire en même temps que son exploration si cette fonction est une fonction générative récursive. Cet arbre est constitué de cette famille d'arbres ceci étant possible du aux propriétés récursives de la famille d'arbres.

Cet arbre de recherche représente l'espace de solutions des appariements uniquement si la famille d'arbres qui le constitue, contient l'ensemble des solutions et uniquement celles-ci.

A partir de la définition des familles de forêts nous démontrerons l'exhaustivité et la validité de la construction récursive.

Nous introduisons le théorème suivant qui concerne l'exhaustivité.

Théorème 4.2.1. *Soit une fonction générative H.*

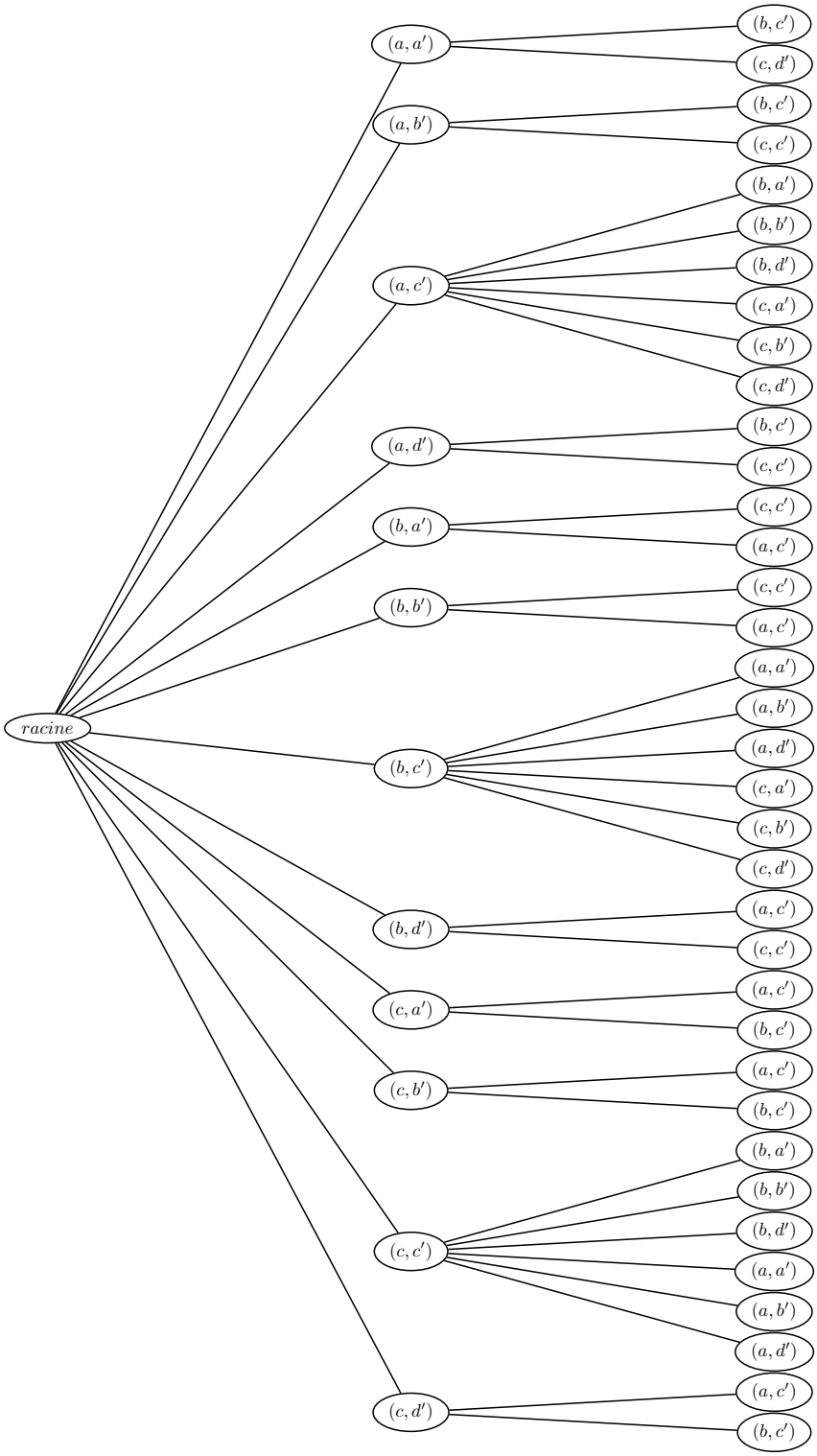


FIGURE 4.10 – $T_1(G, G')$ de la famille $F_{H_{sc}}$

Soit la famille d'arborescences F_H associée à H.
Pour tout couple de graphes G et G' on a :

$$App(H(G), H(G')) = \cup_{i=0}^{+\infty} \Xi(T_i(G, G'))$$

Avec T_i des arborescences de F_H restreintes à H^i.

Ce théorème justifie l'obtention de tous les appariements possibles.

Pour que nous puissions construire facilement un arbre $T_k(H_k, ., .)$ il faut nous assurer de la possibilité de le construire par une méthode récursive à partir de $T_{k-1}(H_{k-1}, ., .)$.

Théorème 4.2.2. *Soit une fonction générative H qui produit des ensembles de chemins récursifs, et la famille d'arborescence F_H associée à H.*
Alors, Pour tout couple de graphes G et G' on peut construire T_{k+1} à partir de T_k, G, G' et H_k

Les preuves de ces théoremes se trouvent en annexe. Nous obtenons un arbre lié à une définition de famille d'arbre, nous assurant de l'exhaustivité de l'espace de solution représenté par l'arbre ainsi que de l'unicité. Cet arbre est utilisé par la suite dans nos algorithmes présentés dans la section suivante.

4.3 Algorithmes

Nous présentons dans cette section des algorithmes qui s'appuient sur les outils que nous avons précédemment décrits pour calculer rapidement des similarités entre graphes, une fois la représentation implémentée par un type arbre. Ce type possède une méthode d'ajout d'un nœud qui calcule automatiquement la similarité en fonction d'opérateurs , et une méthode de création de l'arbre à la profondeur 1. L'intérêt de la représentation traitée dans les sections précédentes est de pouvoir mettre en œuvre simplement le calcul de différentes fonctions de similarité de graphes.

Le premier exemple traité montre la facilité de mise en place du calcul d'une fonction de similarité de type somme. Le deuxième exemple traite de la mise en place d'algorithme de recherche de maximum, appliquée au calcul d'un de nos noyaux de similarité présenté au chapitre précédent. Enfin nous verrons des implications et des notions générales pour l'utilisation d'algorithme dans cette représentation .

4.3.1 Exemple avec un noyau somme

Les noyaux à base de somme ont été utilisés dans le cadre des travaux de Kashima et de Suard. L'algorithme de calcul à l'aide de l'arbre de recherche a une formulation simple. En terme de complexité, Kashima a présenté une méthode non arborescente, de complexité moindre, mais qui suppose des contraintes sur les fonctions noyaux mineurs utilisés (k_v et k_e). L'utilisation d'arbre de recherche pour les sommes peut se discuter en terme de complexité si des convergences sont trouvées au niveau de la fonction de similarité sur graphes. Ces convergences peuvent permettre une expression du problème

en résolution de systèmes linéaires ce qui réduit la complexité. Par exemple, Kashima a utilisé des propriétés de convergence pour le résoudre comme un système linéaire. Néanmoins il est intéressant d'utiliser un algorithme utilisant l'arbre de recherche pour y voir la simplicité du calcul via cette méthode.

Pour présenter l'algorithme nous utilisons le noyau somme suivant :

$$K(G,G') = \sum_{h \in H(G)} \sum_{h' \in H(G')} K_C(h,h') \tag{4.1}$$

avec :

$$K_C(h,h') = k_v(v_0,v'_0) \prod_{i=1}^{|h|} (k_e(e_i,e'_i)k_v(v_i,v'_i))$$

H une fonction générative de chemins.

Pour rappel, on utilise la notation $K(G,G')$ alors que dans notre cas la fonction réelle est plutôt $K(H,G,G')$ puisque le choix de H influe sur la possibilité d'obtenir l'arbre et la complexité du calcul.

K_C peut s'écrire récursivement :

$$K_C(h_k,h'_k) = k_v(v_0,v'_0) \prod_{i=1}^{k-1} (k_e(e_i,e'_i)k_v(v_i,v'_i)) \times (k_e(e_k,e'_k)k_v(v_k,v'_k))$$

$$K_C(h_k,h'_k) = K_C(h_{k-1},h'_{k-1}) \times (k_e(e_k,e'_k) \times k_v(v_k,v'_k))$$

avec $h_k = h_{k-1}(e_k,e'_k)(v_k,v'_k)$. On définit, en s'appuyant sur la section 4.1 de ce chapitre, les trois opérateurs : op_v,op_e et op_{noeud}. Les opérateurs op_v et op_e se définissent par :

$$op_v = k_v$$

$$op_e = k_e$$

.

L'opérateur op_{noeud} est décomposable en deux sous opérateurs dans ce cas :

un opérateur (op_{n_1}) combinant la similarité entre h_{k-1} et h'_{k-1} et les similarités combinées des sommets et des arêtes nécessaires pour passer de h_{k-1} et h_{k-1} à h_k et h'_k soit $\times$ dans notre exemple,

un opérateur ($op_{n_2} = \times$) combinant les similarités des sommets et des arêtes nécessaires pour passer de h_{k-1} et h_{k-1} à h_k et h'_k .

$$op_{noeud}(n_{k+1}) = op_{n_1}(op_{n_2}(op_e(e,e'),op_v(v,v')),K_C(h_{k-1},h'_{k-1}))$$

Ces opérateurs ne sont utilisés qu'au moment de la création d'un nouveau nœud et ne sont donc pas présents dans le premier algorithme. Celui-ci suppose en effet qu'il existe une méthode initialisant l'arbre T_0 et une méthode créant l'arbre T_{k+1} à partir de T_k. Dans les algorithmes que nous décrivons par la suite, nous avons choisi de noter les similarités portées par un nœud n ou une feuille s comme des attributs d'objets dans certains langages : n.similarité ou f.similarité .

L'algorithme du calcul de la similarité $CalculSomme$(voir algo.1) prend en entrée les deux graphes G et G', la fonction générative d'ensembles H et un nombre réel (Borne). L'algorithme est simple, il crée un arbre de recherche T_0 pour H,G et G'. Puis il stocke la valeur de la somme des feuilles de cet arbre dans une variable $SommeArbre$. Puis tant que la différence entre cette somme et celle de toutes les similarités (la variable $Somme$ initialisée à 0) est supérieure à la borne : il additionne $Somme$ et $SommeArbre$ puis il développe l'arbre de un niveau. La somme des similarités des feuilles de cet arbre est mise dans la variable $SommeArbre$. Ainsi on obtient bien la somme de toute les similarités portées par toutes les feuilles de chaque arbre (de la famille correspondante) jusqu'à ce que l'apport soit négligeable.

Les parties non détaillées dans cet algorithme sont : la construction du premier arbre de profondeur 1. ($ConstruireArbre(H, G, G', 0)$) et l'obtention de l'arbre de profondeur augmentée de un. Les calculs de similarité des nœuds et des feuilles ne s'effectuent que lors de la construction de ceux-ci et ne sont donc pas présents dans l'algorithme $CalculSomme$.

La construction du premier arbre(algo.2) qui contient toutes les similarités entre les chemins de longueur 0, s'effectue en produisant tous les appariements possibles entre sommets qui sont des débuts de chemins dans $H(G)$ et $H(G')$. Puis pour chaque couple de sommets les rajouter comme nœud fils de la racine($root$) de l'arbre, avec leur similarité pour chacun. Enfin pour chaque nœud fils de la racine, leur rajouter tous les nœuds des couples de sommets produisant lors du parcours entre la racine et eux-même un appariement entre $H(G)$ et$H(G')$. En même temps on calcule la similarité portée par ces nouveaux nœuds.

Il ne nous reste plus qu'à définir la partie3 qui passe d'un arbre T_k à un arbre T_{k+1}, k étant supérieur ou égal à un. Comme précédemment, on va itérer sur les nœuds pour produire de nouveaux nœuds valides. Ces nœuds seront les feuilles de l'arbre. Ainsi la profondeur est augmentée de un.

Une amélioration possible de ce type d'algorithme, valable pour ce K_C, est de ne pas développer les fils d'un nœud dont la similarité est nulle. Ainsi on évite des multiplications inutiles, aboutissant à des similarités nulles. Une autre possibilité est d'effectuer le calcul en parallèle d'un certain nombre de branches de l'arbre.

4.3.2 Exemple avec le noyau de graphe proposé

Après nous être intéressés à la mise en place de calcul de similarité pour un cas simple, nous allons dans cette section aborder celle d'une fonction noyau que nous avons proposée dans le chapitre précédent. Ce noyau nous permet d'aborder l'exploration partielle de l'arbre à travers des algorithmes classiques de type recherche de maximum par profondeur d'abord ou de type "branch en bound".

4.3.2.1 Le noyau de graphe proposé

Le noyau considéré est un mélange de sommes et de recherche de maximums. Il se traduit par un choix au début de l'algorithme de $(v+v')$ nœuds. Ces nœuds représentent

Entrées : H une fonction générative, G et G' deux graphes
Sortie : un réel Somme
CalculSomme $(H, G, G', Borne)$
$T_0 : ConstruireArbre(H, G, G', 0)$
$SommeArbre = 0$
$Somme = 0$
$k = 1$
$Borne$: **un nombre réel fixant l'apport minimal autorisé**

Pour chaque feuille f de l'arbre T_0 **faire**
  | $SommeArbre = SommeArbre + f$.similarité
Fin Pour
Tant que ($|Somme - SommeArbre| > Borne$) **faire**
  | $Somme = Somme + SommeArbre$
  | Construire l'arbre T_k à partir de $T_{(k-1)}$, H, G et G'
  | *[rq : équivaut à (AugmenterArbre($T_{(k-1)}$, H, G et G'))]*
  | $SommeArbre = 0$
  | **Pour** chaque feuille f de l'arbre T_{Entier} **faire**
  | | $SommeArbre = SommeArbre + f$.similarité
  | **Fin Pour**
  | $Entier = Entier + 1$
Fait
Retourner $Somme$

Algorithme 1: Algorithme de calcul d'un noyau somme .Cet algorithme suppose l'existence de deux fonctions : le calcul de T_0 et la construction de T_{k+1} à partir de T_k. Le noyau calculé par cet algorithme est le K_{sum}. L'opérateur + sert à sommer les similarités des feuilles(f.similarité) entre elles à chaque arbre T puis à additionner ces sommes entre elles pour obtenir la valeur de cette formule : $\sum_{h \in H(G)} \sum_{h' \in H(G')} K_C(h, h')$.

Entrée :H une fonction générative, G et G' deux graphes
Sortie : un arbre T_0
ConstruireT_0
Soit T_0 un arbre de comparaison de profondeur 1
Créer T_0 avec une racine *root [le nœud root ne contient aucune information]*
Pour chaque sommet v dans $H(G)$ **faire**
  Pour chaque sommet v' dans $H(G')$ **faire**
    ajouter le nœud $n(v, v')$ au nœud *root* de T
    $n.similarit = k_v(v, v')$
  Fin Pour
Fin Pour

Algorithme 2: ConstruireT_0 construit un arbre de profondeur 1 qui ne comporte que les similarités de sommets dans notre cas.

les meilleurs appariements pour tous les sommets de G d'une part et tous les sommets de G' d'autre part. Il somme ensuite toutes les meilleures similarités issues de ces débuts d'appariements choisis. Ci-dessous l'équation de ce noyau :

$$\begin{aligned} K_{new}(G,G') &= \frac{1}{|V|} \sum_{\substack{v\in G \\ v'=s(v)}} \max_{h\in H_v(G)} \max_{\substack{h'\in H_{v'}(G') \\ |h'|=|h|}} K_C(h,h') \\ &+ \frac{1}{|V'|} \sum_{\substack{v'\in G \\ v=s'(v')}} \max_{h'\in H_{v'}(G)} \max_{\substack{h\in H_v(G) \\ |h|=|h'|}} K_C(h',h) \end{aligned} \tag{4.2}$$

avec,

$$s : V \to V'$$

$$s(v) = argmax_{x\in V'}(k_v(v,x)$$

et

$$s' : V' \to V$$

$$s'(v') = argmax_{x\in V}(k_v(x,v')$$

.

L'algorithme général 4 de ce noyau comporte deux parties. La première constitue la liste des nœuds initiaux, la deuxième à partir d'un nœud choisi et de son arbre trouve le meilleur appariement de chemins, de longueur k, représenté par un parcours dans l'arbre débutant par ce nœud. Au final toutes les similarités des feuilles représentant ces maximums sont sommées.

Entrées : Un arbre T , une fonction générative H et deux graphes G et G'
Sortie : T modifié
AugmenterProfondeur
$T_k(H, G, G')$: **un arbre de comparaison de profondeur k sur les ensembles** $(H(G), H(G'))$
Pour toute feuille $f(h, h')$ de T_k **faire**
  Pour tout e et v_2, hev_2 dans $H(G)$ **faire**
    Pour tout e' et v'_2, $he'v'_2$ dans $H(G')$ **faire**
      ajouter le nœud $fils(v_2, v'_2)$au nœud f
      fils.similarité= op_{n_1}(f.similarité, $op_{n_2}(op_e(e, e'), op_v(v_2, v'_2))$)
    Fin Pour
  Fin Pour
Fin Pour

Algorithme 3: Algorithme augmentant la profondeur d'un arbre de recherche T de un. Dans le cas du calcul de K_sum, on prend les opérateurs fixés précédemment. L'algorithme génère dans un premier temps des nœuds fils portants les appariements de sommets(v_2,v'_2) qui permettent de constituer dans $H(G)$ et $H(G')$ de nouveaux chemins hev_2 et $h'e'v'_2$. Diverses méthodes peuvent être utilisées en fonction de la fonction H concernée pour obtenir ces sommets(liste d'adjacences, liste tabou...). Une fois ce nœud créé on met à jour sa similarité (soit celle de l'appariement chemins qu'il représente).

Les recherches de maximum peuvent bien sûr être effectuées en parallèle (même si nous ne l'avons pas fait au niveau des expériences). Le choix d'algorithme performant pour la recherche de ces max dépend du comportement de la fonction K_C en fonction de la longueur des chemins considérés. On entend par là de savoir si la fonction va décroître à l'ajout d'une arête au chemin considéré. Cela revient à regarder si pour un nœud, la valeur de la fonction est toujours supérieure à celle portée par les nœuds fils.

C'est pourquoi, dans les paragraphes suivants, dans un premier temps, nous montrons une recherche par profondeur d'abord pour une fonction K_C décroissante avec la longueur du chemin. Dans un deuxième temps, nous effectuons une recherche par "branch and bound" pour des fonctions a priori non décroissantes.

4.3.2.2 Recherche du max dans le cas d'un K_C décroissant

On peut utiliser dans ce cas une recherche en profondeur d'abord d'un max et des coupures des branches de l'arbre. L'arbre est partiellement développé à la fin de l'algo-

Entrées : une fonction générative H, deux graphes G et G' et un entier K
Sortie : un réel SommeMax
CalculerNoyauSemiMax
Developper T_1 de H,G, G'
$listeNoeud$: **liste de nœud vide**

Pour tout $v \in V$ **faire**
  chercher $v' \in V'$ tel que $op_v(v, v')$ est maximisé
  stocker $noeud(v, v')$ dans $listeNoeud$
Fin Pour
Pour tout $v' \in V'$ **faire**
  chercher $v \in V$ tel que $op_v(v, v')$ est maximisé
  stocker $noeud(v, v')$ de T dans $listeNoeud$
Fin Pour
$SommeMax$: **0**
Pour tout nœud n de listeNoeud **faire**
  $SommeMax = SommeMax + ChercherMax(n, T, H, G, G', K)$.similarité
Fin Pour
retourner sommeMax

Algorithme 4: Algorithme général de calcul du noyau K_{new}. Il recherche dans un premier pour chaque sommet de G et G' son meilleur appariement selon l'opérateur op_v choisi(dans le cas du noyau k_v). Ensuite il crée les nœuds correspondant à ces appariements ($|V|+|V'|$ nœuds). A chacun de ces nœuds, on applique l'algorithme $ChercherMax$ (algo.6) pour obtenir la meilleure similarité entre chemins commençant par la paire de sommets de ce nœud. Une variante possible est de récupérer l'appariement lui même pour savoir quels chemins ont été les mieux appariés selon les opérateurs choisis.

ChercherMax(G,G',H,K : entier)
Entrées : une fonction générative H, deux graphes G et G' et un entier K
Sortie : un noeud d'arbre meilleurnoeud
T =arbre d'appariement développé à la profondeur 1 pour H, G et G'
n = meilleur nœud de T à la profondeur 1

Noeud meilleurnoeud=RechercherMeilleurNoeud(n,T,H,G,G',K)

MeilleurNoeud=RechercherAutresolution(
MeilleurNoeud,
meilleurnœud,
T,H,G,G',K,profondeur(n))
Retourner MeilleurNoeud

Algorithme 6: ChercherMax(G,G',H ,K)

rithme.

A l'initialisation on construit $T_0(H, G, G')$ et initialise n comme le noeud portant le meilleur appariement de sommet. On cherche un premier maximum à une profondeur k(algo.7). Pour cela on choisit le fils le plus prometteur à chaque fois jusqu'à la profondeur k. Une fois cette première estimation trouvée, on cherche s'il existe un autre maximum en remontant dans l'arbre (algo.8) pour tous les autres nœuds non visités dont la valeur est supérieure au maximum actuel pour la profondeur k. Si un nouveau maximum est trouvé il devient le maximum actuel.

Par exemple le K_C du précédent exemple est décroissant en fonction $|h|$:

$$K_C(h, h') = k_v(v_0, v'_0) \prod_{i=1}^{|h|} k_e(e_i, e'_i) k_v(v_i, v'_i))$$

en supposant que $k_e < 1$ et $k_v < 1$.

Plus on augmente la profondeur de l'arbre, plus les valeurs sont petites. Et la valeur

de similarité d'un nœud fils est toujours inférieure à celle du nœud père.

Entrées : un noeud N, un arbre T, une fonction générative H, deux graphes G et G', un entier K
Sortie : un noeud d'arbre MAX
RechercherMeilleurNoeud(N,T,H,G,G',K)
MAX : **une variable de type Noeud**
$MAX = NOEUDVIDE$
Pour chaque couple de $(ev, e'v')$ **faire**
  Si (hev appartient à $H(G)$ et $he'v'$ appartient à H(G')) **Alors**
    ajouter le nœud $f(v, v)$ comme fils de N
  Fin Si
  f.similarité=N.similarité$*k_e(e,e')k_v(v,v')$
  Si (MAX = VIDE) **Alors**
    MAX=f
  Sinon
    Si (f.similarité > MAX.similarité) **Alors**
      ALORS MAX=f
    Fin Si
  Fin Si
Fin Pour
Si (f.profondeur < K) **Alors**
  MAX=RechercherMeilleurNoeud(MAX, T, H, G, G', K)
Fin Si
Retourner MAX

Algorithme 7: *RechercherMeilleurNoeud*, cet algorithme renvoie soit un noeud avec une similarité plus élévée que N, soit N lui même. Il effectue sa recherche en profondeur et développe qu'un seul noeud de N.

Entrées : deux noeuds N et MAX, un arbre T, une fonction générative H, deux graphes G et G', deux entiers i et K
Sortie : un noeud d'arbre MAX
RechercherAutresolution(MAX : Noeud,N : Noeud,T,H,G,G',K : entier ,i :entier)

```
Si ( N.profondeur > i) Alors
    Noeud Ancetre=N.parent
    Pour  tout fils f de Ancetre  faire
        Si ( f.profondeur<K
        et f non développé
        et f.similarité >MAX.similarité) Alors
            Noeud solPart=RechercherAutreConcurrent(MAX, f, T, H, G, G', K)
            Si ( solPart.similarité > MAX.similarité) Alors
                MAX=solPart
            Sinon
                Si ( f.profondeur=K et f.similarité >MAX.similarité)
                Alors
                    MAX=f.similarité
                Fin Si
            Fin Si
            MAX=RechercherAutreSolution(Max, Ancetre, T, H, G, G', K, i)

        Fin Si
    Fin Pour
Fin Si
Retourner MAX
```

Algorithme 8: *RechercherAutreSolution*, cet algorithme va chercher une autre solution possible en développant uniquement les nœuds qui peuvent avoir un de leur fils avec une valeur de similarité plus élevée que la solution courante MAX.

Entrées : un noeud MAX, un arbre T, une fonction générative H, deux graphes G et G', deux entiers i et K
Sortie : un noeud d'arbre MeilleurNoeud
RechercherAutreConcurrent(MAX, T, H, G, G', K, i)
MeilleurNoeud : **Noeud**
$MeilleurNoeud = RechercherMeilleurNoeud(n, T, H, G, G', K)$

Si (MAX<MeilleurNoeud) **Alors**
  | MAX=MeilleurNoeud
Fin Si
MeilleurNoeud=
RechercherAutreSolution(Max,meilleurnœud,T,H,G,G',K,n.profondeur,i)
Retourner MeilleurNoeud

Algorithme 9: *RechercherAutreConcurrent*.

4.3.2.3 K_C non décroissant (branch and bound)

Dans ce cas on ne peut plus appliquer telle quelle la recherche de solution précédente. Le branch and bound (séparer et borner) consiste à s'appuyer sur des fonctions d'estimation, que l'on note b_{min} et b_{max}, d'un nœud et de borne pour éliminer les branches inutiles. Ces fonctions doivent permettre de borner les résultats de similarité possibles dans les noeuds fils. Elles doivent se calculer facilement sans nécessiter de développer l'arbre. Le choix au niveau du nœud n'est plus basé uniquement sur la similarité portée par le nœud mais celle de b_{min},b_{max} et de la valeur de similarité du nœud max actuel. Si cette fonction de borne est toujours supérieure à la similarité pour un nœud donné alors la recherche du maximum est exhaustive. Sinon certaines branches de l'arbre qui peuvent contenir la solution finale sont écartées.

On cherche à construire b de façon à borner K_C. On suppose que nos noyaux mineurs sont bornés par une valeur α (i.e $k_v(v,v) <= \alpha$ et $k_e(e,e) <= \beta$). Dans ce cas on pose :

$$b(n(h,h'),k) = n.\text{similarité} + k_C(h_(k-|h|), h_(k-|h|)) - k_v(v,v)$$

"b" renvoie une valeur hypothétique supposant que l'appariement des sommets et des arêtes soient parfaits par la suite. Prenons ce K_C non monotone :

$$K_{C_{new2}}(h,h') = K_V(v_0,v'_0) \times \prod_{i=1}^{|h|} K_E(e_i,e'_i) \times (1 + K_V(v_i,v'_i)) \tag{4.3}$$

$$b(n(h,h'),k) = n.similarit + \alpha + \prod_{i=1}^{k-|h|} ((1+\alpha) * \beta) - \alpha$$

La fonction b est calculable aisément pour tout nœud. Il s'agit de modifier la fonction *RechercherAutreSolution* pour utiliser cette fonction et éliminer les développements inutiles.

```
Entrées : deux noeuds N et MAX, un arbre T, une fonction générative H,
deux graphes G et G', deux entiers i et K
Sortie : un noeud d'arbre MAX
RechercherAutreSolution
T : un arbre de comparaison
MAX :Noeud de T avec la meilleur similarité
N(h,h') : Noeud de T
H : fonction generative
G,G'deux graphes
K,i : entier

Si ( N.profondeur > i) Alors
    Noeud Ancetre=N.parent
    Pour  tout fils f de Ancetre faire
        Si    ( f.profondeur<K    et    f    non    developpé    et
        (f.similarité+b(f))>MAX.similarité) Alors
            Noeud solutionPartielle=RechercherAutreConcurrent(MAX,f,T,H,G,G',K)
            Si ( solutionPartielle.similarité > MAX.similarité) Alors
                MAX=solutionPartielle
            Sinon
                Si  ( f.profondeur=K  et  f.similarité  >MAX.similarité)
                Alors
                    MAX=f.similarité
                Fin Si
            Fin Si
        Fin Si
    Fin Pour
    MAX=RechercherAutreSolution(Max,Ancetre,T,H,G,G',K,i)

Fin Si
RETOURNER MAX
```

Algorithme 10: *RechercherAutreSolution*$(MAX, N, T, H, G, G', K, i)$ variante ; cet algorithme inclut des coupures issues du "branch and bound".

4.3.3 Complexités des algorithmes

La complexité du calcul dans notre représentation dépend d'une part de l'espace considéré, des fonctions de calculs impliquées et des algorithmes utilisés.

Si on prend le cas du calcul d'un noyau de graphe de type somme, la complexité est égale à l'espace des solutions soit l'ensemble des couples possibles entre deux chemins de G et G' de longueur k. A chaque niveau de l'arbre on calcule de nouveaux nœuds autant de fois qu'il y a de chemins de longueur égale à ce niveau. Par exemple, au niveau 0 on obtient tous les appariements de sommets et au niveau 1 tous les appariements d'arêtes possibles.

La complexité du niveau 2 peut être bornée dans un premier temps par le nombre de sommets des graphes et des arêtes : le nombre de nœuds au niveau 1 ($|E|.|E'|$) fois le nombre d'appariements possibles entre les sommets ($|V|.|V'|$). Cependant on peut affiner cette borne en utilisant le voisinage des noeuds du graphe. En effet le nombre de nœuds fils possibles pour le nœud(h, h') dépend des degrés(nombre de voisins directs) des sommets finissant les deux chemins h et h'. Au final il est possible d'approximer cette borne en utilisant le degré moyen du graphe(notons la d_{moy}). Le coût de calcul d'un noyau somme pour une longueur k, pour des graphe d'ordre n($|V| = n$) et de degré moyen d_{moy}, peut s'approximer par : $n^2.d_{moy}^{2k}$.

Si on impose des restrictions sur les ensembles de chemins considérés cela revient à remplacer le d_{moy} par un autre paramètre qui est le nombre moyen de sommets voisins utilisables et par conséquent de réduire le nombre de voisins et donc la complexité considérée.

Pour les graphes complets le degré moyen est : $n-1$. Dans ce cas la complexité devient de l'ordre de n^{2k}. Pour le noyau proposé on réduit les choix initaux de n^2 couples de sommets à $2n$ couples de sommets l'approximation devient :$2n.d_{moy}^{2k}$ et dans le cas d'un graphe complet on a : $(2n)n^{2(k-1)}$. La complexité au pire d'une recherche de maximum est semblable à la somme. Dans le cadre d'exploration d'arbre on s'intéresse au coût moyen de l'algorithme. Nous appuyerons sur les résultats de temps calculs dans nos experiences pour en produire une estimation de ce coût moyen dans le chapitre suivant pour un ensemble de graphes dont les sommets varient entre 15 à 25 sommets et un nombre moyen de voisins compris entre 4 et 5.

4.3.4 Généralités

Nous avons désormais une représentation des appariements sous la forme d'une forêt. Dans cette section nous allons voir la manipulation classique de cet arbre selon les types d'opérations utilisées dans nos fonctions de similarité. Ces fonctions présentent plusieurs niveaux d'opérations :

- les noyaux sur sommets et arêtes,
- les noyaux sur chemins,
- les sélections des similarités de chemins,
- les combinaisons des similarités,
- les restrictions de la taille des ensembles de chemins considérés.

Nous présenterons les 3 derniers types d'opérations puisque nous avons déjà présenté comment effectuer le calcul de similarité sur chemins et que le calcul des similarités d'arêtes et de sommets n'est pas le propos du chapitre.

4.3.4.1 Opérations autour des ensembles de chemins

Les fonctions de similarité de sacs de chemins sélectionnent et combinent des appariements de chemins en fonction de leur valeur de similarité. Ils sont dotés pour cela d'une fonction donnant une valeur de similarité entre chemins. Cette catégorie de fonction de similarité sur graphes se prête de manière générale bien à notre représentation. Une des conditions nécessaires est que la fonction de similarité entre chemins se plie aux règles que nous avons précédemment étudiées.

Restrictions de chemins Une restriction de chemins dans notre représentation est donnée par la fonction générative de chemins H. Par exemple pour que H_{sc} génère des chemins sans cycle dans les graphes, il est possible d'utiliser une fonction de test simple vérifiant qu'il n'existe pas deux fois le même sommet dans les chemins. Dans le cas de la génération des chemins en même temps que la construction de l'arbre, il suffit de vérifier que le nouveau sommet ajouté pour chaque graphe n'est pas déjà présent dans les chemins. Les opérations de restrictions de l'espace d'appariements hors conditions sur la similarité correspondent à utiliser des ensembles ou sacs de chemins à moindre cardinalité. Les opérations de sélection concernent, quant à elles, des choix d'appariements (par exemple le max) dont les valeurs de similarité seront par la suite combinées entre elles. Les combinaisons sont les opérations sur les valeurs entre les appariements choisis : somme, moyenne, produit...

Restrictions de sacs de chemins Les restrictions de sacs de chemins correspondent à un choix de fonction génératrice H. L'intégration de génération d'un chemin appartenant à H dans l'arbre peut se faire à partir de fonctions déterminant(cf algo.11)

si le chemin peut être généré par H.

Entrées : un arbre T, une fonction générative H, deux graphes G et G'
Sortie : T modifié
H : **fonction générative**
G et G' : **deux graphes**
n : **un nœud** $n = (h, h')$ **un arbre** $T(H, G, G')$
$estGenereeparH(h : chemins)$:
une fonction qui renvoie vraie
si h **peut être généré par** H.
listAdjG et listAdjG : **les listes d'adjacences de** G **et** G'.
v : **le dernier sommet de** h
v' : **le dernier sommet de** h'

Pour élément e de listAdj[v] **faire**
  Pour élément e' de listAdj[v'] **faire**
    soit l'arête $a = (v, e)$
    soit l'arête $a' = (v', e')$
    Si ($estGenereeparH(hae)$ et $estGenereeparH(h'a'e')$ sont vrais)
    Alors
      rajouter un nœud fils f à n tel que $f = (hae, h'a'e)$
    Fin Si
  Fin Pour
Fin Pour

Algorithme 11: Simple algorithme pour la restriction à des chemins issus de $H(G)$ et $H(G')$

Les restrictions de chemins possibles doivent pouvoir être liées à une fonction H telle que $H(G)$ ait des propriétés récursives. Le choix de $H(G)$ a une influence directe sur la largeur de l'arbre exploré.

Sélections d'appariements de chemins La sélection d'appariements peut s'effectuer soit sur les valeurs des similarités associées aux appariements, soit par d'autre contraintes. C'est celle qui aura le plus d'influence quant au choix d'exploration de l'arbre d'appariement. Une sélection de type maximum sera couplée avec des algorithmes de "branch and bound" ou recherche de min/max.

De ce fait, le type de sélection choisie fait varier la complexité. Les sélections qui identifient les différents chemins (exemple : chercher le meilleur appariement pour un chemin donné) augmentent la complexité par le mécanisme d'identification des chemins.

Notons que si l'on veut obtenir des fonctions noyaux il faudra rendre symétrique cette sélection d'appariements.

4.3.4.2 Pondérations

Les pondérations peuvent être soit directement introduites dans les fonctions de similarité et calculées au cours du calcul de la similarité, soit calculées de manière séparées afin de les utiliser comme éléments de coupure dans l'arbre.

Par conséquent, il est préférable de choisir des pondérations répondant aux mêmes critères que les fonctions de similarité. Par exemple les fonctions de probabilité sont de bonnes candidates pour les pondérations. De plus elles ont déjà été introduites pour des fonctions noyaux autour des graphes.

4.3.4.3 Cas particuliers

Effectuer des appariements de chemins de longueurs différentes Dans le cadre d'appariement de chemins il devient intéressant d'apparier des chemins de longueurs différentes pour pallier aux problèmes liés aux fusions ou aux divisions de sommets. Ces problèmes sont courants dans la segmentation en régions qui peut sur deux images très similaires découper une même zone en une région sur l'une des images et en deux sur l'autre. Dans ce cas les images ne possèdent pas le même graphe alors qu'elle sont proches.

Les appariements qui sont effectués dans le cadre de nos arbres sont exclusivement entre chemins de longueur semblable. Nous avons trouvé une solution permettant dans notre formalisme de comparer des chemins de longueur différente.

Nous avons donc introduit une arête particulière, la boucle (notée *loop*), ayant les mêmes propriétés quel que soit le sommet considéré. La boucle se traduit dans un chemin par la redondance d'un sommet : aab soit $a\ loop\ a\ e_{a,b}b$ (en affichant les arêtes).

Ainsi nous pouvons apparier les chemins ab et $a'b'c'$:

- aab avec $a'b'c'$
- abb avec $a'b'c'$

Nous avons plusieurs appariements possibles qui correspondent à des choix de fusion (par exemple $a'b'$ donnant a ou $b'c'$ donnant b). La valeur de la similarité s_h de ces appariements détermine l'appariement considéré comme le plus proche.

Nous imposons la propriété suivante aux fonctions de similarité entre arêtes :

Pour tout graphe G,

$\forall e \neq loop,\ s_e(loop, e) = s_e(e, loop) = \delta$

avec δ un paramètre.

On impose donc une similarité identique quelle que soit l'arête comparée à *loop*. Ceci revient à mettre la boucle dans une position particulière dans l'espace des arêtes. Notons

que si $\delta < min(s(e, e'))$ alors on rend pratiquement impossible l'appariement d'arêtes boucle avec non boucle. Alors que l'inverse $\delta = max(s(e, e'))$ favorise cette situation.

Dans le cadre de similarité normalisée ($s_e(e, e) = constante$), si δ est strictement égal à cette constante, alors une boucle se comporte comme un élément neutre. Le coût de la fusion ou division dans ce cas est porté uniquement sur les sommets.

Cette solution a été testée dans nos expérimentations présentées dans le chapitre suivant.

Prise en compte de l'information de la non adjacence Dans notre cadre de recherche qui est la mise en correspondance de graphes issus de segmentation sur les images à comparer, on peut voir le fait que deux régions soient non adjacentes comme une information supplémentaire. Or nos graphes utilisés sont des graphes d'adjacences. Une solution envisageable quoique possiblement coûteuse, est de considérer un graphe complet issu du graphe d'adjacences. Ce graphe est constitué de deux type d'arêtes : celle représentant une adjacence et les autres.

Afin d'éviter un coût en complexité on peut imposer l'impossibilité d'apparier une arête d'adjacence avec une arête de non-adjacence. Cette opération est une opération de sélection d'appariement et donc de coupure dans l'arbre. Nous n'avons pas effectué d'expériences autour de cette représentation d'image.

Conclusion

Nous avons présenté des algorithmes de calculs pour nos noyaux sur graphes basés sur des représentations arborescentes de l'espace d'appariement. Afin de garantir que cette représentation pouvait être utilisée comme un arbre de recherche et que les calculs effectués étaient exacts, nous avons formalisé cette représentation. Cette représentation est exhaustive et chaque appariement de chemins y est unique. De plus, elle ne contient pas d'appariement non existant.

Nous avons montré dans ce cadre les diverses utilisations d'algorithmes et notamment spécifié une catégorie de fonction de borne pour l'algorithme "branch and bound" en fonction du noyau sur chemin choisi.

Enfin nous avons présenté quelques modifications possibles pour intégrer des pondérations, des appariements de chemins de longueurs inégales et la prise en compte de l'information de non adjacences.

Chapitre 5

Évaluation des noyaux de graphes

Dans le cadre de cette thèse les outils d'appariement inexact de graphe précédemment introduits(noyaux de sacs de chemins) sont employés pour apparier des graphes représentant des images ou des objets 3D. Nos expérimentations servent à étudier le comportement des noyaux sur chemins lors de leur utilisation et leur capacité à permettre de retrouver des objets ou images similaires dans une base de données. Le cadre choisi pour étudier ce type de noyaux et leur capacité à retrouver des images ou objets similaires est la recherche interactive par contenu avec apprentissage. Les noyaux sont testés en les combinant à un SVM. Les SVM calculent des séparateurs à vastes marges en s'appuyant sur des fonctions noyaux afin de s'affranchir des problèmes non linéairement séparables dans l'espace initial.

Dans ce cadre, le MAP(courbe de Mean Average Précision) permet d'effectuer des évaluations sur des bases de données, dès lors que l'on a la vérité terrain. Ces courbes permettent de comparer les méthodes à noyaux avec d'autres fonctions ou méthodes et d'affiner ou de justifier un paramétrage.

De plus le noyau sur graphe ou ensemble de chemins(ou chemins) est composé de noyaux sur les sous-éléments du graphe qui peuvent influer sur le résultat. De ce fait le bon résultat d'apprentissage d'un noyau de graphe par rapport à un autre, sur un même SVM, doit-il être imputé aux noyaux sur arêtes, sommets, chemins ou à la combinaison de ceux-ci. Nous avons choisi aussi d'étudier par niveau les différents paramètres des noyaux sur graphes ainsi que leurs combinaisons.

Pour mieux aborder les expériences nous allons dans un premier temps présenter leur protocole général en décrivant les bases de données utilisées, l'outil d'expérimentation ReTIN et les méthodes de comparaison choisies Puis nous aborderons la représentation d'une image ou d'un objet 3D sous forme de graphes étiquetés par des descripteurs complexes sur les arêtes et les sommets. Enfin les diverses expériences seront vues suivant une étude progressive des éléments constituant le noyau sur sacs de chemins.

FIGURE 5.1 – Exemples d'images issues de la base COLUMBIA

5.1 Protocole d'évaluation

Nous allons décrire le protocole utilisé dans le cadre de nos expérimentations en commençant par les bases d'images et d'objets. Puis nous verrons l'outil d'expérimentation utilisé (ReTIN). Enfin nous verrons la méthode choisie pour comparer les résultats.

5.1.1 Description des bases

Lors des expériences, nous avons utilisé 5 bases d'images orientées recherche d'objet et une base d'objets 3D. Pour toutes les bases que nous allons présenter nous avons pris le soin de calculer les matrices de Gram pour les noyaux non Mercer que nous avons utilisés(le maximum et la somme de maximum). Ces matrices se sont toujours révélées définies positives quelle que soit la base.

5.1.1.1 COLUMBIA

La base est constituée de 6000 images détaillant 100 objets différents sous diverses vues. Ces vues sont obtenues par des rotations axiales verticales. L'arrière-plan est le même pour tous les objets : un fond noir(fig. 5.1).

5.1.1.2 COLUMBIA+ANN

COLUMBIA+ANN est une base de donnée de 600 images construites. Les images sont le résultat d'une fusion entre deux images de deux bases de données différentes : COLUMBIA,ANN. Une vue de paysage issue de ANN forme l'arrière-plan de l'image. On superpose une image d'un objet de COLUMBIA, privé de son fond noir sur cet arrière-plan. Pour construire cette base, 12 vues de 50 objets de COLUMBIA ont été placées sur des images aléatoires de ANN. Ainsi cette base permet de tester si une technique est capable de chercher un objet sans s'appuyer uniquement sur des informations issues de l'arrière-plan. Cette base contient donc 50 catégories différentes constituées de 12 images chacune.

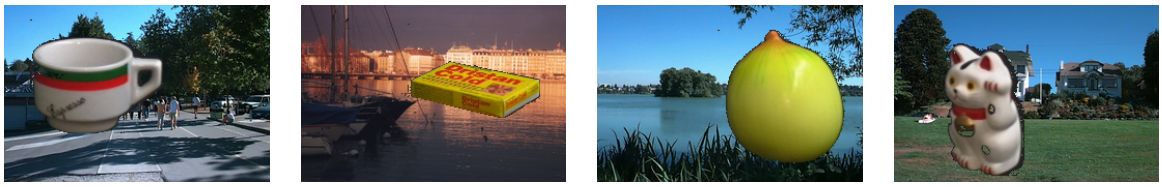


FIGURE 5.2 – Exemples d'images issues de la base COLUMBIA+ANN

FIGURE 5.3 – Exemples d'images issues de Caltech

5.1.1.3 Caltech

La base Caltech est une base issue des challenges PASCAL. Cette base contient 4620 images et les catégories suivantes : avions, voitures, visages, motos, arrières-plans de routes et arrières-plans génériques.

5.1.1.4 VOC

Les bases VOC sont des bases d'images généralistes standardisées autour de la recherche d'objets dans le cadre des challenges PASCAL. Les expériences ont été menées autour de la base VOC de 2006. La base contient 5304 images regroupées en diverses catégories d'objets tels que : bicyclettes, autobus, chats, voitures, vaches, chiens, chevaux, motos, gens, moutons.

5.1.1.5 Birds

La base Birds est une petite base de 600 images, composée de six catégories correspondant à des espèces distinctes d'oiseaux.

FIGURE 5.4 – Exemples d'images issues de VOC2006

FIGURE 5.5 – Exemples d'images issues de Birds

5.1.1.6 Eros 3D

La base est issue du projet EROS-3Dhttp://eros3d.ensea.fr/. Elle contient des objets 3D représentant des œuvres d'art ou moules : moules de statuettes, vases, statuettes, fragments de statuettes ou d'autres objets d'archéologie. Elles contient 700 objets et les catégories suivantes : déesses mères, Venus, vases ...

5.1.2 Outil d'expérimentation RETIN

ReTIN(fig.5.8) est un logiciel du laboratoire ETIS visant à mettre en place un cadre pour la recherche interactive dans des bases de données. Au départ créé pour des bases d'images, il a été étendu à d'autres documents multimédia objets 3D, vidéos ... Il utilise des bases de données sur lesquelles les phases d'extraction et d'indexation ont été effectuées. Il est possible de tester différentes méthodes à base de SVM pour la recherche d'images. On peut l'utiliser de manière graphique (fig.5.10 et fig.5.9) pour une réelle recherche interactive ou en ligne de commande afin de simuler des recherches interactives d'utilisateur. La figure 5.9 présente l'interface de ReTIN pour les objets 3D. La partie bleue permet de voir la base. À droite se situe l'emplacement pour la visualisation de l'attribut d'un objet. En bas, se trouvent les propositions d'objets à étiqueter de la part de ReTIN.

Session de recherche sous ReTIN Une session de recherche avec l'interface graphique de ReTIN se lance en effectuant une sélection d'images labellisées (positivement ou négativement). Si on a une seule requête, l'outil renvoie une première réponse ordonnée selon la distance par ordre croissant avec cette requête. En interne, il essaie de trouver une séparation des deux classes(positives et négatives). L'utilisateur a de nouveau la possibilité de labelliser des images de la base. ReTIN ayant un algorithme d'apprentissage dit recherche active, il propose de plus une sélection d'images à l'utilisateur ayant pour but d'accélérer la détermination de la frontière par l'algorithme d'apprentissage. Une fois le choix d'exemples négatifs et positifs effectué, le système recalcule les frontières et l'ordre des images. L'utilisateur arrête les itérations de recherche

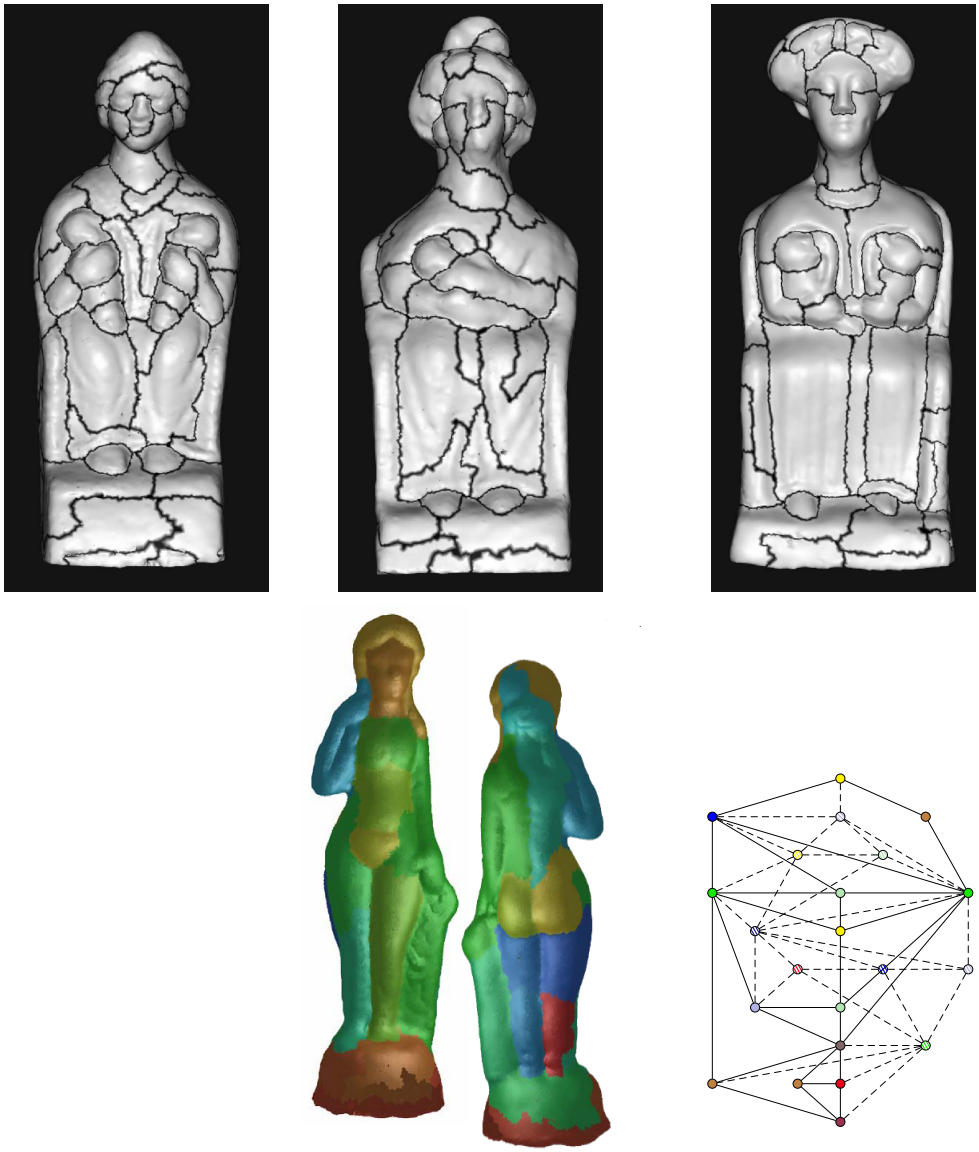


FIGURE 5.6 – Exemples d'objets 3D de Eros 3D suivis d'exemples sur la segmentation de surface basée sur la courbure et de graphe extrait pour un objet. En haut se trouvent 3 déesses mères segmentées, en bas une vénus suivie de son graphe d'adjacence

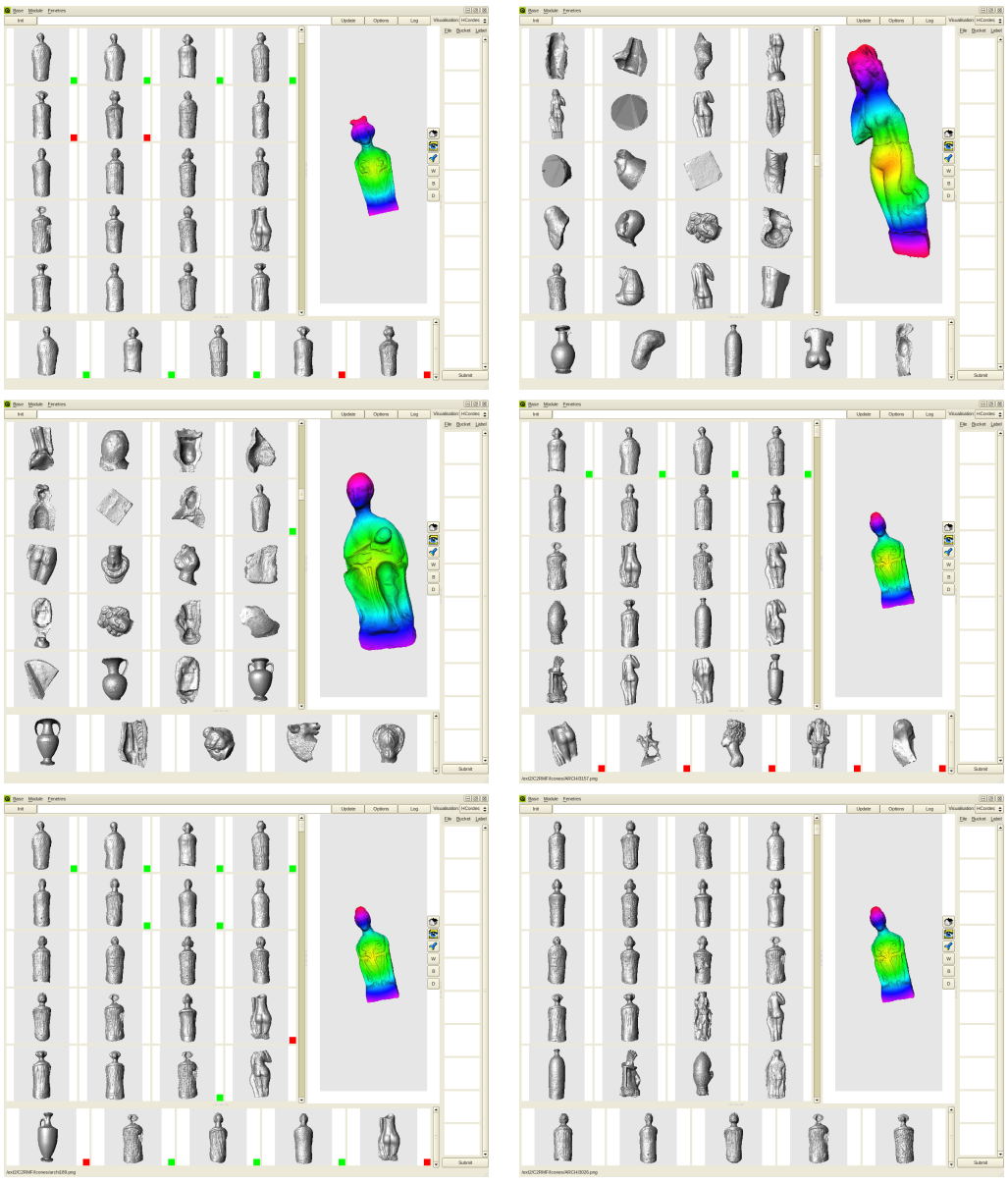


FIGURE 5.7 – Exemple d'une session de recherche effectuée sur des objets 3D de la base Eros3D. Le type d'objet recherché est celui des déesses mères. ReTIN renvoie en quatre itérations de la session de recherche plus d'une vingtaine de statuettes de déesses mères. Parmi les 20 premiers retours du dernier résultat seulement 3 objets ne sont pas dans la catégorie recherchée.

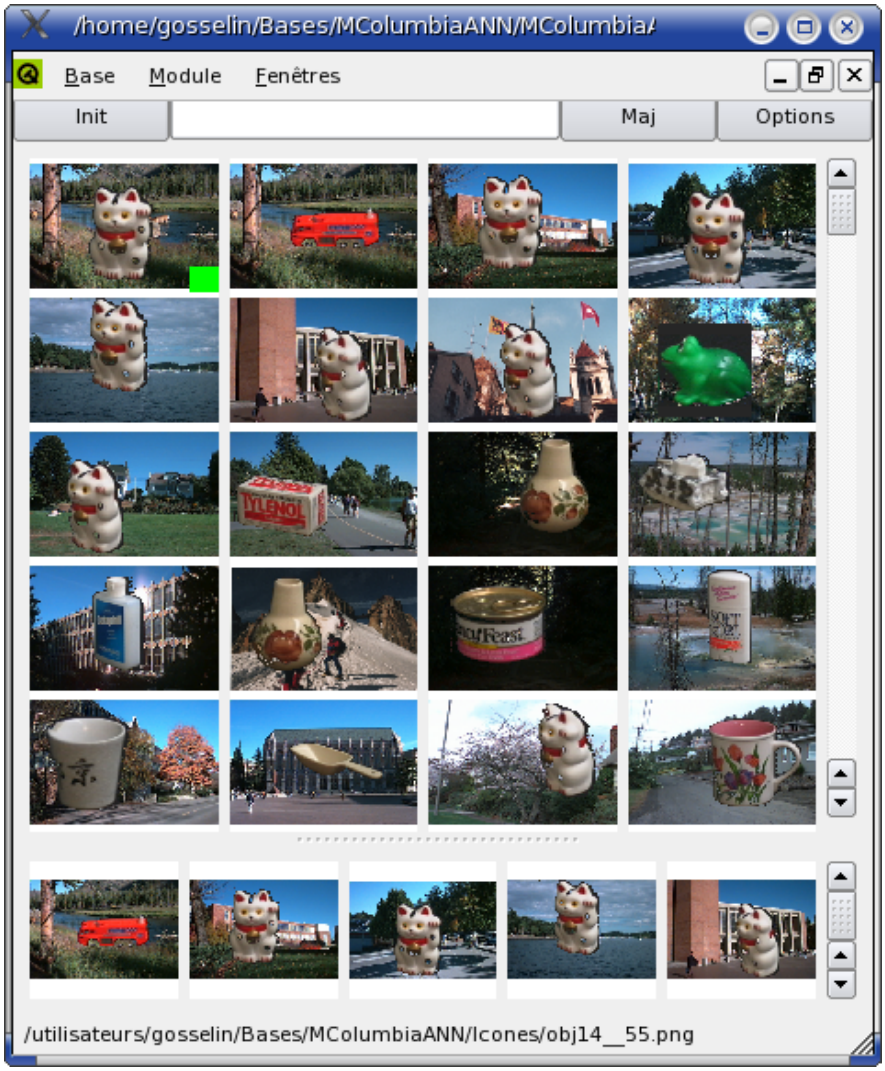


FIGURE 5.8 – Interface graphique du logiciel ReTIN. Le choix des éléments annotés positivement est indiqué par un carré vert. En bas 5 propositions d'éléments à labelliser issues de l'algorithme d'apprentissage actif. Cet algorithme propose des éléments qui accélèrent l'apprentissage. Les images son classées par ordre de pertinence décroissante de gauche à droite et de haut en bas.

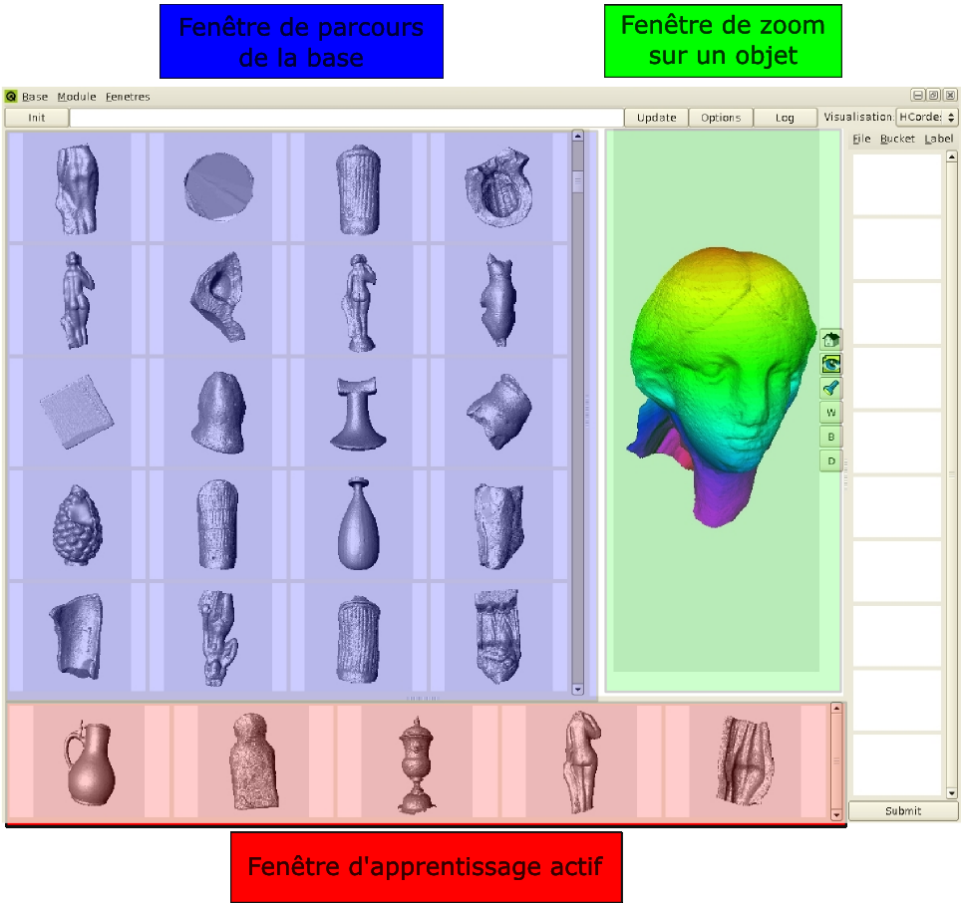


FIGURE 5.9 – L'interface ReTIN avec les objets 3D se découpe en 3 zones. La zone en bleu contient les résultats de requête ou l'affichage de la base(si aucun objet n'a été sélectionné). La zone rouge contient les propositions de l'algorithme dit de "recherche active". La zone verte est une vue détaillée de l'objet sélectionné, ici, avec un attribut de distance au centre de gravité selectionné.

une fois qu'il est satisfait du top des images renvoyé par ReTIN.

La figure 5.10 est un exemple de session de recherche effectuée sur la base COLUMBIA+ANN. . Dans cette session de recherche, l'objet à retrouver est une figurine en forme de chat. Les images les plus ressemblantes après la première requête se divisent en deux catégories : celles qui contiennent un fond ressemblant et celles qui contiennent l'objet. Peu, à peu au cours des sessions, ce sont les images qui contiennent l'objet qui sont ramenées en premier.

5.1.3 Méthode de comparaison : MAP

Différentes mesures ont été proposées pour évaluer des systèmes de recherche de données. Elles se séparent en deux catégories : les simples qui se basent directement sur les résultats des expérimentations et les complexes qui le plus souvent utilisent celles de la première catégorie pour se calculer.

Les mesures simples sont par exemple : la précision, le rappel, spécificité et "fall-out". Les mesures tournent autour de deux notions[77] :

- exhaustivité (ex :rappel)
- spécificité (ex : précision)

Parmi les mesures complexes, on trouve F-mesure et le Mean Average Precision. C'est cette dernière que nous avons choisie d'utiliser. Pour comprendre le MAP, tout d'abord revoyons les mesures de précisions et de rappel.

Précision La précision est la fraction des documents récupérés qui sont pertinents à la requête de l'utilisateur. $P = \frac{\text{documents pertinents récupérés}}{\text{documents récupérés}}$

Plus la requête retourne d'éléments corrects pour un même nombre de documents retournés plus la précision augmente. Dans le cadre de documents retournés classés on peut utiliser le P_N qui est une précision à l'ordre N.

$P_N = \frac{\text{documents pertinents parmi les N premiers récupérés}}{N}$

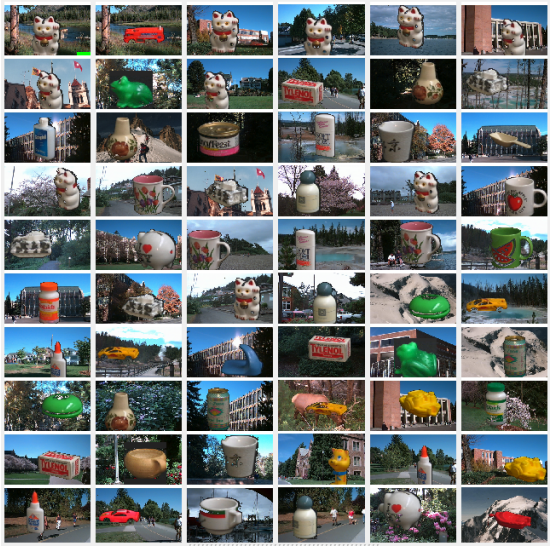
Rappel Le rappel est la fraction des documents pertinents à la requête qui sont récupérés avec succès.

$R = \frac{\text{documents pertinents récupérés}}{\text{documents pertinents}}$

Dans la classification binaire, le rappel est appelé sensibilité. Il peut donc être considéré comme la probabilité qu'un document pertinent soit retrouvé.

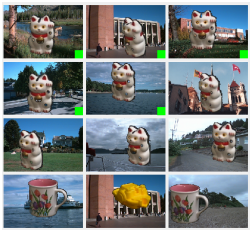
Il est trivial d'obtenir le rappel de 100% en retournant tous les documents en réponse à toute requête. Par conséquent le rappel ne suffit pas, mais il faut mesurer le nombre de documents non-pertinents aussi, par exemple en calculant la précision.

Les courbes affichant des points dont l'abscisse est la précision et l'ordonnée le rappel sont appelées courbes de rappel/précision. Un système dont la courbe de rappel/précision est au-dessus de celle d'un autre est considérée comme un meilleur système.



(a)

(b)



(c)

(d)

(e)

FIGURE 5.10 – Dans cette session de recherche, l'objet à retrouver est une figurine en forme de chat. Les images les plus ressemblantes après la première requête se divisent en deux catégories : celles qui contiennent un fond ressemblant et celles qui contiennent l'objet. Peu, à peu au cours des sessions, ce sont les images qui contiennent l'objet qui sont ramenées en premier.

Précision moyenne (MAP) La précision moyenne (AP) est calculée à différents niveaux de rappel (0%, 10%, 20%, ...,100%) :

$$AP = \frac{\sum_{\text{sur tous les rangs } rpertinents \text{ de la requête}} P_{\text{au rang r}}}{\text{documents retournés}} \tag{5.1}$$

Pour chaque niveau de rappel, les valeurs calculées sont moyennées sur l'ensemble des requêtes. La MAP est calculée comme suit :

$$MAP = \frac{\sum_{q \in Q} AP_q}{|Q|} \tag{5.2}$$

avec AP_q la précision moyenne pour une requête q et Q l'ensemble des requêtes.

On appelle parfois ces mesures, d'un point de vue géométrique, l'aire sous la courbe de rappel/précision(MAP).

5.2 Représentation des images et objets 3D

Pour des images et des objets 3D sous forme de graphe, il est nécessaire de les transformer en ensemble de sous-éléments reliés entre eux. Une fois ces sous-éléments extraits ainsi que leurs liens entre eux, ils constituent un graphe dont les sous-éléments sont les sommets et les liens sont les arêtes. On cherche ensuite à obtenir des descripteurs pertinents sur ces arêtes et sommets. Le graphe généré ainsi porte des descripteurs sur ses arêtes et sommets. A ce stade on peut utiliser ces graphes avec des méthodes à noyaux pour comparer deux images ou deux objets entre eux.

5.2.1 Segmentation

La phase d'extraction consiste à prélever de l'image des sous-ensembles de pixels. On applique l'extraction des descripteurs sur ces sous-ensembles. L'ensemble des descripteurs obtenus ainsi est utilisé dans la construction de fonction de similarité des images. L'image est donc représentée non plus par un descripteur global mais plusieurs descripteur locaux. Certaines techniques d'extraction prennent la quasi-totalité de l'image et dans ce cas on parle plutôt de segmentation de l'image, tandis que d'autres ne prennent que très peu de pixels de l'image. On cherche à extraire des sous-ensembles de l'image informatifs et à priori cohérents :

- points d'intérêt Moravec[78], Harris[79],[80]
- contours[81],
- régions

Dans notre cas, nous utilisons une segmentation en régions floues initialisée par watershed. Il existe diverses méthodes pour obtenir des régions. Les différentes voies selon Bezdek[82] qui mènent à une segmentation floue, sont le seuillage flou, la classification floue de pixels, des règles floues ou croissance de région. On peut aussi distinguer les méthodes par le type de régions sortie : régions nettes et régions floues. En principe, la segmentation en région est définie comme une partition en ensembles nets : chaque

pixel ne peut appartenir qu'à une seule région. Cette définition ne tient pas compte de ce fait que certains pixels du centre d'une région homogène appartiennent certainement plus à celle-ci que des pixels à l'extrémité, obtenus lors d'une phase de croissance de région. C'est pourquoi la segmentation floue propose de construire des régions qui ne sont plus des ensembles nets comme dans la segmentation classique mais des ensembles flous.

L'algorithme proposé par Philipp-Foliguet&al[83] se veut aussi général que possible. Il n'est pas dédié à un type d'image particulier, ne nécessite pas de réglage de paramètre pour chaque image ce qui permet de segmenter des bases entières sans avoir à adapter des paramètres. Cet algorithme est basé sur la croissance de régions. La croissance est une augmentation des pixels constituant les régions à partir de germes de régions. L'idée de l'algorithme est de lier l'appartenance à la région avec sa distance au germe de la région. Un germe est constitué d'un ensemble de forte homogénéité colorimétrique. L'ensemble de régions floues obtenues ainsi est modélisé par des ensembles flous, possédant les propriétés suivantes :

- chaque ensemble est homogène en couleur
- l'expansion des régions est limitée par des normes du gradient élevées.
- présence de l'incertitude (ou le degré de flou) quand au moins deux régions se rejoignent

L'algorithme mixe les approches contour et région de la segmentation. La croissance de région impose des régions homogènes et l'utilisation de la norme du gradient de l'image pour effectuer le calcul contraint les régions par les bords. Autrement dit l'algorithme fait croître les régions en rajoutant des pixels homogènes et les zones de l'image possédant une norme de gradient élevé vont former un barrage à la croissance de ces régions. On peut cependant les contourner pour s'affranchir du bruit impulsionnel. Afin d'automatiser l'initialisation de l'algorithme, le choix des germes correspondent aux minimas locaux de la norme du gradient. Les degrés d'appartenance sont alors calculés au moyen "d'une distance topographique", définie par Philipp-Foliguet&al [83]. Cette distance dépend du chemin le plus court tracé sur la surface constituée par la norme du gradient dans un espace 3D.

Pour déterminer nos germes, on s'appuie sur un algorithme de segmentation de type de ligne de partage des eaux(watershed)[84] . L'algorithme utilisé est composé de trois étapes. La première exécute l'algorithme de calcul de la ligne de partage des eaux et la construction des bassins versants. La deuxième fusionne les bassins versants obtenus qui sont souvent en trop grand nombre. La troisième effectue la "fuzzication" en calculant le degré d'appartenance de chaque pixel aux régions floues. Les deux premières étapes peuvent être inversées comme il a été proposé dans l'article de Philipp-Foliguet&al [85] .

Le choix du calcul de la norme du gradient s'est porté sur la méthode de Di Zenzo.

Un algorithme de ligne de partage des eaux s'inspire du fonctionnement de celle définie en géographie : c'est la ligne où l'écoulement de l'eau est différent pour chaque coté de celle-ci. Les zones longeant cette limite s'appellent des bassins versants. Si on simule l'inondation de ces bassins, on obtient comme séparation de ceux-ci une ligne de partage des eaux les séparant. On obtient ainsi une segmentation de cet espace

géographique. L'algorithme de ligne de partage des eaux reprend cette idée pour les images. A la place d'utiliser la hauteur, on utilise d'autre fonctions applicables aux images : dans notre cas la norme du gradient colorimétrique.

Le calcul des degrés d'appartenance aux régions s'est effectué pour être compatible avec le stockage d'un octet. Au lieu d'être stockés en nombre réel entre 0 et 1, ils sont stockés en tant qu'entier compris entre 0 et 255. Les appartenances des pixels germes sont initialisés à 255 et les autres à 0. Les pixels des germes sont stockés dans la file d'attente Q_B de leur bassin B

Pour tout bassin B **faire**
  Pour tout pixel $s \in B$ **faire**
    Si $(s \in germe(B))$ **Alors**
      $\mu_B(s) = 255$
      Ajout de s dans la file Q_B
    Sinon
      $\mu_B(s) = 0$
    Fin Si
  Fin Pour
Fin Pour

Algorithme 12: Extrait de l'article [83]

Les bassins sont fusionnés en fonction de leur aire, de leur profondeur ou de leur volume. Les bassins dont le volume, l'aire ou la profondeur est de taille insuffisante est absorbé par son bassin voisin. On dénote le bassin absorbant du bassin B par $Absord(B)$ et la norme du gradient d'un pixel s par $g(s)$. La différence entre les niveaux des deux fonds de bassins implique une pénalité sur les degrés d'appartenance du bassin absorbé

B.

Pour tout pixel $s \in germe(B)$ **faire**
  $h_B = g(s)$
Fin Pour
Tant que (il existe un bassin A, tel que $A = Absord(B)$) **faire**
  Pour tout pixel $c \in germe(A)$ **faire**
    $h_A = g(c)$
    Pour tout pixel $s \in germe(B)$ **faire**
      $\mu_B(s) = 255 - |h_B - h_A|$
    Fin Pour
    $B = A$
  Fin Pour
Fait

Algorithme 13: Extrait de l'article [83]

Chaque ensemble de bassins versants fusionnés donne une région floue. Le degré d'appartenance d'un pixel est inversement proportionnel à la distance topographique. Celle-ci est définie comme la longueur du chemin le plus court connectant le pixel avec le germe le plus proche sur la surface topographique obtenue par la norme du gradient. Les pixels appartenant à des germes ont un degré d'appartenance maximal s'ils ne sont pas issus de bassin absorbé. Plus les valeurs diminuent, plus les pixels s'éloignent des pixels germes de la région jusqu'à atteindre la valeur nulle. Chaque pas d'écartement spatial coûte la suppression d'une valeur proportionnelle à différence entre les normes de gradients(du pixel considéré et du pixel germe) et la suppression de 1. Le paramètre k sert à pondérer la distance spatiale avec le germe de région le plus proche et la valeur de la différence entre les normes de gradients. Il influera sur la diffusion des régions : entre 0.5 pour des régions larges et 3 pour des régions compactes. Il est mis en général

à 2, notamment pour la segmentation entière de base d'images.

Pour tout bassin versant B de R **faire**
  Tant que (la file Q_B est non vide) **faire**
    extraire s de Q_B
    Pour tout pixel v voisin de s **faire**
      $\mu = \mu_B(s) - (k.|g(v) - g(s)| + 1$
      Si $(\mu > \mu_B(v))$ **Alors**
        $\mu_B(v) = \mu$
      Fin Si
      mettre v dans Q_B
    Fin Pour
  Fait

Fin Pour

Algorithme 14: Extrait de l'article [83]

méthode utilisée La technique utilisée pour le calcul de la ligne partage des eaux est un algorithme basé sur le principe de la goutte d'eau. On utilise un graphe pondéré sur les arêtes et des coupes watershed-cut .

La génération des régions floues se fait à partir des régions nettes. Ces régions servent à obtenir une base de la région floue qui va s'agrandir progressivement suivant un gradient de couleur(grossissement de région).

Cette technique de segmentation a été appliquée à des objets 3D[86] en s'appuyant sur les courbures de l'objet pour construire la ligne de partage des eaux et les bassins. Dans le cadre des objets 3D nous n'avons pas de régions floues.

5.2.2 Graphes issus d'image ou d'objet

Les régions issues de la segmentation deviennent dans le graphe les sommets de celui-ci et les relations d'adjacence entre elles forment les arêtes.

5.2.2.1 Image

Les graphes issus de la segmentation et des relations d'adjacence de l'image sont dans notre cas planaires. Cette propriété implique une réduction de la complexité de calcul des noyaux sur graphes puisqu'elle implique une réduction du nombre de chemins possibles. La figure 5.12 montre le passage d'une image de la base de données Bird dans les différentes étapes qui produisent le graphe. On remarque que l'oiseau de la figure est divisé en 6 régions. Le sous-graphe de l'objet dans l'image est en général un petit

FIGURE 5.11 – Objet 3D segmentés : déesses mères.

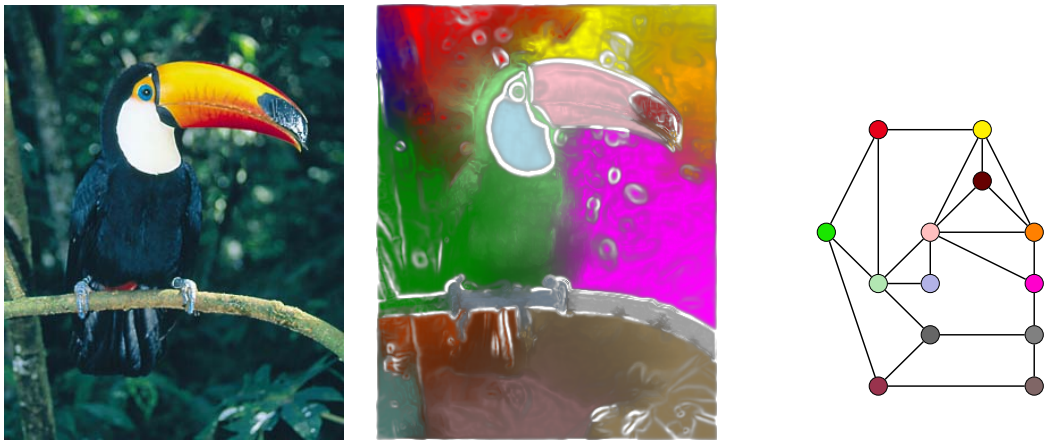


FIGURE 5.12 – Passage d'un toucan à un graphe d'adjacence de régions floues.

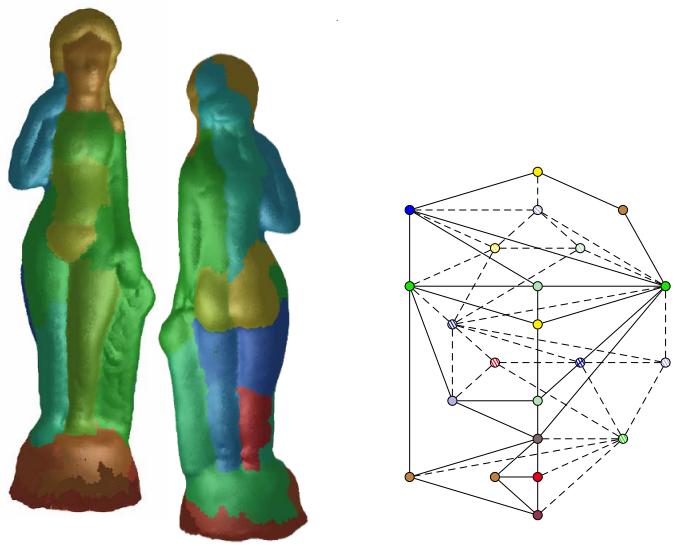


FIGURE 5.13 – Passage d'une statuette de Venus à un graphe d'adjacence

graphe de 6 sommets en moyenne. Nos réglages permettant un choix de la taille des régions concernées, nous avons opté pour une taille qui donne entre 12 et 35 régions dans chaque base. On remarque néanmoins que plus la base est générique plus la taille de sous-graphe varie pour un même objet. Ceci est relié au fait que l'objet n'est pas forcement toujours le sujet central de l'image. Par exemple les chats dans VOC peuvent soit couvrir toute l'image, soit n'être qu'un élément au fond de la scène de l'image. La recherche d'objet dans ce cadre est l'équivalent d'une recherche de sous-graphe.

5.2.2.2 Objet 3D

Les graphes issus de la segmentation en régions dans le cadre d'objets 3D ne sont pas forcément planaires. La figure 5.13 montre le passage d'une image de la base de données Eros3d (une Venus) au graphe. Le graphe de la figure 5.13 en est un exemple il possède 54 arêtes, 22 sommets et 26 faces et par conséquent ne satisfait pas la formule d'Euler des graphe planaires. Néanmoins ils en sont proches : Si on prend un morceau de surface qui ne boucle pas(ne produit pas un tube, ou une "boule"), le sous-graphe correspondant est planaire.

La particularité principale de ces graphes est qu'ils représentent les objets entièrement. Dans ce cadre, la recherche d'objet est équivalente à une recherche du graphe le plus proche soit un appariement inexact de graphe.

5.2.3 Descripteurs des sommets et arêtes

5.2.3.1 sommets

Image Une large gamme de descripteurs existent (couleur, gradient...) pour les régions de pixels. Dans notre cas, nous utilisons un histogramme de couleurs et de gradient (taille 136) sur les régions de pixels. Les couleurs sont représentées dans l'espace L_{ab}. Le gradient est sur L et est calculé à trois résolutions différentes obtenues par filtre moyenné pour chaque pixel de la région. Il est recalé ensuite par rapport à la direction du gradient global de la région en séparant 8 directions. Toutes les valeurs sont pondérées par le degré d'appartenance à la région.

objets 3D Les descripteurs utilisés pour les objets 3D dans le cadre de nos expériences sont les CEGI Complex Extended Gaussian Images.

Le complexe EGI (CEGI) est une variante de l'EGI qui permet de résoudre le problème de la discrimination entre les concavités et les convexités. La fonction décrit l'objet grâce à deux attributs : l'orientation de la face et la distance entre la face et le centre de gravité de l'objet. Pour chaque facette de la sphère de Gauss, l'accumulation de ces deux attributs est réalisée dans l'espace complexe.

$$Pnk = \sum_{l=1}^{N_k} A_{l,n_k} e^{jd_{l,k}} \tag{5.3}$$

où $d_{l,k}$ est la distance entre le centre de la face et le centre objet, N_k est le nombre surface de l'objet dans une direction n_k et $A_{l,nk}$ est la surface de la face pour l'orientation n_k.

Pour augmenter la différence entre les concavités et les convexités, cette distance est signée. Elle est négative lorsque la face est dirigée vers le centre de l'objet et positive dans l'autre direction. Enfin le module et la phase sont calculés et composent le descripteur CEGI.

5.2.3.2 arêtes

Nos graphes représentent des graphes topologiques de régions. Nous avons donc étudié les descripteurs de relations spatiales relatives. Ils tentent de décrire les relations spatiales entre deux régions. Une première description est celle des relations temporelles d'Allen[87] adaptées aux relations spatiales : A est gauche de B, C est en-dessous de D etc... Cependant elles posent un problème de calcul pour des régions aux formes non régulières (région entourant une autre région etc...). De plus, elles sont vraies ou fausses mais non valuées.

Dans l'article de [88], la description des relations spatiales se fait en considérant les régions comme deux objets de la physique qui exercent mutuellement des forces entre eux.

Pour notre part, nous nous sommes intéressés aux frontières des régions. Dans notre cas, le descripteur agit comme un compteur des positions relatives entre les pixels ou les voxels des deux régions R_1 et R_2. En supposant que les régions soient non floues, le

compteur va comptabiliser pour la région R_1 tous les pixels à la frontière de R_1 et R_2 qui sont au-dessus, en-dessous, à gauche et à droite dans un vecteur à quatre dimensions. L'arête orientée entre deux régions est donc décrite par 4 attributs : A(above), B(below), L(left) and R(right). Pour deux régions adjacentes R_i et R_j, nous considérons l'ensemble F_{ij} de couples de pixels $(p_i, p_j) \in R_i \times R_j$ voisins par une connectivité de 4. Nous définissons les propriétés suivantes :

$$T_{ij}^{\mathcal{R}_t} = \frac{|\{(p_i, p_j) \in F_{ij}\ ,\ p_j \mathcal{R}_t p_i\}|}{|F_{ij}|}$$

avec les relations spatiales suivantes :

$$p_i \mathcal{R}_1 p_j \Leftrightarrow p_i \text{ est au dessus } p_j$$

$$p_i \mathcal{R}_2 p_j \Leftrightarrow p_i \text{ est en dessous } p_j$$

$$p_i \mathcal{R}_3 p_j \Leftrightarrow p_i \text{ est à gauche } p_j$$

$$p_i \mathcal{R}_4 p_j \Leftrightarrow p_i \text{ est à droite } p_j$$

Ces quatre attributs sont assemblés dans le vecteur suivant : $(T_{ij}^{\mathcal{R}_1} T_{ij}^{\mathcal{R}_2} T_{ij}^{\mathcal{R}_3} T_{ij}^{\mathcal{R}_4})$ pour l'arête des des deux sommets v_i et v_j.

Pour voir la pertinence de ce descripteur nous avons regardé dans un premier temps s'il existait dans une base d'images d'objets simples comme COLUMBIA, des répartitions différentes pour les valeurs de ces descripteurs en fonction des catégories.

Nos régions étant surfaciques dans nos objets 3D, le même type de descripteur est utilisable. Néanmoins, il nécessite une adaptation sur les relations horizontales gauche et droite. L'idée est que si on regarde les deux régions, visuellement, l'une sera à gauche et l'autre à droite. En réalité on considère que gauche est une rotation dans le sens trigonométrique par rapport à un couple de vecteur (u, v) avec u un vecteur normal à la surface et v un vecteur orthogonal. La droite est la rotation inverse au sens trigonométrique par rapport à ce même couple de vecteur.

De plus il est possible d'utiliser des variantes de ce descripteur en fusionnant les valeurs horizontales entre elles et les valeurs verticales entre elles.

En s'inspirant des travaux effectués par Hudelot et Bloch[89] sur les relations spatiales floues, on peut envisager une évolution de ces attributs pour être calculés et adaptés à des régions floues. De plus on peut prévoir de les modifier afin d'obtenir une robustesse aux phénomènes de rotation plus présents dans les objets 3D.

5.3 Résultats comparatifs

Dans un noyau sur graphe basé sur des ensembles de chemins, il existe plusieurs noyaux pouvant inférer sur les résultats : les noyaux mineurs (arêtes et sommets), les noyaux sur chemins et enfin les noyaux sur les ensembles de chemins. Nous avons décidé d'effectuer des expériences sur ces trois "niveaux" de noyaux pour évaluer nos noyaux sur graphes.

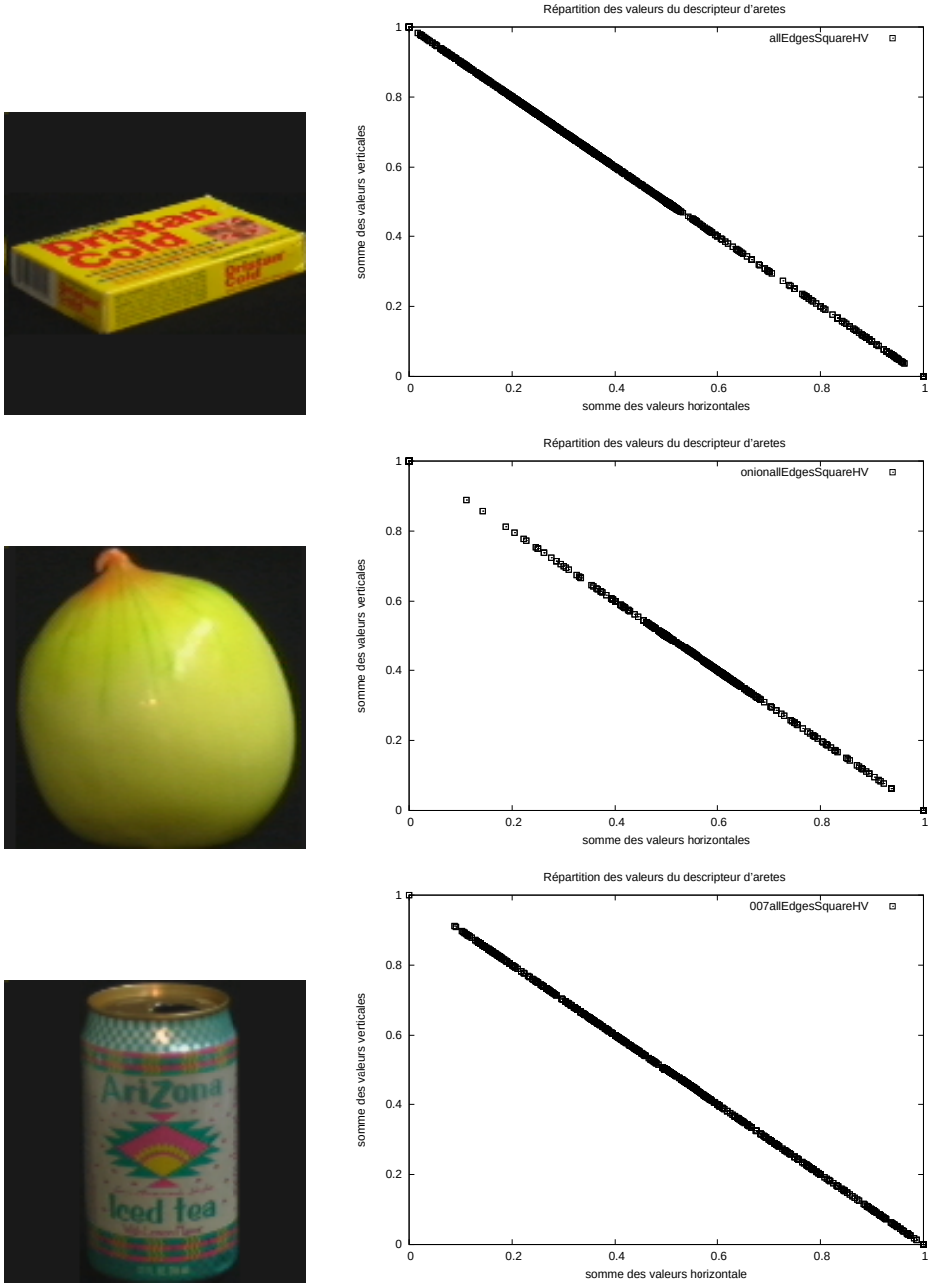


FIGURE 5.14 – Répartition des valeurs horizontales et verticales pour différentes catégories d'objets de la base COLUMBIA

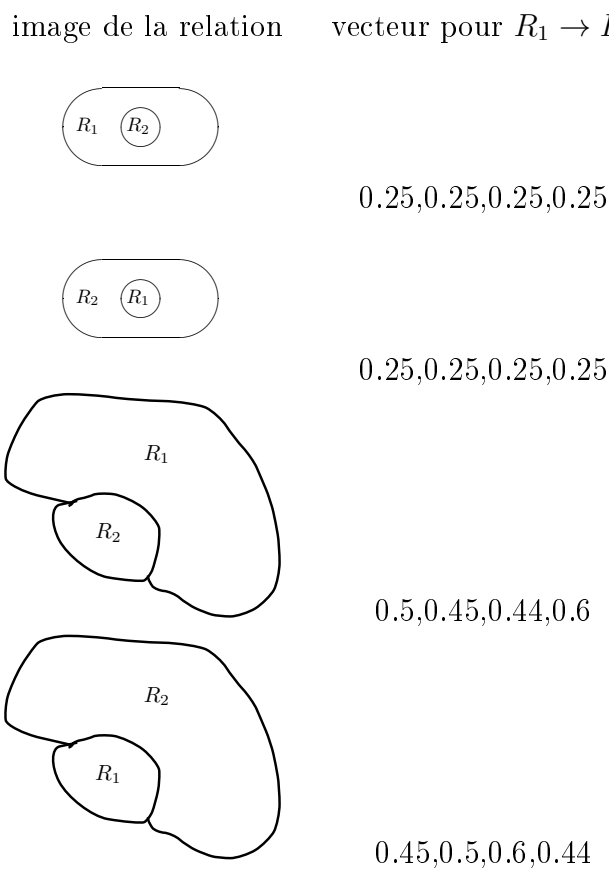


FIGURE 5.15 – Cette figure présente quelques valeurs du vecteur du descripteur, pour quelques positions relatives entre deux régions R_1 et R_2.

Ceci nous a permis de ne pas effectuer trop d'expériences avec un paramétrage aveugle et de voir les effets des choix et des combinaisons entre les noyaux.

Notre problématique est d'une part l'intérêt des noyaux sur graphe, dont celui développé lors de la thèse mais aussi d'entamer une analyse des mécanismes mis en jeu dans les noyaux sur graphes pour la reconnaissance d'image ou d'objet.

Ainsi les expériences sur noyaux mineurs servent à vérifier les caractères discriminants des noyaux utilisés et la pertinence de ceux-ci sur les descripteurs utilisés.

Les expériences sur chemins permettent d'étudier le comportement des noyaux en fonction des longueurs de chemin.

Les expériences menées sur les noyaux sur graphes cherchent à montrer leur intérêt dans la recherche interactive d'objets.

5.3.1 Noyaux mineurs : sommets et arêtes

Pour les noyaux dit mineurs, nous avons utilisé les travaux effectués sur les noyaux de sommets dans la littérature des sacs de régions et effectué nos propres expériences notamment sur les noyaux sur arêtes.

sommets Les études basées sur les sacs de noyaux de régions sont un moyen d'évaluer les noyaux mineurs k_v sur régions[73]. Ces travaux indiquent un meilleur comportement dans les expériences d'apprentissage par SVM des noyaux gaussiens avec une distance du χ^2.

arêtes Nous avons étudié deux points : la pertinence de nos arêtes dans nos noyaux (figure 5.16) et le meilleur noyau applicable aux descripteurs sur arêtes. Les expériences ont été effectuées sur la base COLUMBIA(cf section 5.1.1.1). Le noyau sur graphe mis en œuvre pour la première expérience est celui de la somme associé au noyau sur chemins sommé lui aussi. Pour obtenir le noyau ne prenant en compte que le sommet(resp. l'arête), le noyau sur arêtes(resp. sommets) retourne une valeur neutre quelques soient les éléments appariés : $k_e(e,e') = 1$(resp $k_e(e,e') = 1$). Ainsi on obtient $k_e(e,e')k_v(v,v') = k_v(v,v')$ resp $(k_e(e,e')k_v(v,v') = k_e(,e')$) dans notre noyau sur chemin : $K_C = k(v_0,v_0') + \sum k_e(e,e')k_v(v,v')$.

Pour les noyaux mineurs sur arêtes nous avons choisi de prendre un noyau sur sacs d'arêtes. Celui-ci somme les noyaux mineurs d'arêtes :

$$K(G(V,E),G'(V',E')) = \sum_{e\in E}\sum_{e'\in E'} k_e(e,e')$$

.

La figure 5.2.3.2 montre des répartitions pour quelques catégories d'objet de la base COLUMBIA différentes en séparant 3 cas : un noyau sur graphe n'utilisant que les arêtes(barre en vert foncé), un noyau sur graphe n'utilisant que les sommets(barre en vert clair) et un noyau sur graphe utilisant arêtes et sommets(barre en jaune). Les catégories sont dans l'ordre(1,2,3,4) un oignon, une boîte jaune, une tasse avec un motif particulier dessus et un bateau en plastique. Ces résultats montrent qu'il existe un gain

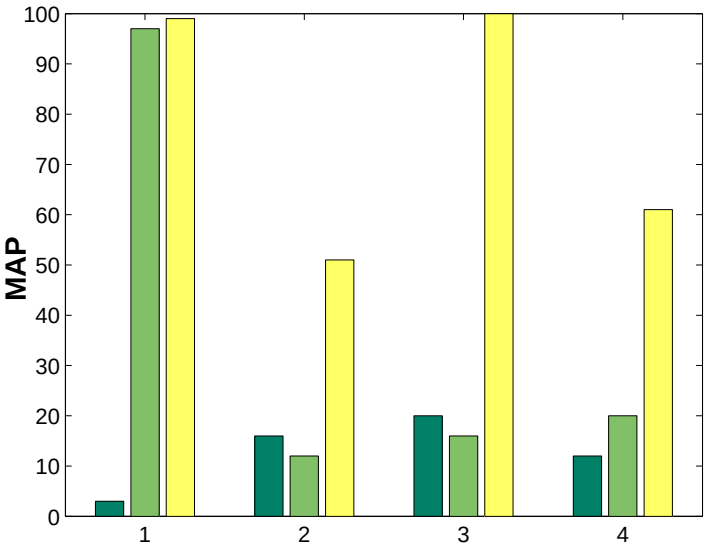


FIGURE 5.16 – Études des MAPs pour quatre catégories de COLUMBIA : vert foncé pour, un noyau sur graphes uniquement avec des descripteurs sur les arêtes, vert clair, avec des descripteurs sur les régions et en jaune avec des descripteurs sur les arêtes et les sommets. Les catégories sont dans l'ordre(1,2,3,4) un oignon, une boîte jaune, une tasse avec un motif particulier dessus et un bateau en plastique.

en utilisant les arêtes et que pour certaines catégories d'objets(la tasse avec un motif) on peut les distinguer en n'utilisant que l'information sur arêtes.

Les expériences sur différents k_e n'ont pu mettre en évidence de noyau donnant de meilleurs résultats que les autres. Pour les expériences effectuées par la suite nous avons choisi un noyau gaussien avec une distance L_1. Néanmoins ces expérimentations montrent que l'utilisation uniquement de petits descripteurs sur les arêtes permettent dans cette base une recherche d'objet. En regardant les résultats par catégorie, on a pu observer de plus des MAP de 80% dès 30 labels pour certaines d'entre elles(les tasses avec motifs par exemple). Pour rappel, les descripteurs sur les régions sont des vecteurs de taille 128(contre 4 pour ceux des arêtes). Ces expériences à défaut de déterminer le meilleur noyau pour les arêtes, montrent l'intérêt de nos descripteurs.

5.3.2 Études sur la longueur des chemins

Un premier examen des noyaux sur chemins nous permet de distinguer, dans un premier temps, trois types de comportement du noyau en fonction de l'augmentation de la longueur des chemins : comportement croissant, comportement décroissant ou comportement indéterminé. Pour illustrer les trois cas nous allons prendre 3 noyaux sur

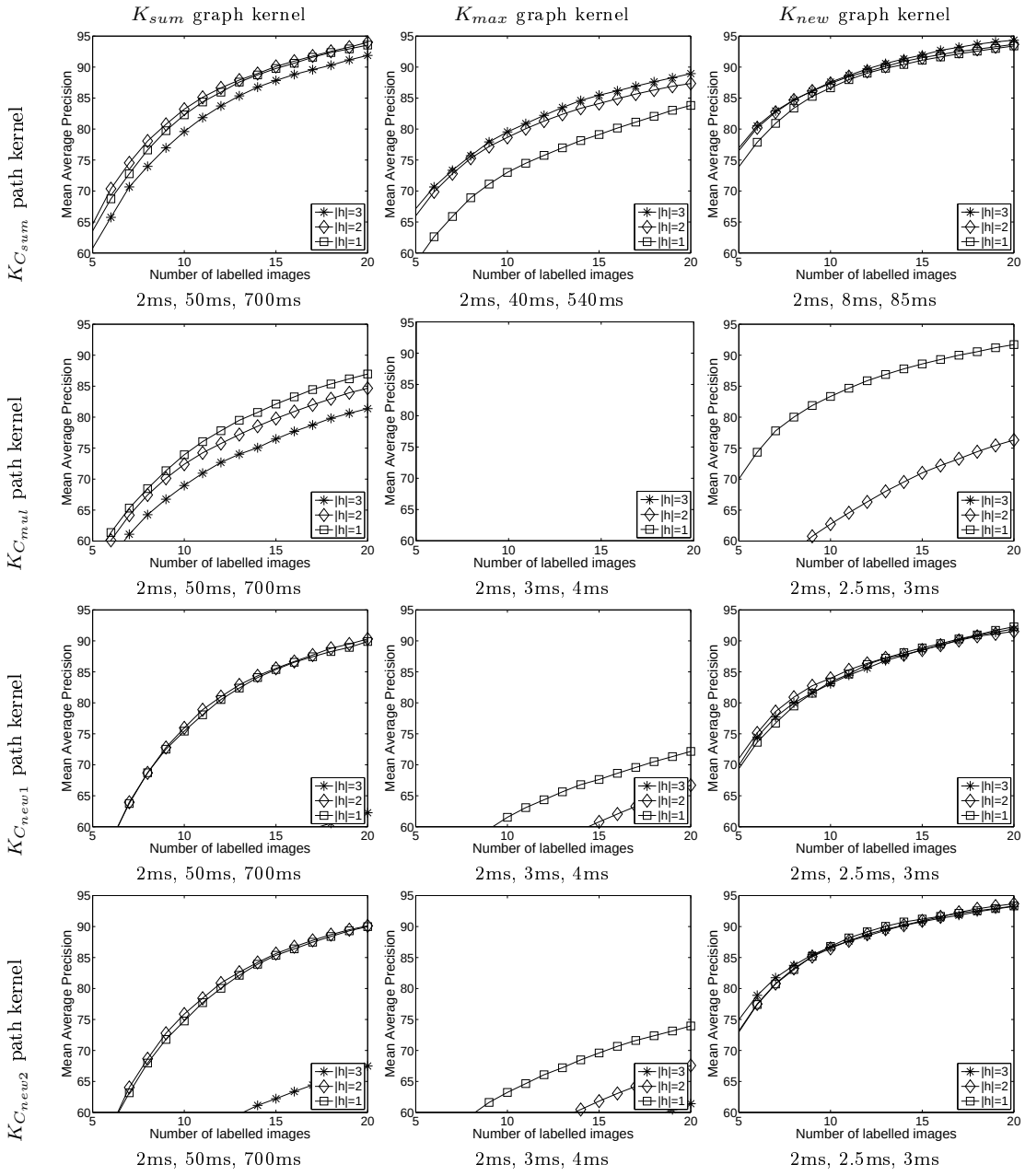


FIGURE 5.17 – Cette figure récapitule un ensembles d'expériences. Les résultats sont organisés de façon à avoir le même noyau principal(K) pour toutes les expériences d'une même colonne et le même noyau sur chemin (K_C) pour les expériences étant sur la même ligne.

chemin différents :

$$K_{C_{sum}}(h,h') = k_v(v_0,v'_0) + \sum_{i\leq|h|} k_e(e_i,e'_i)k_v(v_i,v_i) \tag{5.4}$$

$$K_{C_{mul}}(h,h') = k_v(v_0,v'_0) \times \prod_{i\leq|h|} k_e(e_i,e'_i)k_v(v_i,v_i) \tag{5.5}$$

$$K_{C_{new1}}(h,h') = k_v(v_0,v'_0) \times \prod_{i\leq|h|} (1+k_e(e_i,e'_i))k_v(v_i,v_i) \tag{5.6}$$

Autrement dit soit un couple de chemins donné (h,h') tel que : $K_C(h,h') = K_{C_d}(h,h') = K_{C_i}(h,h') = \alpha$. Pour tout couple de chemins h_2 et h'_2 débutant respectivement par h et h' différents de ceux-ci on a :

- $K_{C_{sum}}(h_2,h'_2) = K_{C_{sum}}(h,h') + k_e(e_{i+1},e'_{i+1})k_v(v_{i+1},v'_{i+1}) \geq \alpha$
- $K_{C_{mul}}(h_2,h'_2) \leq \alpha$
- $K_{C_{new1}}(h_2,h'_2) \leq \alpha$ ou $K_{C_i}(h_2,h'_2) \geq \alpha$

Par exemple, les noyaux sur chemin à base additive(Eq.5.4) sont croissants, tandis que ceux à base multiplicative(Eq.5.5) pour des noyaux mineurs inférieurs à 1 sont décroissants. Dans le premier cas on favorise les chemins de grande longueur et dans le second ceux de petite longueur. Le dernier cas(Eq.5.6) ne favorise ni l'un, ni l'autre.

Pour tester le comportement des noyaux en fonction de la longueur du chemin, on se place dans la base Bird. On fixe les noyaux mineurs sur arêtes et sommets par un noyau gaussien avec une distance du χ^2 pour respecter la condition précédente. Pour les trois noyaux présentés, on effectue des recherches d'objet avec ReTIN. Les paramètres variants sont les noyaux majeurs(K_{max},K_{sum} et K_{new} voir sous-section 5.3.4.1) et les longueur de chemins suivantes : 1,2,3 (arêtes). Les résultats sont sur les figures 5.18,5.19,5.20.

Le noyau 5.4($K_{C_{sum}}$) a des courbes de MAP supérieures en fonction de la taille des chemins sur les deux noyaux majeurs (K_{max} et K_{new}). Le noyau K_{sum} présente une courbe MAP plus basse pour la longueur de chemin la plus haute.

Le noyau 5.5($K_{C_{mul}}$) a des courbes de MAP inférieures en fonction de la taille des chemins sur les deux noyaux majeurs (K_{sum} et K_{new}).

Le noyau 5.6($K_{C_{new1}}$ sur la figure) a des courbes de MAP stables en fonction de la taille des chemins sur les deux noyaux majeurs (K_{sum} et K_{new}).

Pour la majeure partie des résultats, les chemins composés d'une arête donnent de bons résultats.

Le noyau 5.4 est croissant, les valeurs des similarités entre chemins sont croissantes en fonction de la longueur du chemin et à priori de plus en plus discriminante. Ceci explique les résultats pour les deux noyaux K_{max} et K_{new}. Mais plus on est discriminant, plus le risque est que tous les chemins soient à égale distance les uns des autres et par conséquent que les résultats deviennent mauvais à partir d'une certaine longueur(impossibilité de faire des sous-groupes de plus d'un chemin).

Le noyau 5.5 est décroissant, les valeurs des similarités entre chemins sont décroissantes en fonction de la longueur du chemin et à priori de plus en plus semblables puisque tendant vers 0. Par conséquent, plus on travaille avec des chemins de grande longueur moins on les discerne entre eux. Il est plus difficile pour le système de les classifier d'où les courbes MAP les plus basses pour les longueurs les plus hautes pour ce noyau.

Le noyau 5.6 n'est ni croissant, ni décroissant. Quelle que soit la longueur de chemin utilisée, les chemins ne deviennent ni semblables, ni avec la même différence entre eux.

En conclusion, l'impact de la longueur du chemin varie selon le noyau sur chemin utilisé ainsi que le noyau sur graphe utilisé. Prendre des chemins plus longs a un impact négatif si on prend une fonction noyau sur chemin décroissante. A l'inverse l'impact est positif si la fonction noyau sur chemin a un comportement croissant. Les chemins d'une arête permettent déjà d'obtenir de bons résultats sans avoir forcément un gain extrêmement notable en prenant des chemins plus longs.

5.3.3 Comparaison de noyaux sur chemins

Il est nécessaire dans le cadre de graphes sur noyaux construits à partir de sous-éléments structurés de regarder le comportement des noyaux combinés pour construire le graphe. Pour les noyaux sur chemins, nous nous sommes intéressés d'une part au motif qu'ils peuvent représenter dans les graphes et d'autre part à leur stabilité dans les différents noyaux sur sacs de chemins. Les noyaux sur chemins ne sont pas efficaces sur tout type de noyau sur graphes .Nous avons par conséquent étudié sur divers types de noyau de graphe le comportement de 3 noyaux sur chemins présentés à la section précédente avec un supplémentaire :

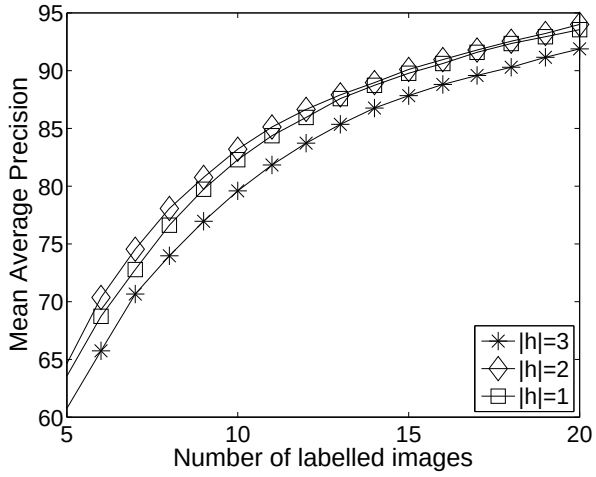
$$K_{C_i}(h,h') = k_v(v_0,v'_0) \times \prod_{i\leq|h|}(1+k_e(e_i,e'_i))k_v(v_i,v_i) \tag{5.7}$$

Ce noyau($K_{C_{new2}}$ sur la figure5.21) est à comportement non monotone. Les résultats sont répartis sur quatre figures(fig.5.18,5.19,5.20,5.21) chacune centrée sur un noyau de chemin.

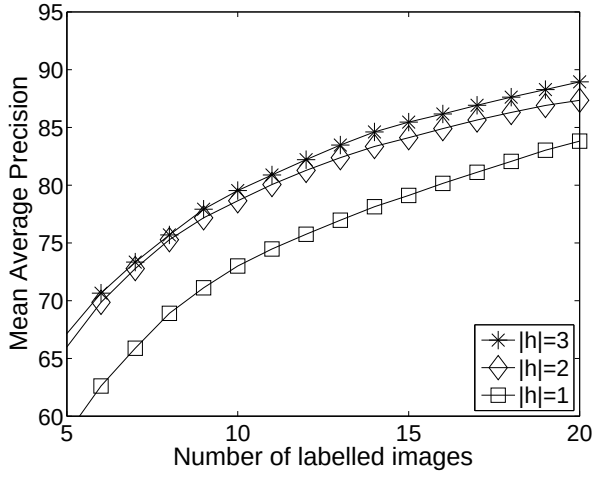
Un des premiers résultats constatés est une différence de comportement pour ces noyaux en fonction du noyau sur graphe utilisé. Par exemple le noyau (5.5) donne des MAP plus bas sur le noyau K_{max} que les deux autres. Notons que le noyau (5.4) donne de bons MAP quelque soit le noyau.

On peut considérer que les noyaux (5.4) et (5.6) sont globalement plus stables sur ces 3 types de noyaux.

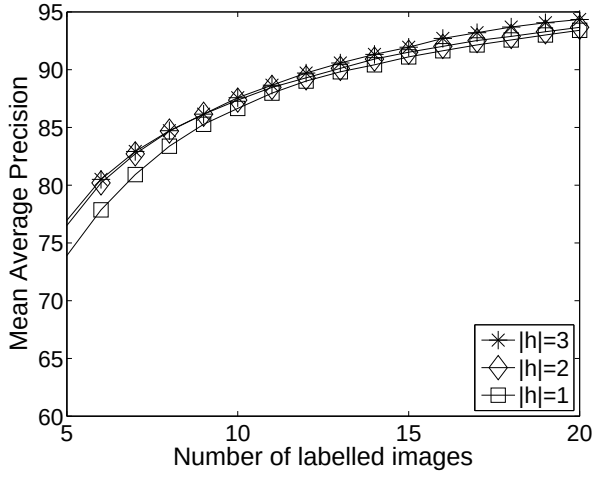
On peut expliquer la faiblesse du noyau (5.5) avec le noyau max pour les chemins les plus longs. La propriété intrinsèque de ce noyau (5.5) est de donner des valeurs de similarité de chemins de plus en plus identiques. Ainsi les chemins pris en compte deviennent de plus ressemblants et tendent à avoir la même distance entre eux.



K_{sum} graph kernel

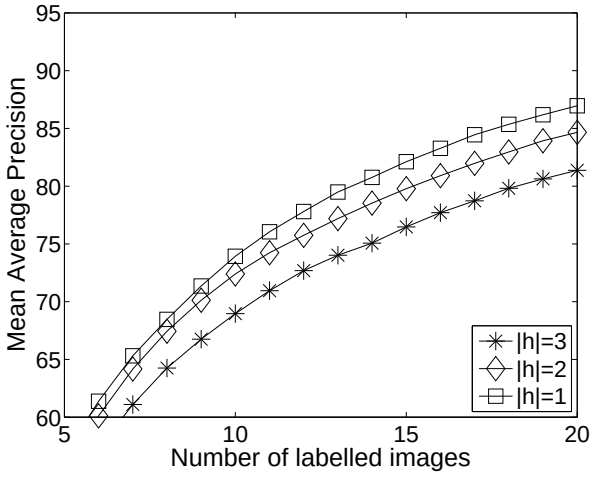


K_{max} graph kernel

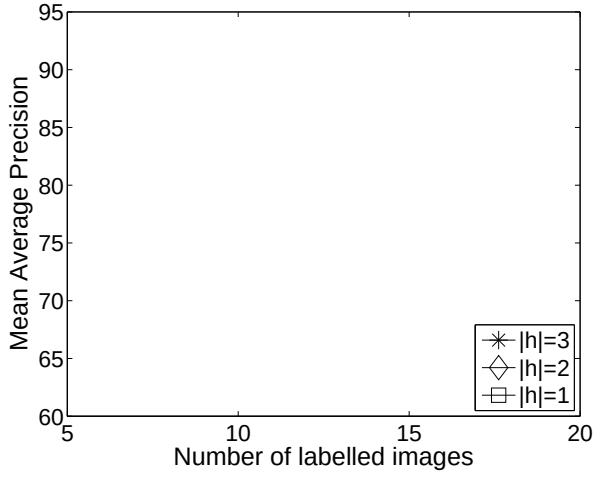


K_{new} graph kernel

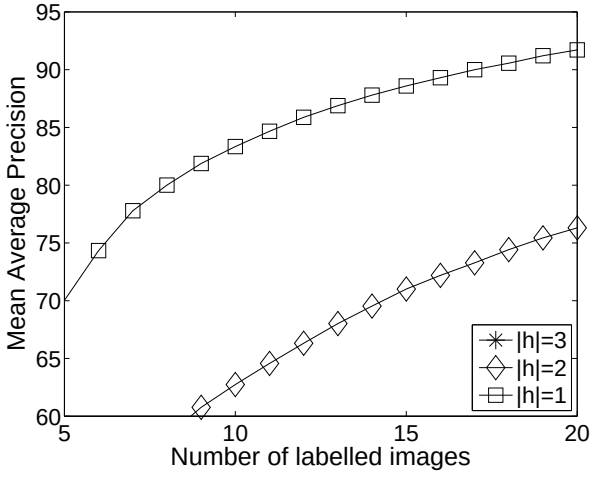
FIGURE 5.18 – Les résultats présentent les expériences pour 3 noyaux sur graphe différents pour un même noyau sur chemin ($K_{C_{sum}}$) pour des chemins de longueur allant de 1 à 3 .



K_{sum} graph kernel

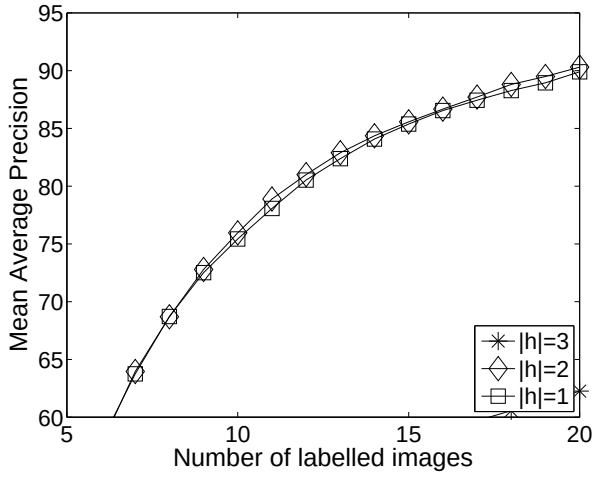


K_{max} graph kernel

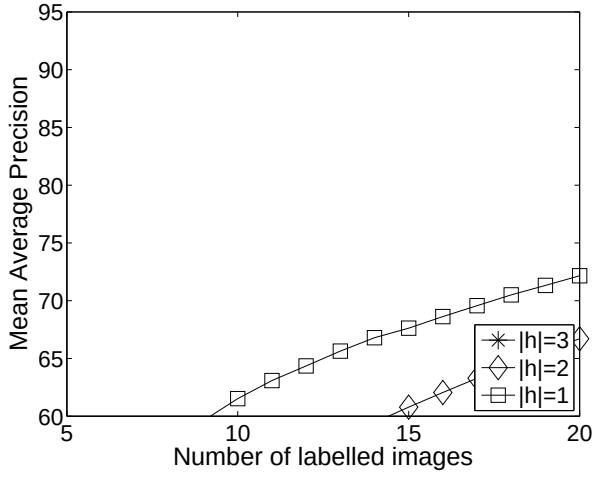


K_{new} graph kernel

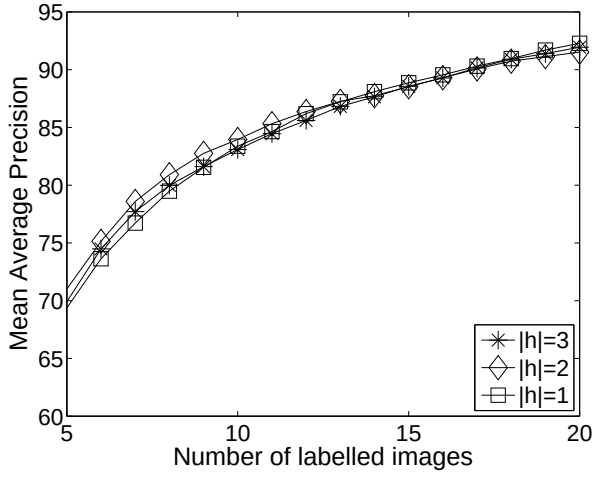
FIGURE 5.19 – Les résultats présentent les expériences pour 3 noyaux sur graphe différents pour un même noyau sur chemin ($K_{C_{mul}}$) pour des chemins de longueur allant de 1 à 3 .



K_{sum} graph kernel

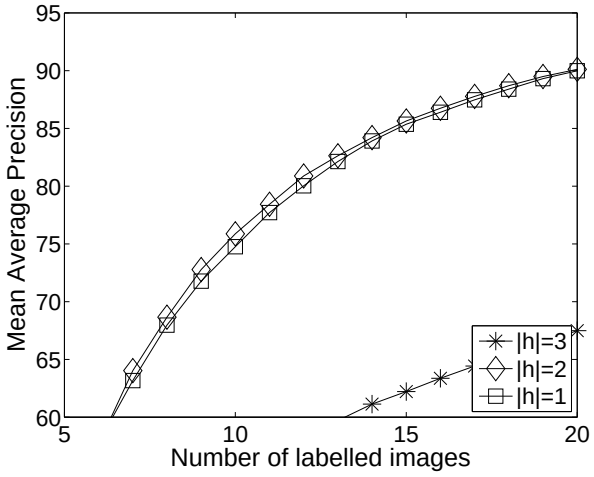


K_{max} graph kernel

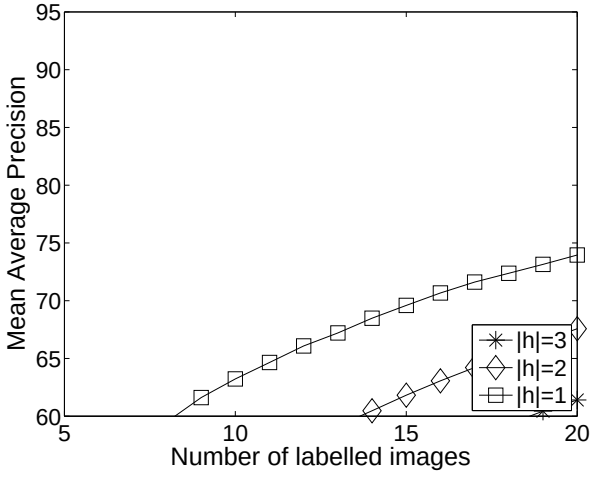


K_{new} graph kernel

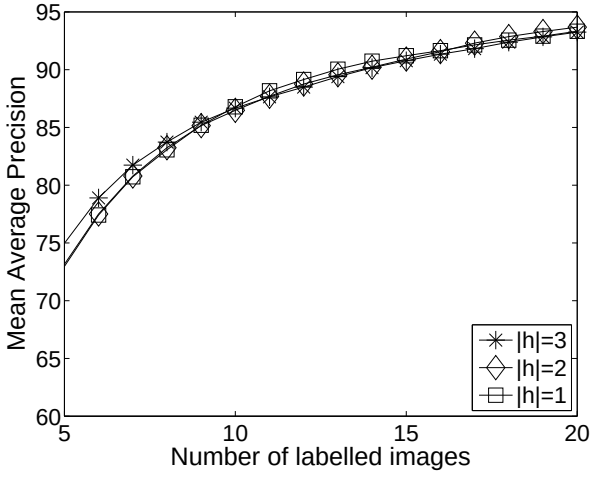
FIGURE 5.20 – Les résultats présentent les expériences pour 3 noyaux sur graphe différents pour un même noyau sur chemin ($K_{C_{new1}}$) pour des chemins de longueur allant de 1 à 3 .



K_{sum} graph kernel



K_{max} graph kernel



K_{new} graph kernel

FIGURE 5.21 – Les résultats présentent les expériences pour 3 noyaux sur graphe différents pour un même noyau sur chemin ($K_{C_{new2}}$) pour des chemins de longueur allant de 1 à 3 .

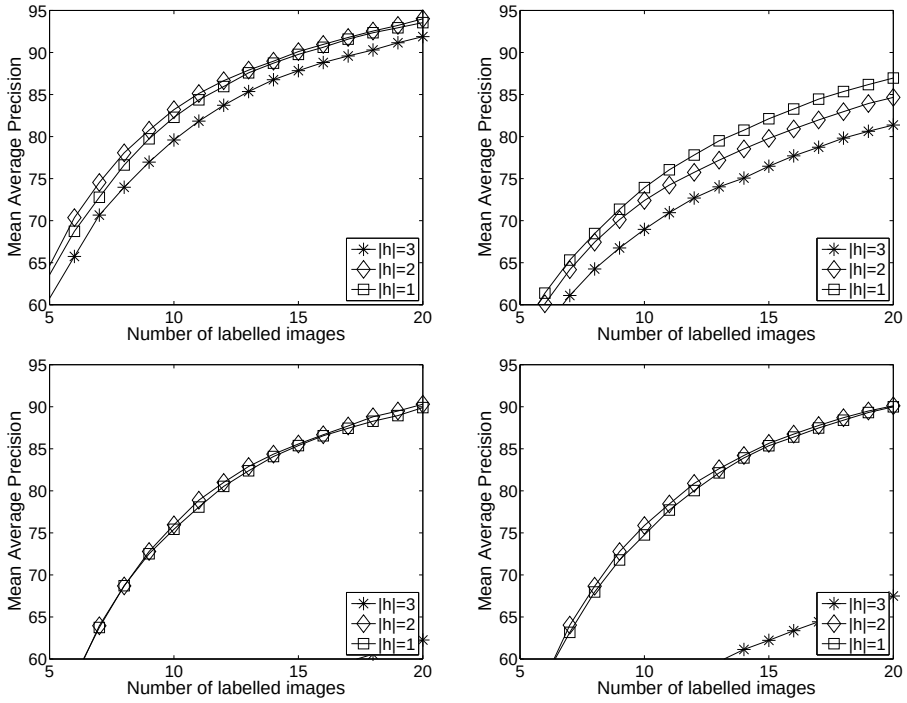


FIGURE 5.22 – Les résultats sont organisés de façon à avoir le même noyau principal(K_{sum}) pour toutes les quatre variations du noyau K_C pour des chemins de longueur allant de 1 à 3 .

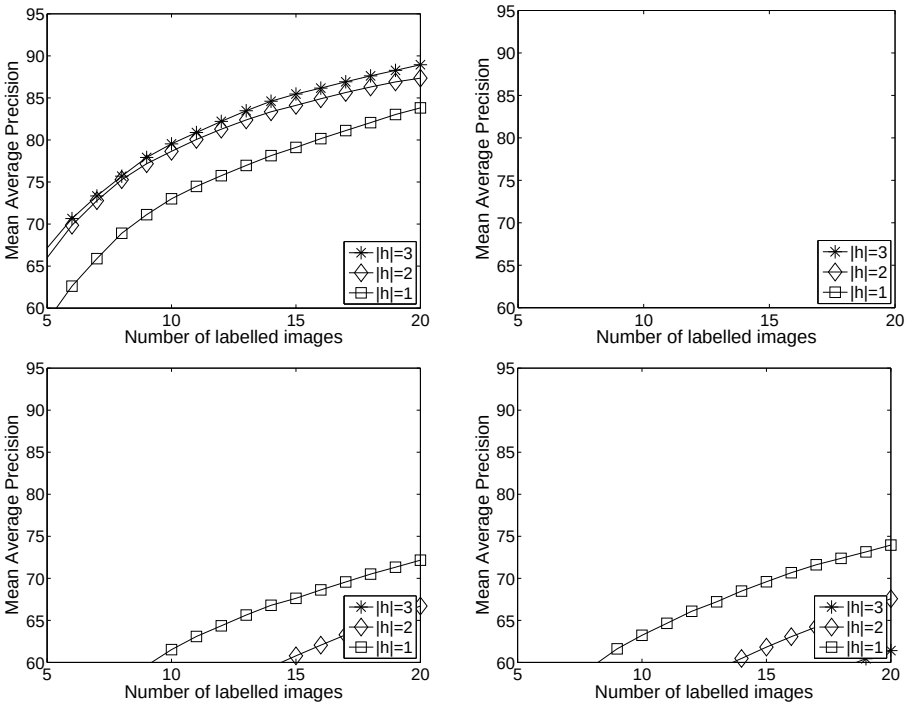


FIGURE 5.23 – Les résultats sont organisés de façon à avoir le même noyau principal(K_{max}) pour toues les quatre variations du noyau K_C pour des chemins de longueur allant de 1 à 3 . Les résultats étant tres mauvais pour l'une des series d'expériences les courbes ne sont pas visibles.

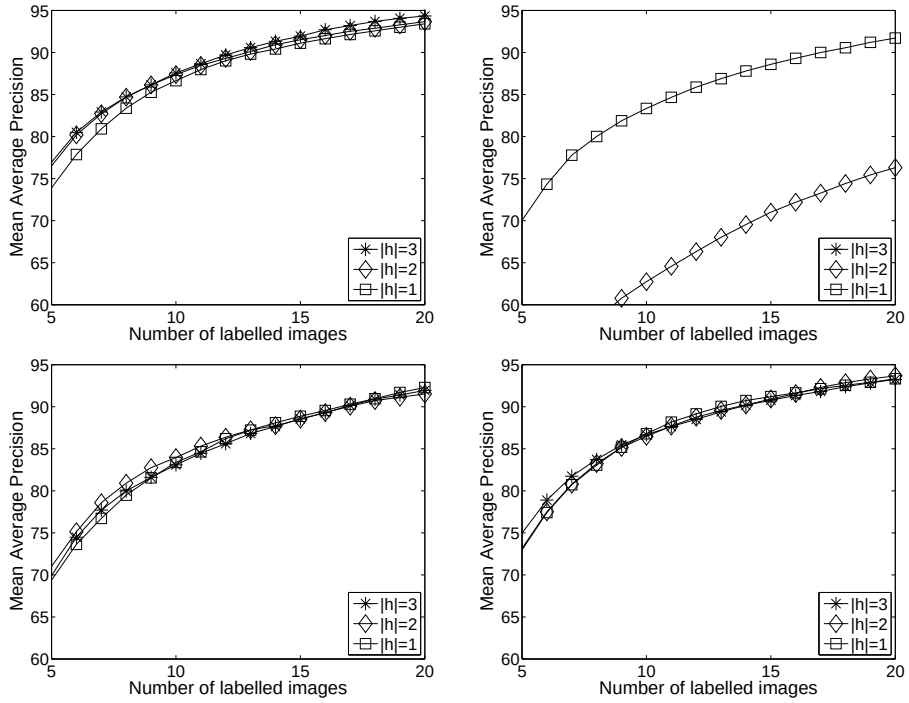


FIGURE 5.24 – Les résultats sont organisés de façon à avoir le même noyau principal(K_{new}) pour toues les quatre variations du noyau K_C pour des chemins de longueur allant de 1 à 3 .

En conclusion, les noyaux sur chemins à valeur non décroissante en fonction de la longueur donnent de meilleurs résultats. Une des raisons probables, c'est qu'ils favorisent l'utilisation d'appariements diversifiés au lieu de restreindre "l'exploration" aux mêmes appariements d'arêtes ou de sommets à fortes valeurs. Le noyau K_{sum}5.4 est le noyau le plus stable quel que soit le noyau sur graphe utilisé mais son coût en temps d'exécution est globalement le plus élevé pour une longueur donnée.

5.3.4 Noyaux sur graphes

Nous avons restreint notre choix à deux types de noyaux en plus du nôtre. Le choix s'est porté sur deux noyaux encadrant le nôtre : le noyau maximum et le noyau somme. Cette étude regarde leur comportement associé aux précédents noyaux sur chemins ainsi que les temps de calculs. Ensuite, nous avons regardé succinctement, les effets des choix de type de sacs de chemins d'un point de vue temps de calculs, efficacité (MAP), et représentation des appariements utilisés sur les graphes. Enfin nous avons comparé notre méthode à d'autres sur des bases plus complexes et diversifiées.

5.3.4.1 Étude comparée de trois noyaux : la somme, le max et celui proposé

Les expérimentations sont résumées sur les figures 5.22, 5.23 et 5.24. La base est est la base Bird et les noyaux mineurs choisis pour les sommets sont des noyaux gaussiens avec une distance du χ^2 et pour les arêtes un noyau gaussien avec une distance du L_1.

Pour rappel les formules 3 noyaux sur graphes testés :

le max(fig. 5.23) :

$$K_{max}(G,G') = \max_{n=1}^{\infty} \sum_{\substack{h\in H(G)\\|h|=n}} \sum_{\substack{h'\in H(G')\\|h'|=n}} K_C(h,h')) \tag{5.8}$$

la somme (fig. 5.22) :

$$K_{somme}(G,G') = \sum_{n=1}^{\infty} \sum_{\substack{h\in H(G)\\|h|=n}} \sum_{\substack{h'\in H(G')\\|h'|=n}} K_C(h,h')) \tag{5.9}$$

notre noyau, un intermédiaire(fig. 5.24) :

$$\begin{aligned} K_{new}(G,G') &= \frac{1}{|V|} \sum_{\substack{v\in G\\ v'\in \mathrm{ppv}_k(v)}} \max_{h\in H_v(G)} \max_{\substack{h'\in H_{v'}(G')\\|h'|=|h|}} K_C(h,h') \\ &+ \frac{1}{|V'|} \sum_{\substack{v'\in G\\ v\in \mathrm{ppv}_k(v')}} \max_{h'\in H_{v'}(G)} \max_{\substack{h\in H_v(G)\\|h|=|h'|}} K_C(h',h) \end{aligned} \tag{5.10}$$

Temps de calculs des divers noyaux Le temps de calcul présenté est un temps de calcul moyen sur la base birds. Pour chaque couple d'image de la base on a effectué le calcul du noyau et obtenu ainsi un temps d'execution. Ce temps dans les graphiques des figures(5.25,5.26) a été divisé par le nombre de couple d'image. Les graphes issus de birds dans nos expériences ont un nombre de sommets moyen de 25 sommets et un sommet possède 4 voisins en moyenne. Le temps de calcul dans ces graphiques se traduit par un temps moyen de calcul pour la valeur noyau d'un couple de graphe. Les temps sont en millisecondes.

A partir de ces temps nous avons obtenu une constante 0.0025 qui correspond à un cout moyen d'une opération dans nos graphes. Ceci nous as permis de calculer les valeurs de deux fonctions $fcout1$ et $fcout2$ représentant un cout moyen de calcul de noyau pour des graphes de 25 sommets et d'un voisinage moyen de quatre sommets à partir des formules d'estimation de complexité du chapitre précédent.

$$fcout1(k) = 0.0025 \times 2n \times d_{moyen}^{2k}) = 0.0025 \times 2 \times 25 \times 4^{2k})$$

et

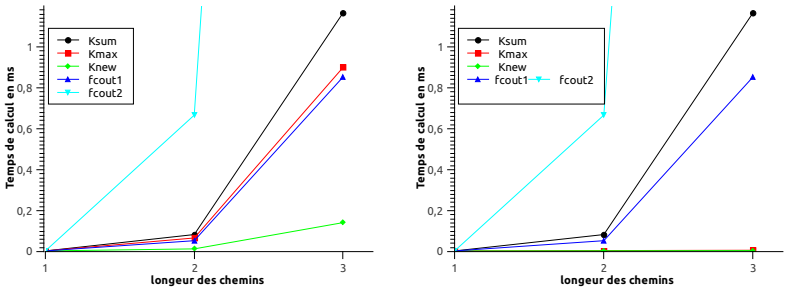
$$fcout2(k) = 0.0025 \times 2n^2 \times d_{moyen}^{2k} = 0.0025 \times 2 \times 25^2 \times 4^{2k})$$

$fcout1$ estime le temps de calcul pour le noyau K_{new} pour des graphes de 25 sommets et d'un voisinage moyen de quatre sommets. $fcout2$ estime le temps de calcul pour les noyaux K_{sum} et K_{max} pour des graphes de 25 sommets et d'un voisinage moyen de quatre sommets.

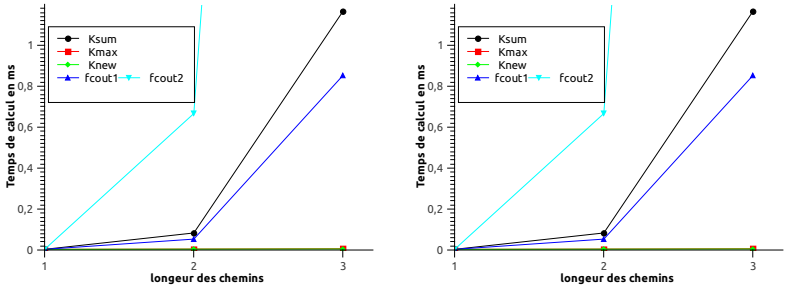
Les temps de calcul sont plus élevés pour le noyau K_{sum} que pour les deux autres et il aussi identique quelque soit le noyau sur le chemin utilisé il est néanmoins inférieur à notre fonction fonction d'estimation du temps de calcul $fcout2$. Pour les noyaux K_{max} et K_{new} on remarque que le choix du noyau sur chemin influe sur le temps de calcul. Le noyau sur chemin $K_{C_{sum}}$ est le noyau sur chemin qui donne les plus grand temps de calcul pour ces deux noyaux alors que les 3 autres donnent des temps de calcul identiques. La figure 5.26 permet de voir que le noyau K_{new} se calcule plus rapidement que le noyau K_{max}.

Tous nos noyaux sur graphes ont des temps de calcul inférieur à leur fonction d'estimation respective. Pour K_{max} et K_{new} on peut penser que l'algorithme de branch and bound propose des coupures efficace de l'arbre. Cette explication permet aussi de comprendre la différence de résultat entre le noyau sur chemin $K_{C_{sum}}$ et les autres : il ne permet pas d'élagage efficace dans l'arbre de recherche. Par contre pour le noyau K_{sum} on ne peut que s'appuyer sur l'explication suivante : les noyaux utilisent des chemins euleriens. Ils diminuent ainsi le nombre de possibilités ce qui revient à supprimer des sommets dans la liste des voisins lors de la construction des chemins de longueur supérieure pour un sommet.

bilan résultats et temps de calculs Le noyau K_{max}(fig. 5.23), n'a des résultats comparables aux autres noyaux sur graphes que combiné avec le noyau somme.



$K_{C_{sum}}$ $K_{C_{mul}}$



$K_{C_{new1}}$ $K_{C_{new2}}$

FIGURE 5.25 – Temps de calcul moyen(issus de la base Birds) pour des graphes de 25 sommets d'un voisinage moyen de 4 sommets et répartis sur quatre graphiques où le noyau sur chemin a été fixé.

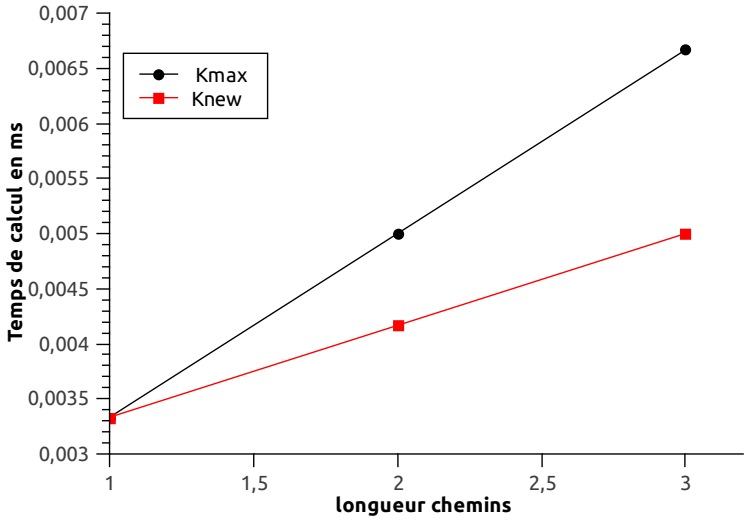


FIGURE 5.26 – Temps de calcul pour moyen(issus de la base Birds) pour des graphes de 25 sommets d'un voisinage moyen de 4 sommets pour les noyaux K_{max} et K_{new}.

Le noyau K_{sum}, commence(5 annotations) avec des valeurs de MAP nettement plus faibles que le noyau K_{new} mais les valeurs des MAP sont proches à partir des 20 annotations. Au vu du temps de calcul nécessaire pour les chemins de longueur 3 (environ 7 secondes si on prend une base de 6000 images) et du faible résultat de ce noyau avec ces chemins, il n'est pas intéressant d'en augmenter la longueur.

Quelque soit le noyau sur chemins utilisé, le nouveau noyau montre une stabilité de résultats par rapport aux deux autres. De plus on remarquera que son temps de calcul est performant et n'augmente pas fortement si on augmente la longueur des chemins à l'inverse des deux autre noyaux. Il présente aussi les meilleurs MAP pour les annotations initiales de 5 images(supérieures à 70% en général).

5.3.4.2 Comparaison avec d'autres méthodes

Les autres méthodes utilisées : régions issues de point d'intérêt(MSER)[90] , le classique global et sacs de régions pondérés. Le global consiste à calculer des signatures de descripteurs sur l'ensemble de l'image. Les sacs de régions pondérés sont une méthode noyau partageant les mêmes régions que nos noyau sur graphe et les mêmes descripteurs sur ces régions mais n'utilisent pas les arêtes. La méthode labellée MSERs utilise un ensemble de régions différentes de nos régions floues.

Nous avons aussi effectué des essais sur des bases objets 3D à partir de régions issues d'une segmentation basée sur le relief et comparé à des résultats de techniques de type sacs de région et de type global.

Bases Caltech et VOC

Ces bases présentent une difficulté croissante par rapport au précédentes bases : elles sont plus grandes et plus diversifiées.

Catégorie	MSER	Global	Régions	$K_{exp}, H_e \vert h \vert \leq 3$
avion	66	60	65	**87**
Arrière-plan	87	**90**	70	76
car	54	82	88	**91**
moto	59	**78**	59	68
Visage	70	77	82	**84**
Moyenne	67	77	72	**81**

TABLE 5.1 – Le tableau affiche les MAP au bout de 50 annotations sur la base Caltech de 4 méthodes dont le noyau sur graphe (basée sur des chaines eulériennes)

Le noyau choisi est différent pour Caltech et VOC. Sur Caltech, le noyau sur graphe choisi est combinaison de $K_{C_{sum}}$ avec K_{max} et une longueur de chemin de 1 à 3 arêtes. Tandis que sur VOC, le choix a été la combinaison $K_{C_{new2}}$ avec K_{new} avec une longueur de chemin de 1 arête.

Le tableau 5.1 montre que les résultats pour 50 annotations sont meilleurs pour 3 catégories sur 5. Les deux catégories où le noyau sur graphe est moins bon que d'autres méthodes sont des catégories où la méthode globale a les meilleurs résultats. On peut supposer que ces catégories portent plus sur des critères de reconnaissance globaux pour retrouver un objet.

Les courbes de MAP de la figure 5.27 se séparent en deux catégories : les méthodes à régions et les autres. Les méthodes à régions possèdent des courbes MAP supérieures aux deux autres méthodes. On remarque aussi les faibles valeurs des MAP même au bout de 50 annotations par rapport aux bases précédemment vues. Ceci est dû à la difficulté de cette base par rapport aux précédentes certaines catégories ayant une grande variabilité. La méthode employant le noyau sur graphe possède un meilleur résultat que celle basée sur des régions sans adjacence.

Les résultats sur les deux bases montrent que le passage de sacs de régions à des graphes de régions permet une meilleure réponse aux requêtes de l'utilisateur dans le cadre d'une recherche d'objet dans une image.

Bases Objet 3D Pour finir nous avons débuté l'expérimentation sur des bases d'objets 3D avec des régions de surface en comparant 3 méthodes : notre noyau sur graphe($K_{new} + K_{C_{new2}} + |h| = 1$) , les sacs de régions et la globale. Pour rappel, les descripteurs des régions sont des CEGI et les descripteurs sur arêtes sont décrit dans les section précédentes.

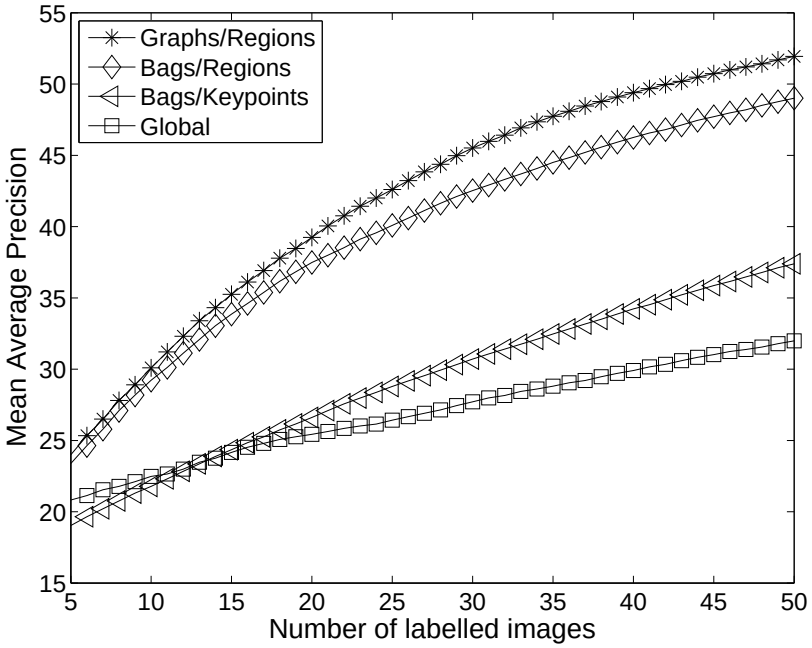


FIGURE 5.27 – La figure montre des expériences effectuées sur la base VOC.

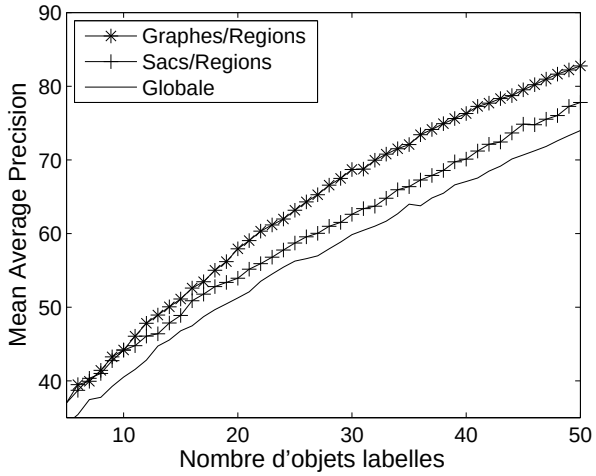


FIGURE 5.28 – EROS3D

Sur la figure 5.28 les méthodes à régions ont de meilleurs résultats que la méthode globale. On note que la méthode avec le noyau sur graphe a de meilleurs résultats que la méthode par sacs.

Contours Les noyaux sur graphes ont été aussi testés sur des graphes de contours dans le cadre de la thèse de Jean Emmanuel Haugeard[91, 92, 93, 94]. Leur utilisation permet d'extraire et de reconnaître des fenêtres dans des images de façade de bâtiments ou des objets. Ils exploitent des chemins de plus grande longueur mais avec des contraintes plus poussées sur les chemins choisis.

5.4 Conclusion

Les expérimentations ont permis d'une part de s'assurer qu'il était intéressant d'utiliser des noyaux sur graphes pour de la recherche d'objet dans des images ou d'objets 3D. D'autre part d'effectuer un pas vers la compréhension d'évaluation des noyaux sur graphes en général. En effet découper les expérimentations en fonction des niveaux de noyaux ont permis d'étudier certains effets de combinaisons ou de paramètres intrinsèques aux noyaux (les chemins dans notre cas). De plus les temps de calcul étant raisonnables, ils permettent d'envisager d'expérimenter ces noyaux sur de plus grandes bases ou des bases donnant des plus grands graphes.

En conclusion, d'une part les noyaux sur graphes ont montré à travers leur tests dans différentes expériences et bases de données leurs possibilités pour la recherche interactive. D'autre part, le choix des différents noyaux mineurs et la combinaison de ceux-ci influe sur la capacité des noyaux à permettre au SVM de différencier les objets ou images dans le cadre d'une recherche.

Conclusion et perspectives

Conclusions

Nous avons présenté des noyaux sur graphes, des méthodes de calculs de ces noyaux et leur application dans la recherche interactive d'images.

Dans le premier chapitre, nous nous sommes intéressés à la problématique de la mise en correspondance de graphes et plus particulièrement aux graphes valués dans l'optique de les utiliser pour comparer deux images. Nous avons vu qu'il existait de nombreuses méthodes pour comparer des graphes. Une méthode a retenu notre attention puisqu'elle permettait de travailler sur des valeurs de similarité exploitable par des méthodes de type SVM : noyaux sur graphes.

Dans le deuxième chapitre, nous avons vu les méthodes à noyaux, les méthodes à noyaux sur graphes et plus particulièrement ceux basés sur des ensembles de chemins issus de graphes. De plus nous avons présenté un nouveau noyau de graphes basé sur ces ensembles de chemin.

Dans le troisième chapitre, nous avons étudié la mise en algorithme du calcul de ces noyaux sur chemins à partir d'arbre de recherches. Nous avons montré que selon un ensemble de contraintes les arbres utilisés était bien des arbres de recherches complets(toutes les solutions présentes) et non redondants(unicité). Nous avons également présenté, un peu plus en détail les algorithmes mis en œuvres.

Enfin dans le dernier chapitre, nous avons rappelé les buts recherchés par nos expériences, présenté le cadre expérimental de nos études sur les noyaux sur chemins, les expériences, les analyses de leurs résultats et également un petit descripteur d'arêtes qui s'est avéré être porteur d'informations.

Nos résultats ont permis de valider l'exploitation en temps réel de la recherche interactive d'images basée sur des noyaux sur ensembles de chemins. Dans le cadre, il s'est avéré de nos expériences que des très petits chemins étaient suffisants pour une reconnaissance correcte dans ce cadre avec un faible nombre d'itérations. Enfin le noyau que nous avons proposé possède une bonne stabilité de résultats et de temps de calculs quelque soit le noyau sur chemin utilisé contrairement aux autres noyaux comparés.

Perspectives

Cette thèse a commencé par la comparaison des méthodes celle de Kashima et de FreBir, en montrant qu'elles étaient proches, leur principale différence étant que l'une reposait sur un noyau de Mercer et pas l'autre. Il s'est avéré que la littérature est de plus en plus riche, néanmoins on manque d'outils d'études pour les noyaux sur graphes et de recul pour effectuer des choix parmi les multiples combinaisons possibles entre les différents niveaux de noyaux.

Une fin de thèse est toujours une frustration en soit, parcequ'à un moment, il faut arrêter de penser aux solutions envisageables et les expérimentations.

Les noyaux sont des outils intéressants actuellement en forte évolution. Par exemple, les nouveaux types de noyaux, non définis positifs, sont valables sans avoir à calculer de matrice de Gram. L'utilisation de ces noyaux appliqués aux ensembles de chemins est une voie à explorer.

Dans pratiquement toute cette thèse, les informations considérées n'ont été que sommets ou adjacences or l'information de non adjacence peut elle aussi être importante. Mettre en application les pistes de prise en compte de cette information de non adjacence entre sommets pourrait se révéler intéressante.

Pour pallier à la gourmandise en temps de calcul possible des propositions précédentes, les représentations présentées dans cette thèse permettent le calcul parallèle de leurs algorithmes associés.

Utiliser les noyaux non définis comme le max pour récupérer les appariements et mieux étudier ceux induits par les noyaux sur chemins permettrait peut-être de trouver de meilleurs noyaux sur chemins pour les noyaux de graphes en général.

Annexe A

Preuves des lemmes

lemme 4.1.1

Théorème A.0.1. *Soit H une fonction générant un ensemble de chemins à partir de graphe.*
Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes.
Soit $T(H^k, G, G')$ un arbre d'appariement
Alors on a : $App(H^k(G), H^k(G')) = \Xi(T(H^k, G, G'))$.
Autrement dit :
Quelque soit l'appariement entre deux chemins h de $H^k(G)$ et h' de $H^k(G')$ il existe un chemin c de la racine à la feuille de l'arbre T tel que : $c = (root)\binom{h}{h'}$
Quelque soit le chemin $c = (root)\binom{v_0}{v'_0} \cdots \binom{v_p}{v'_p}$ de la racine à une feuille de l'arbre T, $\alpha = (v_0...v_p, v'_0...v'_k)$ est un appariement entre deux chemins de $H^k(G)$ et $H^k(G')$.

Preuve

① Montrons que $\Xi(T(H^k, G, G'))$ est inclus dans $App(H(G), H(G'))$.
Soit un chemin $c_f = (root)\binom{h_f}{h'_f}$ de l'arborescence T.
Soit f la feuille du chemin c. D'après définition :
$\exists h \in H^k(G)$, h_f début de h
$\exists h' \in H^k(G')$, h'_f début de h'
$|h_f| = |h'_f|$
Si $|h_f| \leq |h|$
(rappel $|h| = |h'| = k$) alors
$\exists v_0...v_p$, $h = hfv_0...v_p$
$\exists v'_0...v'_p$, $h' = hfv_0...v_p$
Or $\exists! c_{bis} = (root)\binom{h}{h'}$ d'après définition de T
Donc c_{bis} inclut le chemin c par conséquent, la feuille f a forcement un fils (absurde)
Donc $|h'_f| = |h_f| = |h| = |h'| = k$

Donc $(h_f, h'_f) \in App(H(G), H(G'))$

② Montrons que $App(H(G), H(G'))$ est inclus dans $\Xi(T(H^k, G, G'))$
Soit un appariement $\alpha = (h, h')$ entre deux chemins de$H(G)$ et $H(G')$
D'après définition de $T(H^k, G, G')$, il existe un chemin c dans T tel que :
$c = (root)\binom{h}{h'} = (root)\binom{v_0 \dots v_k}{v'_0 \dots v'_k}$
Supposons que le nœud $n = \binom{v_k}{v'_k}$ du chemin c ne soit pas une feuille.
$\Rightarrow$ n a donc au moins un fils $n_f = \binom{v}{v'}$ dans T
$\Rightarrow (root)\binom{hv}{h'v'}$ est un chemin de T Alors :
$\exists h_x \in H^k(G)$, hv est le début de h_x
$\exists h'_x \in H^k(G')$, $h'v'$ est le début de h'_x
$\Rightarrow |hv| \leq |h_x|$ et $|h'v'| \leq |h'_x|$
$\Rightarrow k+1 \leq k$ et $k+1 \leq k$ (absurde)
$\Rightarrow$ le nœud n n'a pas de fils $\Rightarrow \alpha \in \Xi(T(H, G, G'))$

■

Théorème A.0.2. *Soit une fonction générative H.*
Soit la famille d'arborescences F_H associée à H.
Pour tout couple de graphes G et G' on a :

$$App(H(G), H(G')) = \cup_{i=0}^{+\infty} \Xi(T_i(G, G'))$$

Avec T_i des arborescences de F_H restreintes à H^i.

Ce théorème justifie l'obtention de tous les appariements possibles. Il est facilement démontrable :

Preuve. On a par définition $App(H(G), H(G')) = \cup_{i=0}^{+\infty} App(H^i(G), H^i(G'))$ car *App* n'apparie que des chemins de même longueur. Or d'après le théorème 4.1.1 on a : $App(H^k(G), H^k(G')) \equiv \Xi(T(H^k, G, G'))$ donc $App(H(G), H(G')) = \cup_{i=0}^{+\infty} \Xi(T(H^i, G, G'))$ soit $App(H(G), H(G')) = \cup_{i=0}^{+\infty} \Xi(T_i(G, G'))$

■

A.1

Théorème A.1.1. *Soit une fonction générative H qui produit des ensembles de chemins récursifs, et la famille d'arborescence F_H associée à H.*
Alors, Pour tout couple de graphes G et G' on on peut construire T_{k+1} à partir de T_k, G, G' et H_k

Preuve. On construit un arbre T à partir de T_k de la façon suivante : Pour toute feuille f de t_k associée aux chemins on rajoute les nœuds fils tels que les arêtes et

sommets associés à un nœud $(ev, e'v')$, hev et $h'e'v'$ forment des chemins dans $H_{k+1}(G)$ et $H_{k+1}(G')$.

On supprime toute branche de l'arbre ne possédant aucune feuille de profondeur $k+1$

T_k contient tous les appariements entre $H_k(G)$ et $H_k(G')$ au niveau de ses feuilles.

Or H produit des ensembles récursifs donc $H_{k+1}(G)$ et $H_{k+1}(G')$ peuvent être construits à partir des chemins de $H_k(G)$ et respectivement $H_k(G')$. Toutes les feuilles de T correspondent à tous les appariements de chemins possibles entre $H_{k+1}(G)$ et $H_{k+1}(G')$.

Comme on a supprimé toutes les branches n'ayant pas de feuille de profondeur $k+1$, il ne reste que les appariements entre les chemins de longueur $k+1$. Par conséquent $T = T_{k+1}(H, G, G')$.

On a bien construit $T_{k+1}(H, G, G')$ à partir de T_k, G, G' et H_k

■

Bibliographie

[1] C. Berge, *Théorie des Graphes et ses Applications.* Collection Universitaire de Mathématiques, Dunod, 1958.

[2] J. R. Ullman, "An algorithm for subgraph isomorphism," in *J. Assoc. Comput.*, Mach 1976, pp. 31–42.

[3] D. E. Ghahraman, A. K. C. Wong, and T. Au, "Graph monomorphism algorithms," *IEEE Transaction on System, Man and Cybernetics*, vol. 10, pp. 189–197, 1980.

[4] L. P. Cordella, P. Foggia, C. Sansone, F. Tortorella, and M. Vento, "Graph matching : a fast algorithm and its evaluation," in *IEEE International Conference on Pattern Recognition*, 1998, pp. 1582–1584.

[5] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "Subgraph transformation for the inexact matching of attributed relational graphs," *International journal of Computing*, vol. 12, pp. 43–52, 1998.

[6] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An improved algorithm for matching large graphs," in *IAPR-TC15 Workshop p, Graph-Based Representations in Pattern Recognition*, 2001, pp. 149–159.

[7] A. R., S. Fischer, and H. Bunke, "Graph edit distance with node splitting and merging, and its application to diatom identification," in *IAPR-TC15 Workshopp on Graph-based Representation in Pattern Recognition*, 2003, pp. 95–106.

[8] M. Neuhaus, K. Riesen, and H. Bunke, "Fast suboptimal algorithms for the computation of graph edit distance," in *SSPR/SPR*, 2006, pp. 163–172.

[9] K. Riesen, M. Neuhaus, and H. Bunke, *Bipartite Graph Matching for Computing the Edit Distance of Graphs.* Springer Berlin / Heidelberg, 2007.

[10] P. M. Pardalos, J. Rappe, Mauricio, and M. G. Resende, "An exact parallel algorithm for the maximum clique problem," in *In High Performance and Software in Nonlinear Optimization.* Kluwer Academic Publishers, 1998, pp. 279–300.

[11] Y. Shinano, T. Fujie, Y. Ikebe, and R. Hirabayashi, "Solving the maximum clique problem using pubb," in *IEEE International Parallel and Distributed Processing Symposium*, 1998, pp. 326–332.

[12] B. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. 30, pp. 45–87, 1981.

[13] B. Messmer, "Efficient graph matching algorithm for preprocessed model graphs," Ph.D. dissertation, Institut fur Informatik und Angewandte Mathematik, University of Bern, 1995.

[14] B. T. Messmer and H. Bunke, "Efficient subgraph isomorphism detection : a decomposition approach," in *IEEE Transaction on Knowlegde and Data Engineering*, vol. 12, 2000, pp. 307–323.

[15] C. Irniger and H. Bunke, "Graph matching : filtering large databases of graphs using decision trees," in *IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, 2001, pp. 239–249.

[16] W. H. Tsai and K. S. Fu, "Error-correcting isomorphisms of attributed relational graphs for pattern analysis," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, pp. 757–768, 1979.

[17] ——, "Subgraph error-correcting isomorphisms for syntactic pattern recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, pp. 48–62, 1983.

[18] H. Bunke and X. Jiang, "Graph matching and similarity," in *Intelligent Systems and Interfaces*, ser. International series in intelligent technologies, H.-N. Teodorescu, D. Mlynek, A. Kandel, and H.-J. Zimmermann, Eds. Dordrecht, The Netherlands : Kluwer Academic Publishers, 2000, pp. 281–301.

[19] T. Gärtner, K. Driessens, and J. Ramon, "Graph kernels and gaussian processes for relational reinforcement learning," in *ILP*, 2003, pp. 146–163.

[20] A. C. M. Dumay, R. J. van der Geest, J. J. Gerbrands, E. Jansen, and J. H. C. Reiber, "Consistent inexact graph matching applied to labelling coronary segments in arteriograms," in *IEEE International Conference on Pattern Recognition*, 1992, pp. 439–442.

[21] S. Berretti, A. D. Bimbo, and E. Vicario, "Efficient matching and indexing of graph models in content-based retrieval," *IEEE Trans. on PAMI*, vol. 23, no. 10, pp. 1089–1105, 2001.

[22] L. Gregory and J. Kittler, "Using graph search techniques for contextual colour retrieval," in *IAPR International Workshops on SSPR and SPR*, 2002, pp. 186–194.

[23] R. Allen, L. Cinque, S. Tanimoto, L. G. Shapiro, and D. Yasuda, "A parallel algorithm for graph matching and its maspar implementation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, pp. 490–501, 1997.

[24] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE Transactions on Pattern Analysis and Machine Intellenge*, vol. 10, pp. 695–703, 1988.

[25] L. Xu and I. King, "A pca approach for fast retrieval of structural patterns in attributed graphs," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 31, pp. 812–817, 2001.

[26] M. Carcassoni and E. R. Hancock, "Weighted graph-matching using modal clusters," in *IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, 2001, pp. 260–269.

[27] S. Kosinov and T. Caelli, "Inexact multisubgraph matching using graph eigenspace and clustering models," in *IAPR International Workshops on SSPR and SPR*, 2002, pp. 133–142.

[28] M. Leordeanu and M. Hebert, "A spectral technique for correspondence problems using pairwise constraints," in *International Conference of Computer Vision (ICCV)*, October 2005, pp. 1482–1489.

[29] A. C. Berg, T. L. Berg, and J. Malik, "Shape matching and object recognition using low distortion correspondence," in *IEEE International Conference on Computer Vision and Pattern Recognition*, 2005, pp. 26–33.

[30] Y. Zheng and D. Doermann, "Robust point matching for nonrigid shapes by preserving local neighborhood structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28(4), p. 643, 2006.

[31] O. Duchenne, F. Bach, I. Kweon, and J. Ponce, "A tensor-based algorithm for high-order graph matching," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, November 2009.

[32] B. T. Messmer and H. Bunke, "A new algorithm for error-tolerant subgraph isomorphism detection," *IEEE Transactions on Pattern Analysis and Machince Intelligence*, vol. 20, pp. 493–504, 1998.

[33] F. Fuchs and H. L. Men, "Building reconstruction on aerial images through multi-primitive graph matching," in *IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, 1999, pp. 21–30.

[34] ——, "Efficient subgraph isomorphism with a priori knowledge," in *IAPR International Workshops on SSPR and SPR*, 2000, pp. 427–436.

[35] P. N. Suganthan, "Attributed relational graph matching by neural-gas networks," in *IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, 2000, pp. 366–374.

[36] C. W. Liu, K. C. Fan, J. T. Horng, and Y. K. Wang, "Solving weighted graph matching problem by modified microgenetic algorithm," in *IEEE International Conference on Systems, Man and Cybernetics*, 1995, pp. 638–643.

[37] A. Perchant, C. Boeres, I. Bloch, M. Roux, and C. Ribeiro, "Model-based scene recognition using graph fuzzy homomorphism solved by genetic algorithm," in *IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, 1999, pp. 61–70.

[38] K. G. Khoo and P. N. Suganthan, "Multiple relational graphs mapping using genetic algorithms," in *Congress on Evolutionary Computation*, 2001, pp. 727–733.

[39] O. Sammoud, S. Sorlin, C. Solnon, and K. Ghedira, "A comparative study of ant colony optimization and reactive search for graph matching problems," in *6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, 2006, pp. 287–301.

[40] A. Hlaoui and S. Wang, "A new algorithm for graph matching with application to content-based image retrieval," in *IAPR International Workshops on SSPR and SPR*, 2002, pp. 291–300.

[41] S. Kirkpatrick, J. Gelatt, and M. Vecchi, "Optimisation by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[42] B. Selman, H. Levesque, and D. Mitchell, "A new method for solving hard satisfiability problems," in *National Conference on Artificial Intelligence*, 1992, pp. 440–446.

[43] F. Glover, "Tabu search," *Journal on Computing*, pp. 190–260, 1989.

[44] S. Sorlin, O. Sammoud, C. Solnon, and K. Ghedira, "Etude comparative de aco et de tabou réactif sur des problémes d'appariement de graphes," in *2èmes Journées Francophones de Programmation par Contraintes (JFPC'06)*, June 2006, pp. 317–326.

[45] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels : Hardness results and efficient alternatives," in *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*, 2003.

[46] H. Kashima and Y. Tsuboi, "Marginalized kernels between labeled graphs," in *International Conference on Machine Learning (ICML)*, Washington, DC USA, August 2003, pp. 321–328.

[47] Z. Harchaoui and F. Bach, "Image classification with segmentation graph kernels," in *International Conference on Computer Vision and Pattern Recognition*, 2007.

[48] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, 2005.

[49] V. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, New York, 1998.

[50] A. Smola, P. Barlett, B. Scholkopf, and C. Schuurmans, *Advances in Large Margin Classifiers*. MIT Press, 2000.

[51] J. Shawe-Taylor and N. Cristianini, *Kernel methods for Pattern Analysis*. Cambridge University Press, ISBN 0-521-81397-2, 2004.

[52] C. Wallraven, B. Caputo, and A. Graf, "Recognition with local features : the kernel recipe," in *International Conference on Computer Vision (ICCV)*, vol. 2, 2003, pp. 257–264.

[53] S. Lyu, "Mercer kernels for object recognition with local features," in *IEEE International Conference on Computer Vision and Pattern Recognition*, San Diego, CA, 2005.

[54] B. Haasdonk, "Feature space interpretation of svms with indefinite kernels," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 4, pp. 482–492, 2005.

[55] B. Haasdonk and E. Pekalska, "Indefinite kernel fisher discriminant," in *Int. Conf. on Pattern Recognition (ICPR)*, Tampa, USA, Dec. 2008, pp. 1–4.

[56] D. Haussler, "Convolution kernels on discrete structures," Institute of California at Santa Cruz, Tech. Rep., 1999.

[57] T. Gärtner, "A survey of kernels for structured data," *SIGKDD Explorations*, vol. 5, no. 1, pp. 49–58, 2003.

[58] H. Kashima and Y. Tsuboi, "Kernel-based discriminative learning algorithms for labeling sequences, trees and graphs," in *International Conference on Machine Learning (ICML)*, Banff, Alberta, Canada, 2004, p. 58.

[59] F. Suard, V. Guigue, A. Rakotomamonjy, and A. Bensrhair, "Pedestrian detection using stereo-vision and graph kernels," in *Intelligent Vehicles Symposium*, Las Vegas, Nevada, June 2005, pp. 267–272.

[60] F.-X. Dupé and L. Brun, "Classification de formes avec un noyau sur graphes flexibles et robuste au bruit," in *RFIA*, Caen, France, Juanary 2010.

[61] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," in *ISMB (Supplement of Bioinformatics)*, 2005, pp. 47–56.

[62] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *AISTATS*, 2009.

[63] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine Learning*, vol. 75, no. 1, pp. 3–35, 2009.

[64] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernel on graphs," in *NIPS*, 2009.

[65] R. C. Wilson and E. R. Hancock, "Pattern spaces from graph polynomials," in *12th International Conference On Image Analysis And Processing*, 2003, pp. 480–485.

[66] Y. Chen and J. Wang, "Image categorization by learning and reasoning with regions," *International Journal on Machine Learning Research*, vol. 5, pp. 913–939, 2004.

[67] F. Moosmann, E. Nowak, and F. Jurie, "Randomized clustering forests for image classification," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 30, september 2008.

[68] H. Bunke and K. Riesen, "A family of novel graph kernels for structural pattern recognition," in *CIARP*, vol. 4756, 2007, pp. 20–31.

[69] A. M. Smalter, J. Huan, and G. H. Lushington, "Gpm : A graph pattern matching kernel with diffusion for chemical compound classification," in *BIBE*, 2008, pp. 1–6.

[70] I. Gondra and D. Heisterkamp, "Learning in region-based image retrieval with generalized support vector machines," in *IEEE International Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, vol. 27, june 2004, pp. 149–149.

[71] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *International Conference on Machine Learning (ICML)*, 2009.

[72] J. Eichhorn and O. Chapelle, "Object categorization with svm : kernels for local features," Max Planck Institute, Tech. Rep., 2004.

[73] P. Gosselin, M. Cord, and S. Philipp-Foliguet, "Kernel on bags of fuzzy regions for fast object retrieval," in *IEEE International Conference on Image Processing*, San Antonio, Texas, USA, September 2007.

[74] T. K. K. Tsuda and K. Asai, "Marginalized kernels for biological sequences," *Bioinformatics*, 2002.

[75] V. D. Of, S. V. N. Vishwanathan, and A. J. Smola, "Fast kernels for string and tree matching," in *Advances in Neural Information Processing Systems 15*. MIT Press, 2003, pp. 569–576.

[76] S. Philipp-Foliguet and J. Gony, "FReBIR : Fuzzy regions-based image retrieval," in *Information Processing and Management of Uncertainty (IPMU)*, Paris, France, July 2006.

[77] N. Govert and G. Kazai, "Overview of the initiative for the evaluation of xml retrieval," *In Fuhr et al*, pp. 1–17, 2003.

[78] H. P. Moravec, "Towards Automatic Visual Obstacle Avoidance," *Proc. 5th International Joint Conference on Artificial Intelligence*, 1977.

[79] M. S. C Harris, "A combined corner and edge detector," *Alvey vision conference*, 1988.

[80] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," *Image and Vision Computing*, vol. 22, no. 10, pp. 761–767, 2004, british Machine Vision Computing 2002. [Online]. Available : http://www.sciencedirect.com/science/article/B6V09-4CPM632-1/2/7e4b5f8aa5a4d6df0781ecf74dfff3c1

[81] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "From contours to regions : An empirical evaluation," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 0, pp. 2294–2301, 2009.

[82] J.C. Bezdek, J. Keller, Krisnapuram, and S. K. Pal, "Fuzzy models and algorithms for pattern recognition and image processing," *ed. Kluwer*, 2000.

[83] S. Philipp-Foliguet, J. Gony, and P.-H. Gosselin, "Frebir : an image retrieval system based on fuzzy region matching," *Computer Vision and Image Understanding*, vol. 113, no. 6, pp. 693–707, June 2009. [Online]. Available : http://publi-etis.ensea.fr/2009/PGG09

[84] J. Cousty, G. Bertrand, L. Najman, and M. Couprie, "Watershed Cuts : Minimum Spanning Forests and the Drop of Water Principle," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 31, no. 8, pp. 1362–1374, Aug. 2009. [Online]. Available : Watershed%2Cminimumspanningforest%2Cminimumspanningtree%2Cgraph%2Cmathematicalmorpholog

[85] S. Philipp-Foliguet, M. Jordan, L. Najman, and J. Cousty, "Artwork 3d model database indexing and classification," *Pattern Recognition*, vol. 44, no. 3, pp. 588–597, 2011. [Online]. Available : http://www.sciencedirect.com/science/article/B6V14-515SR97-1/2/24345afb0ad4fa103c712cd5a318a4c9

[86] M. Alcoverro, S. Philipp-Foliguet, M. Jordan, L. Najman, and J. Cousty, "Region-based 3d artwork indexing and classification," in *Proceedings of the IEEE 3DTV-Con Conference*, Istanbul, May 28-30 2008. [Online]. Available : http://publi-etis.ensea.fr/2008/APJNC08

[87] Allen, "An interval-based representation of temporal knowledge," *Proceedings of IJCAL*, pp. 221–226, 1981.

[88] P. Matsakis, J. M. Keller, O. Sjahputera, and J. Marjamaa, "The use of force histograms for affine-invariant relative position description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1–18, 2004.

[89] C. Hudelot, J. Atif, and I. Bloch, "Fuzzy spatial relation ontology for image interpretation," *Fuzzy Sets and Systems*, vol. 159, no. 15, pp. 1929–1951, 2008, from Knowledge Representation to Information Processing and Management - Selected papers from the French Fuzzy Days (LFA 2006). [Online]. Available : http://www.sciencedirect.com/science/article/B6V05-4RY8SHJ-1/2/42496d7869af0aaa3616309d

[90] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Rosbut wide baseline stereo from maximally stable external regions," in *Proceedings of the Bristish Machine Vision Conference*, 2002, pp. 384–393.

[91] J.-E. Haugeard, S. Philipp-Foliguet, and P.-H. Gosselin, "Kernel on graphs based on dictionary of paths for image retrieval," in *20th International Conference on Pattern Recognition (ICPR)*, august 2010.

[92] J.-E. Haugeard and S. Philipp-Foliguet, "Recherche d'objets par appariement de graphes combinant contours et regions," in *RFIA : Reconnaissance des Formes et Intelligence Artificielle*, CAEN, jan 2010.

[93] J.-E. Haugeard, S. Philipp-Foliguet, and F. Precioso, "Windows and facades retrieval using similarity on graph of contours," in *IEEE International Conference on Image Processing (ICIP 09)*. Citeseer, November 2009.

[94] J.-E. Haugeard, S. Philipp-Foliguet, F. Precioso, and J. Lebrun, "Extraction of windows in facade using kernel on graph of contours," in *Image Analysis*, ser. LNCS, A.-B. Salberg, J. Y. Hardeberg, and R. Jenssen, Eds., vol. 5575. Springer, 2009, pp. 646–656.

Résumé

Les graphes sont des modèles de représentation qui permettent de modéliser un grand nombre de type de documents. Dans cette thèse, nous nous intéressons à leur utilisation pour la recherche dans des bases de données multimédia. Nous commençons par présenter la théorie autour des graphes ainsi qu'un aperçu des méthodes qui ont été proposées pour leur mise en correspondance. Puis, nous nous intéressons plus particulièrement à leur utilisation pour la reconnaissance des formes et l'indexation multimédia. Dans le but de répondre de la manière la plus générique possible aux différents problèmes de recherche, nous proposons de travailler dans le cadre des fonctions noyaux. Ce cadre permet de séparer les problèmes liées à la nature des documents de ceux apportés par les différents types de recherche. Ainsi, toute notre énergie est consacrée à la conception de fonctions de mise en correspondance, mais en gardant à l'esprit qu'elles doivent respecter un certain nombre de propriétés mathématiques. Dans ce cadre, nous proposons de nouvelles solutions qui permettent de mieux répondre aux caractéristiques particulières des graphes issus de primitives et descripteurs visuels. Nous présentons aussi les algorithmes qui permettent d'évaluer rapidement ces fonctions. Enfin, nous présentons des expériences qui mettent en lumière ces différentes caractéristiques, ainsi que des expériences qui montrent les avantages qu'offrent nos modèles vis à vis de la littérature.

Abstract

Graphs are models of representation that can model many type of documents. In this thesis, we focus on their use for research in multimedia databases. We begin by presenting the theory of graphs and around an overview of methods that have been proposed for matching. Then, we are particularly interested in their use for recognition forms and multimedia indexing. In order to respond in the most generic possible different research problems, we propose to work within the framework of kernel functions. This framework allows to separate the problems related to the nature of the documents those introduced by the different types of research. Thus, all our energy is devoted to the design of mapping functions, but bearing in mind that they must meet a number mathematical properties. In this context, we propose new solutions that better meet the specific graphs from primitive and visual descriptors. We also present algorithms to quickly assess these functions. Finally, we present experiments that highlight these different characteristics and experiences that show advantages of our models with respect to the literature.