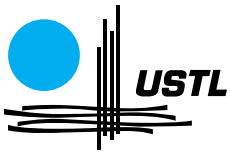

Optimisation de l'indexation multidimensionnelle : application aux descripteurs multimédia.

THÈSE

présentée et soutenue publiquement le 16 Novembre 2007

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Thierry URRUTY

Composition du jury

Président :	Jean-Marc GEIB, Professeur	LIFL, Université de Lille I
Rapporteurs :	Claude CHRISMENT, Professeur	IRIT, Université de Toulouse III
	Joemon JOSE, Associate Professor	University of Glasgow
Examinateurs :	Rémi GILLERON, Professeur	LIFL, Université de Lille III
	Gilles ZURFLUH, Professeur	IRIT, Université de Toulouse I
Directeur :	Chabane DJERABA, Professeur	LIFL, Université de Lille I

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille — UPRESA 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : `direction@lifl.fr`

Remerciements

Tout d'abord, je remercie vivement le professeur Chabane Djeraba de m'avoir accueilli au sein de son équipe pour réaliser ma thèse. Je le remercie pour m'avoir guidé, conseillé et soutenu tout au long de la thèse.

Je suis très reconnaissant envers le professeur Dan A. Simovici pour avoir partagé ses connaissances et ses idées avec moi et pour sa disponibilité. Mes remerciements vont également à Fatima Belkouch pour sa collaboration sur de nombreux travaux. Je la remercie pour son encadrement et son aide pendant ces trois années.

Je tiens à remercier les professeurs Claude Chrisment et Joemon Jose pour avoir accepté d'être les rapporteurs de mon mémoire de thèse, et les professeurs Rémi Gilleron et Gilles Zurfluh pour avoir accepté d'être examinateurs à ma soutenance.

Je remercie vivement le professeur Jean Marc Geib, directeur du Laboratoire d'informatique fondamentale de Lille, d'être le Président de mon jury de thèse et pour m'avoir accueilli au sein de son laboratoire pendant mes trois années de thèse. Je remercie aussi l'ensemble des membres du LIFL, doctorants, chercheurs et personnels administratifs.

Je tiens à saluer l'ensemble de l'équipe Fox-Miire dans laquelle j'ai réalisé ma thèse. Plus particulièrement, je remercie Sylvain pour les nombreuses pauses "café" et discussions, Tarik pour ses conseils le jour et sa collaboration la nuit à ffxi, Chafik pour ses raisonnements posés, Julien pour l'aide en ligne LaTeX, ainsi que les autres doctorants Nacim, Stanislas et Adel pour les discussions diverses et variées.

Je tiens à remercier chaleureusement Mélanie de m'avoir supporté tout au long de ces trois années de thèse et sans qui rien ne serait possible. Je remercie aussi l'ensemble des membres de ma famille et mes amis pour les moments de détente et leurs encouragements.

Résumé

L'importance des documents multimédia et audiovisuels ne cesse d'augmenter. La création, la modification et l'échange de ces documents sont devenus courants. Afin de rendre ces documents facilement accessibles pour tout utilisateur, il est nécessaire d'avoir des moteurs de recherche rapides et efficaces. Mes travaux de thèse s'inscrivent dans le domaine des documents multimédia, à travers un projet exploratoire nommé AVERROES. L'objectif principal est d'utiliser les descripteurs multimédia normalisés Mpeg-7 pour optimiser la gestion d'une base de données de films d'entreprise.

Nous proposons dans ce mémoire deux approches différentes. La première utilise une structure d'indexation multidimensionnelle adaptée à une répartition des données dont la distribution est hétérogène. Pour cela, nos différentes méthodes combinent une technique de classification à une structure d'indexation multidimensionnelle. Dans une deuxième approche, nous proposons d'utiliser une distance structurelle entre deux documents XML pour obtenir une classification préliminaire sur l'ensemble des documents. Un moteur de recherche de séquences vidéo a été développé et permet de tester nos différentes structures d'indexation sur une base de données de films d'entreprises. Enfin, nous proposons d'améliorer l'indexation et la recherche en exploitant les retours des utilisateurs.

Mots clés : données multimédia, indexation, multidimensionnelle, classification, structure XML, journal utilisateur, moteur de recherche

Abstract

Numerical audiovisual and multimedia documents are becoming parts of everyday life for professional users as for end users. This emergence of numerical technologies in the audiovisual sector requires the use of powerful tools and search engine for fast and efficient access to data. My PhD research is included in this multimedia field via an exploratory project named AVERROES. The main objective of this project is to develop tools around the normalized content description standard Mpeg-7, in order to allow efficient audiovisual content search. In this thesis, we propose a first approach that manages all descriptors as a set of features of video sequences. Each video sequence is then represented as a vector in a multidimensional space. This vectorial representation of video sequences has driven the research on multidimensional indexing structure. We present several multidimensional indexing structures that manage a database with an heterogeneous distribution of data. Our different methods combine a clustering technique and a multidimensional indexing structure. The contribution of this combination is to accelerate the response time of a query accessing to the database without losing efficiency. The second approach we propose for managing our Mpeg-7 documents database is to use a structural clustering method based directly on the XML structure. We have developed a search tool and tested it with our indexing structures on a real video database. Finally, we propose an improvement of the quality of our indexing and search tool using feedbacks of users.

Key words : multimedia data, multidimensional indexing, clustering, XML structure, logs, search engine

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Contexte de la recherche

L'importance des documents multimédia et audiovisuels ne cesse d'augmenter. La création, la modification et l'échange de documents numériques sont devenus des utilisations courantes. La prolifération de l'internet haut débit favorise nettement l'échange de ces données numériques. De plus, la qualité des données multimédia présentes sur la toile augmente parallèlement aux débits proposés par les fournisseurs d'accès à internet. De grandes bases de données telles que Internet Movie Database [IMD], YouTube [Tub] ou encore Dailymotion [Dai] apparaissent et deviennent des sites incontournables. Pour que ces documents soient facilement accessibles pour tout utilisateur, il est nécessaire d'avoir des moteurs de recherche rapides et efficaces.

Le travail de ma thèse s'inscrit dans ce contexte par l'intermédiaire d'un projet exploratoire nommé AVERROES. Ce projet a pour but l'utilisation des descripteurs multimédia normalisés Mpeg-7 [Mpe] pour optimiser la gestion d'une base de données de films d'entreprise. Nous détaillons en annexe le projet AVERROES et la norme Mpeg-7. Nous présentons dans la figure 1.1, les grandes étapes du projet. Une première partie du projet consiste à choisir parmi les descripteurs Mpeg-7 existants ceux qui serviront à annoter manuellement l'ensemble des vidéos. L'annotation manuelle de la base de données de films d'entreprise a été réalisée par l'expert métier, partenaire du projet. Nous obtenons alors une base de documents Mpeg-7 contenant la description de chaque séquence vidéo. Mon travail de thèse s'inscrit dans l'indexation optimisée de cette base de données. Le but étant d'accéder le plus rapidement possible aux informations pertinentes pour répondre aux requêtes des utilisateurs. L'application souhaitée par notre partenaire métier est de réutiliser au mieux la base de données vidéo (appelées *rushs*) et éviter ainsi d'envoyer systématiquement une équipe de tournage sur le terrain. Par exemple : retrouver dans les rushs, les séquences vidéo qui contiennent des machines dans un environ de campagne est largement plus économique que d'envoyer une équipe de tournage pour filmer une telle scène.

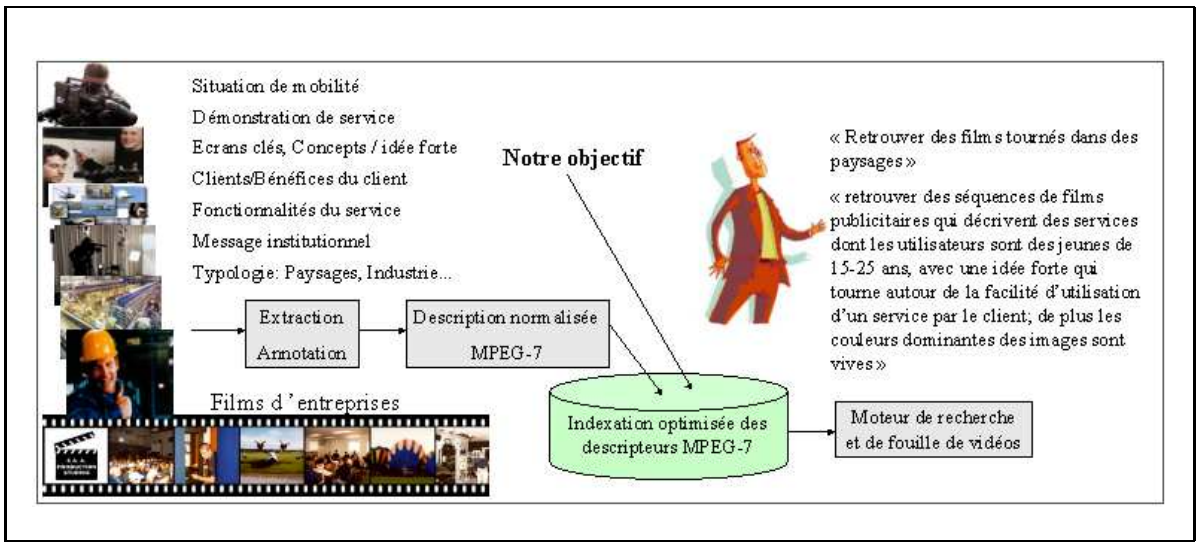

FIG. 1.1 – Schéma global du projet AVERROES

1.2 Problématique et contributions

L'objectif de mon travail de recherche est de proposer une méthode d'indexation optimisée pour la gestion d'une base de données de documents Mpeg-7 utilisant le langage XML. L'objectif de ce travail est de pouvoir fournir un accès rapide et efficace aux informations contenues dans les séquences vidéo. Nous avons orienté nos recherches vers deux méthodes pour gérer notre base de données de documents Mpeg-7.

La première méthode est de gérer l'ensemble des descripteurs comme un ensemble de caractéristiques d'une séquence vidéo. Chaque séquence vidéo peut être alors représentée comme un vecteur dans un espace multidimensionnel. Nous avons donc axé dans un premier temps nos recherches vers le domaine des structures d'indexation multidimensionnelle.

Une base de données de séquences vidéo est par nature hétérogène. Chaque séquence vidéo est décrite par un ensemble de caractéristiques, par exemple une *forêt* au bord d'un *lac* avec quelques *chalets* dans la *montagne*. Mais une séquence vidéo ne peut contenir toutes les caractéristiques en même temps, par exemple *nature*, *industrie*, *sport* et *animaux* ont une très faible probabilité de correspondre à des annotations de la même séquence vidéo. C'est pourquoi, nous avons proposé une première méthode d'indexation multidimensionnelle adaptée à

une base de données dite *hétérogène*, i.e. il est possible de trouver une classification de la base de données. Pour cela, nos différentes approches combinent une méthode de classification à une structure d'indexation multidimensionnelle. L'apport de cette combinaison est d'indexer au mieux chaque séquence de la base de données pour accéder rapidement aux données répondant aux requêtes des utilisateurs. Les performances obtenues montrent que nos approches sont efficaces pour des bases de données volumineuses de dimension élevée, les temps de réponse sont significatifs et le nombre d'accès aux données est fortement diminué par rapport à d'autres méthodes de l'état de l'art.

Nos approches nous ont conduit à étudier les techniques de classification existantes. En effet, une base de données volumineuse de dimension élevée requiert une technique de classification adaptée. L'obtention de classes homogènes est un paramètre déterminant pour de bonnes performances dans notre structure d'indexation multidimensionnelle. Nous proposons une technique de classification combinant des idées provenant de la classification par densité et des projections aléatoires. Nous démontrons que notre méthode présente de bons résultats pour des données de très grande dimension (de l'ordre de 50 à 100 dimensions) réparties dans des classes de forme aléatoire tout en respectant une complexité de $O(N\log(N))$, N représentant le nombre de données.

Une deuxième méthode pour gérer la base de documents Mpeg-7 est d'utiliser la structure XML des documents pour les classifier et ainsi accélérer les recherches pour accéder à l'information. Nos travaux dans ce domaine ne sont qu'à leur commencement, nous proposons une distance structurelle entre deux documents XML. Cette distance basée sur les ensembles de chemins peut être utilisée par une technique de classification pour regrouper les documents en plusieurs classes structurelles. Ces classes permettent de différencier dans notre cadre applicatif les documents Mpeg-7 par exemple selon le nombre de séquences décrites dans la vidéo, ou alors les séquences contenant un schéma de description de *lieu* ou de *personne*.

1.3 Plan de la thèse

Après cette introduction de ce mémoire de thèse, le chapitre 2 présente un état de l'art des méthodes existantes dans les domaines des structures d'indexation et de la classification.

Nous donnons tout d'abord quelques notions sur les distances et mesures de similarité utilisées dans ces domaines ainsi que quelques mesures de similarité entre documents XML. Ensuite nous détaillons les structures d'indexation multidimensionnelle en abordant le problème de la "malédiction des grandes dimensions". Nous détaillons les différentes solutions pour contourner ce problème. Dans une deuxième partie de l'état de l'art, nous rappelons quelques notions de classification non supervisée avant de nous intéresser plus précisément aux techniques de classification de sous espaces vectoriels répondant aux problèmes de classification des bases de données de dimension élevée.

Le chapitre 3 présente nos approches concernant les structures d'indexation multidimensionnelle. Nous proposons une première structure d'indexation multidimensionnelle, *Kpyr*, qui combine une méthode de classification à la technique de la pyramide. Nous détaillons ensuite les performances de *Kpyr* qui ont orienté nos recherches vers la création de *KpyrRec*, une structure d'indexation récursive. *KpyrRec* limite le nombre d'accès aux données lors d'une recherche ce qui améliore nettement les performances.

Dans le chapitre 4, nous présentons tout d'abord notre méthode de classification par projections aléatoires. Après avoir détaillé ses avantages par quelques expérimentations, nous proposons de l'utiliser dans *RPyR*, une structure d'indexation basée sur *KpyrRec*. Les avantages observés de notre méthode de classification procure à *RPyR* de bonnes performances quelque soit la base de données à indexer.

Dans le chapitre 5, nous proposons une distance structurelle entre deux documents XML avec quelques expériences sur des données synthétiques. Le chapitre 6 présente dans premier temps la transformation vectorielle que nous utilisons pour passer d'une base de documents Mpeg-7 à un ensemble de vecteurs multidimensionnels. Nous détaillons ensuite le moteur de recherche mis en place pour tester sur notre base de données réelles nos structures d'indexation. Enfin, nous proposons d'améliorer l'indexation et la recherche en utilisant les retours des utilisateurs.

Nous concluons ce mémoire de thèse au chapitre 7 en résumant les contributions scientifiques et les résultats obtenus. Nous présentons aussi des perspectives de recherche.

Bibliographie

[Dai] Dailymotion. http ://www.dailymotion.com/fr/.

[IMD] IMDB. http ://www.imdb.com/.

[Mpe] Mpeg7. http ://www.chiariglione.org/mpeg/.

[Tub] You Tube. http ://www.youtube.com/.

Chapitre 2

Etat de l'art

Sommaire

L'objectif de ce chapitre est d'aborder les problématiques liées à la prise en compte des données atypiques (données textuelles, multimédia, semi-structurées, multidimensionnelles) dans les bases de données. Ces données atypiques s'éloignent du modèle traditionnel supporté par les SGBD avec des requêtes précises, pour aller vers des besoins d'une exploitation fine et rapide de larges volumes de données à structure complexe ou inconnue.

Nous nous intéressons tout d'abord aux données XML et à quelques mesures de similarité pour des données structurées. Puis, nous donnons quelques notions sur les méthodes de recherche. Nous proposons ensuite un aperçu de méthodes de fouilles de données avant de nous focaliser sur les structures d'indexation multidimensionnelle. Nous finissons cette section par les méthodes de classification qui nous permettent d'affiner l'indexation et ainsi accélérer l'accès aux données lors des requêtes. Nous développons notamment les méthodes de classification de sous espaces vectoriels adaptées aux grandes bases de données de dimension élevée.

2.1 Introduction au langage XML

Le Web est devenu un outil d'échanges et de publications de données et d'informations. L'un des problèmes rencontré est la structure irrégulière des données. Les mêmes informations peuvent exister sous différents formats. Le Web avait donc besoin d'un modèle pour représenter des informations hétérogènes, partiellement structurées et évolutives. Le langage XML pour eXtensible Markup Language a donc été développé par le *World Wide Web Consortium* [W3C]. XML est devenu une norme incontournable d'échange de données sur Internet. Le langage XML est aussi un méta-langage car il contient les données et la description des données (meta données). Le langage XML répond aux caractéristiques requises : générique, puissant, ouvert, flexible, simple à comprendre et à manipuler.

La structure d'un document XML représente la définition de l'ensemble des balises autorisées dans le contenu et de l'ordre dans lequel elles doivent être utilisées. On définit alors les "DTDs" (Document Type Definition), le type et la classe d'un document. Pour écrire les documents XML, il est nécessaire de suivre certaines règles : respecter les noms de balise, toujours clore une balise ouverte, mettre les attributs entre guillemets, ne pas entrelacer les balises, etc. Un exemple de document XML accompagné de sa DTD est donné figures 2.1 et 2.2. Le do-

cument doit rester conforme à sa DTD (ne pas utiliser de noms de balises différents que ceux définis). Il est possible avec XML d'utiliser les DTDs existantes ou d'écrire ses propres DTDs. En résumé, les DTDs ont pour but d'écrire la structure et la grammaire d'un document XML qui lui sont associées. Un document XML est validé lorsqu'il respecte la structure d'une DTD associée.

```
<Video>
  <Title> videoTitle </Title>
  <Subtitles>
    <Subtitle>subtitle_1</Subtitle>
    <Subtitle>subtitle_2</Subtitle>
  </Subtitles>
  <Languages>
    <Language>language_1</Language>
  </Languages>
  <Actors>
    <Actor>
      <FirstName>fn_1</FirstName>
      <LastName>ln_1</LastName>
    </Actor>
  </Actors>
  <Type>action</Type>
  <Chapters>
    <Chapter>
      <Id>chapter_1</Id>
      <StartTime>10</StartTime>
      <Keywords>
        <Keyword>keyword_1</Keyword>
        <Keyword>keyword_2</Keyword>
      </Keywords>
    </Chapter>
    ...
</Video>
```

FIG. 2.1 – Exemple d'un document XML

```
<? xml version='1.0' encoding='UTF-8' ?>
<! ELEMENT Video (Chapters, Companies, Actors, Type, Language, Subtitles, Title) >
<! ELEMENT Title (#PCDATA) >
<! ELEMENT Subtitles (Subtitle)* >
<! ELEMENT Languages (Language)* >
<! ELEMENT Type (#PCDATA) >
<! ELEMENT Actors (Actor)* >
<! ELEMENT Companies (Company)* >
<! ELEMENT Chapters (Chapter)* >
<! ELEMENT Subtitle (#PCDATA) >
<! ELEMENT Language (#PCDATA) >
<! ELEMENT Actor (FirstName, LastName) >
<! ELEMENT LastName(#PCDATA) >
<! ELEMENT FirstName (#PCDATA) >
<! ELEMENT Company (#PCDATA) >
<! ELEMENT Chapter (Keywords, Images, Id, StartTime) >
<! ELEMENT StartTime (#PCDATA) >
<! ELEMENT Id (#PCDATA) >
<! ELEMENT Images (Image, Time)* >
<! ELEMENT Time (#PCDATA) >
<! ELEMENT Image (#PCDATA) >
<! ELEMENT Keywords (Keyword)* >
<! ELEMENT Keyword (#PCDATA) >
```

FIG. 2.2 – DTD du document XML exemple

Dans XML, on peut utiliser ce que l'on appelle "XML Schema", ce qui correspond à une DTD XML pour la définition de DTD, sachant que les DTDs XML sont définies comme des documents XML. Les avantages sont nombreux : on a un seul et même langage pour les documents et la définition de leurs DTDs. Les types de données de bases peuvent être réutilisés et enrichis. Ces DTDs sont importantes pour travailler le document lui-même, par exemple dans le cadre d'une transformation du document XML vers HTML. La connaissance de la structure associée au document XML permet de mieux construire la feuille de style générique et devient applicable à tous document XML vérifiant la même DTD.

2.2 Notions de distance et mesures de similarité

Pour faire la distinction entre deux objets, il est nécessaire d'avoir une mesure de similarité permettant de différencier les deux objets. Dès qu'une mesure de similarité a été définie pour une base de données, il est alors possible de les regrouper, de les ordonner, de faire des recherches par similarité. Dans cette section, notre but n'est pas de faire une liste exhaustive des mesures de similarité et dissimilarité existantes. Notre but est de donner les notions suffisantes pour comprendre l'ensemble de la thèse. Nous développons ensuite quelques notions de distance pour des données spatiales puis nous présentons quelques mesures de similarité utilisées pour les données structurées de type XML. Un très bon tutoriel sur les mesures de similarité et dissimilarité a été réalisé par Teknomo et est disponible en ligne (voir [Tek]).

2.2.1 Mesures et relations de similarité, dissimilarité et distance

Nous donnons ci dessous les définitions des mesures de similarité et dissimilarité que nous utiliserons dans la suite de ce mémoire :

- une **mesure de similarité** sur un ensemble E est une fonction $sim : E \times E \longrightarrow \Re^+$ vérifiant les propriétés suivantes :
  1. sim est symétrique : $\forall (a,b) \in E \times E, sim(a,b) = sim(b,a)$;
  2. $\forall (a,b) \in E \times E, sim(a,a) = sim(b,b) > sim(a,b)$;
- une **mesure de dissimilarité** sur un ensemble E est une fonction $diss : E \times E \longrightarrow \Re^+$ vérifiant les propriétés suivantes :
  1. $diss$ est symétrique : $\forall (a,b) \in E \times E, diss(a,b) = diss(b,a)$;
  2. $\forall a \in E, diss(a,a) = 0$.

Une **distance** sur un ensemble E est une application $d : E \times E \longrightarrow \Re^+$ telle que :

1. $\forall x,y \in E : d(x,y) = d(y,x)$ [symétrie] ;
2. $\forall x,y \in E : d(x,y) = 0 \Leftrightarrow x = y$ [séparation] ;
3. $\forall x,y,z \in E : d(x,z) \leq d(x,y) + d(y,z)$ [inégalité triangulaire].

Notons qu'une dissimilarité sur un ensemble E est une distance (ou métrique) si elle satisfait la propriété de séparation et l'inégalité triangulaire.

Pour illustrer la définition de distance, nous présentons la distance de *Manhattan* dans un espace à deux dimensions. Soit la fonction $d_{man} : \Re^2 \rightarrow \Re_+$ telle que $d_{man}(a,b) = |x_b - x_a| + |y_b - y_a|$, d_{man} satisfait les propriétés suivantes :

- $d_{man}(a,a) = 0$ car $|x_a - x_a| = 0$ et $|y_a - y_a| = 0$.
- $d_{man}(a,b) = d_{man}(b,a)$ car $|x_b - x_a| = |x_a - x_b|$.
- $d_{man}(a,b) = 0 \Rightarrow a = b$ car

  $d_{man}(a,b) = 0$

  $\Rightarrow x_b - x_a = 0$ & $y_b - y_a = 0$

  $\Rightarrow x_b = x_a$ & $y_b = y_a$

  $\Rightarrow a = b$
- $d(a,b) \leq d(a,c) + d(c,b)$ car

  $d_{man}(a,b) \leq d(a,c) + d(c,b)$

  $\Rightarrow |x_b - x_a| \leq |y_c - y_a| + |y_b - y_c|$

  $\Rightarrow |x_b - x_a| \leq 2 \times x_c - (x_b + x_a) \; \forall \; x_c > x_b$ & $x_c > x_a$

  $\Rightarrow |x_b - x_a| + x_b + x_a \leq 2 \times x_c$

  $\Rightarrow x_b \leq x_c \; \forall \; x_b > x_a$

  $\Rightarrow x_a \leq x_c \; \forall \; x_b \leq x_a$

  Par symétrie sur la dimension y et sur les valeurs de x_c et y_c, nous pouvons prouver l'ensemble des cas. La distance de *manhattan* est donc bien une distance.

Nous pouvons citer par exemple la distance de *Minkowski* [Abd07] :

$$D_{Minkowski}(x,y) = (\sum_{j=1}^{L} |x_j - y_j|^r)^{1/r}. \tag{2.1}$$

La distance de *Minkowski* est une généralisation de la distance de Manhattan (pour r = 1) et de la distance Euclidienne (pour r = 2). La majorité des méthodes présentées dans les sections suivantes sont basées sur la distance Euclidienne dans des espaces vectoriels. Toutes ces méthodes respectent les trois propriétés d'une distance. Cependant, pour les données non structurées (texte) ou semi-structurées (type XML), les mesures de dissimilarité ne respectent

pas systématiquement les trois propriétés d'une distance. Des domaines de recherche tels que la bio informatique ou la fouille de données complexes utilisent souvent ces mesures de similarité. Nous donnons quelques exemples non exhaustifs sans rentrer dans le détail. Pour des données binaires, la mesure de la similarité peut se calculer par exemple avec les indices de simple concordance, de Jaccard (appelé aussi indice de communauté), de Hamming, etc.

A partir de ces notions de distance, nous définissons une **relation de similarité** *Sim* entre deux objets *a* et *b* par *a Sim b* si et seulement si $d(a,b) < \varepsilon$ où ε est le seuil de similarité et *d* est une distance.

La relation de Similarité $Sim : E \times E \longrightarrow Booléen$ respectent les propriétés suivantes :

1. elle est réflexive : *a Sim a* car $d(a,a) = 0 < \varepsilon$;
2. elle est symétrique : *a Sim b* alors *b Sim a* car $d(a,b) = d(b,a) < \varepsilon$;
3. elle est anti-transitive : *a Sim b* et *b Sim c* n'implique pas nécessairement *a Sim c*.

A partir de la relation de similarité *Sim* définie ci dessus, nous pouvons définir une **relation de dissimilarité** $Diss : E \times E \longrightarrow Booléen$ par *a Diss b* si et seulement si $d(a,b) > \varepsilon$ qui respectent les propriétés suivantes :

1. elle n'est pas réflexive : *a* ne peut pas être dissimilaire à lui même ;
2. elle est symétrique : *a Diss b* alors *b Diss a* car $d(a,b) = d(b,a) > \varepsilon$;
3. elle est anti-transitive : *a Diss b* et *b Diss c* n'implique pas nécessairement *a Diss c*.

2.2.2 Quelques mesures de similarité basées sur des documents XML

De nombreux travaux ont été menés pour comparer les documents XML. La plupart proposent des mesures (similarité ou dissimilarités) étant des adaptations de mesures existantes pour d'autres applications telles que la classification de données textuelles [WF74, UAH76, YL99, BHR00]. Le langage XML propose cependant des informations supplémentaires relatives à la structure des données qu'il est judicieux d'exploiter. Certaines mesures de similarité basées sur la sémantique, d'autres sur la structure des documents et certaines combinant les deux exploitent effectivement ces informations structurelles [CCS04, SPK07].

Les mesures de similarité dites « sémantiques » sont initialement basées sur le contenu des documents et ne tiennent pas compte de la structure des documents. Yang et al. présentent dans [YL99] un état de l'art des différentes méthodes de classification des documents textuels. Les documents XML étant enrichis par une structure contenant des informations sur les données, nous ne détaillons pas les mesures de similarité dites « sémantiques » qui, bien qu'applicables aux données XML, ne leur sont pas adaptées.

Les mesures de similarité basées sur la structure des documents utilisent les informations relatives à l'organisation hiérarchique des données apportée par l'arbre du document XML :

- l'ensemble des noeuds ;
- leur position dans l'arbre ;
- les relations de descendance et de fraternité ;
- les chemins complets de la racine aux feuilles de l'arbre ;
- les méta-informations apportées par les attributs des noeuds ;
- etc.

Une énumération regroupant l'ensemble de ces informations structurelles accompagnée d'exemples est présentée dans [ZLCZ02].

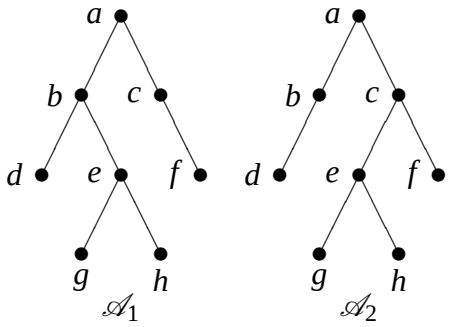

FIG. 2.3 – Exemples d'arbres

Deux des fonctions de comparaison basées sur la structure des documents les plus largement utilisées sont les mesures de similarité basées sur les arêtes et les chemins. Soient A et B

deux arbres XML, on note a_A et a_B l'ensemble de leurs arêtes ainsi que c_A et c_B l'ensemble de leurs chemins. Nous pouvons alors définir :

– la similarité basée sur les arêtes sim_a entre deux arbres XML A et B :

$$sim_a(A,B) = \frac{|a_A \cap a_B|}{|a_A \cup a_B|} \tag{2.2}$$

– la similarité basée sur les chemins sim_c de deux arbres XML A et B :

$$sim_c(A,B) = \frac{|c_A \cap c_B|}{|c_A \cup c_B|} \tag{2.3}$$

Si nous prenons l'exemple des arbres $\mathscr{A}_1$ et $\mathscr{A}_2$ de la figure 2.3, nous pouvons former les ensembles d'arêtes a et de chemins c suivants :

$a_{\mathscr{A}_1} = \{a \to b, b \to d, b \to e, e \to g, e \to h, a \to c, c \to f\}$;
$a_{\mathscr{A}_2} = \{a \to b, b \to d, c \to e, e \to g, e \to h, a \to c, c \to f\}$;
$c_{\mathscr{A}_1} = \{a \to b \to d, a \to b \to e \to g, a \to b \to e \to h, a \to c \to f\}$;
$c_{\mathscr{A}_2} = \{a \to b \to d, a \to c \to e \to g, a \to c \to e \to h, a \to c \to f\}$.

Nous en déduisons leurs valeurs de similarité : $sim_a(\mathscr{A}_1, \mathscr{A}_2) = \frac{6}{8}$ et $sim_c(\mathscr{A}_1, \mathscr{A}_2) = \frac{2}{6}$.

La mesure de similarité basée sur les chemins entiers est beaucoup plus restrictive que celle basée sur les arêtes. D'autres mesures de similarité mélangent ces deux mesures. Il est aussi possible d'enrichir ces mesures pour les optimiser en ajoutant par exemple des coefficients multiplicateurs en fonction du niveau de profondeur des noeuds ou des arêtes étudiés ou en prenant en considération des sous ensembles de chemins [ZLCZ02].

Une grande partie de la recherche sur les mesures de similarité entre deux documents XML s'est orientée vers l'algorithme du *Tree Edit* [ZS89], basé sur la distance *Edit*. Cette distance est apparue pour la première fois dans [WF74] pour comparer deux mots. Elle utilise trois opérations élémentaires :

– suppression d'un caractère
– ajout d'un caractère
– modification d'un caractère

La distance entre deux mots correspond au nombre minimal d'opérations élémentaires nécessaires pour transformer l'un des deux mots en l'autre. Par exemple, si l'on considère les mots

suivants *AGGCAD* et *GGCACD*, en ajoutant la lettre *A* en début du second mot et en supprimant la seconde occurrence de la lettre *C*, nous obtenons le premier mot. La distance entre ces deux mots est de deux opérations (une insertion et une suppression).

Cet algorithme a ensuite été appliqué aux arbres. C'est l'algorithme du *Tree Edit*. A partir des trois opérations élémentaires : suppression, insertion et modification d'un noeud de l'arbre, l'algorithme du Tree Edit donne le coût minimal pour transformer un arbre en un autre. Par exemple, pour transformer l'arbre A_1 en l'arbre A_2, nous avons la possibilité de supprimer les 3 noeuds *e*, *g* et *h* de A_1 et d'ajouter le noeud *e* sous le noeud *c* et les noeuds *g* et *h* sous le noeud *e* avec 3 insertions (figure 2.3). Le coût de cette transformation est de 6 opérations (3 suppressions, 3 insertions). Une autre possibilité est envisageable : modifier le noeud *c* en le renommant *b* et renommer *c* en *b* puis renommer *d* en *f* et *f* en *d* pour un coût de transformation égal à 4 (4 modifications). Finalement, la distance *Tree Edit* entre deux arbres correspond au coût minimal pour transformer un arbre en l'autre. Dans l'exemple précédent la distance entre A_1 et A_2 est donc égale à 4.

L'algorithme original de Zhang et al.[ZS89] donne un coût similaire à chaque opération, par la suite d'autres algorithmes ont pondéré le coût des différentes opérations de transformation, considérant que le coût d'une suppression ou d'une insertion n'est pas équivalent à celui d'une modification. D'autres ont pris en compte la profondeur des noeuds subissant une transformation dans l'arbre, supposant qu'une modification proche de la racine du document est plus importante qu'une modification proche des feuilles. Il a notamment été proposé dans [ZLCZ02] de prendre en compte l'ensemble des opérations élémentaires, la profondeur des noeuds, le nombre d'arêtes qui sont reliées au noeud modifié et enfin une interprétation sémantique des noeuds ; un noeud de même nom peut avoir différentes interprétations avec un noeud père différent.

Outre la distance *Tree Edit*, l'ensemble des mesures proposées ne sont pas des distances au sens mathématique. Il est périlleux de les utiliser dans une application telle qu'un regroupement. Celles-ci ne respectant pas forcément l'inégalité triangulaire, les résultats pourraient être fort différents de ce que l'on attend. Concernant le *Tree Edit*, son point faible est sa complexité élevée, de l'ordre de $O(n^3)$, n étant le nombre de noeuds composant les arbres. Dans de nombreux cas pratique, il n'est donc pas applicable car trop coûteux.

2.3 Méthodes de recherche

2.3.1 Recherche exacte

La recherche par similarité nécessite l'existence d'une mesure de similarité ou, encore mieux, d'une distance entre les données de la base. Dans les bases de données spatiales, la distance *Euclidienne* est souvent mise à l'honneur même s'il existe d'autres solutions. La recherche par similarité retourne un ensemble unique de données résultats, le plus similaire possible à une donnée requête, dans le sens de la mesure de similarité utilisée.

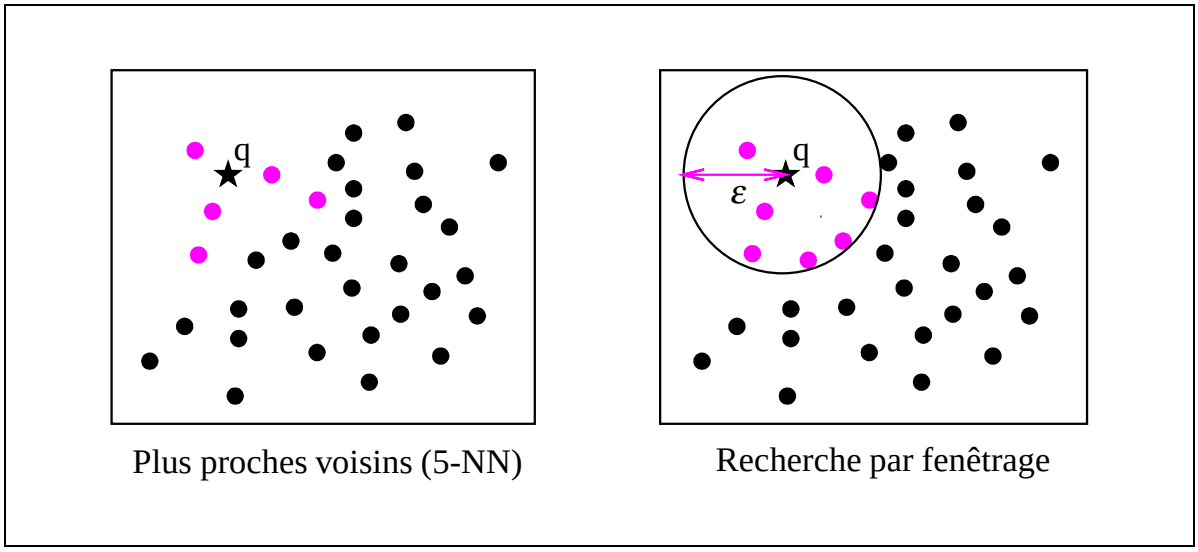

FIG. 2.4 – Recherche par similarité

Il existe différents types de requête par similarité. Une liste exhaustive est présentée dans [GG98]. Les deux types de recherche, illustrés par la figure 2.4 nous intéressent particulièrement :

1. **La recherche des k plus proches voisins** ("k-Nearest Neighbors" ou k-NN) permet de retourner un ensemble unique de k données résultats les plus proches de la donnée requête. Sur la figure, nous retournons les 5 plus proches voisins de la requête q.
2. **La recherche par fenêtrage** ou recherche à ε-près ("window query" ou "range query") permet de retourner toutes les données comprises dans une fenêtre à une distance infé-

rieure ou égale à ε. L'exemple présenté montre qu'il existe 7 résultats à une distance inférieure à ε.

La recherche des plus proches voisins assure un nombre précis de résultats même si ces derniers ne sont pas forcément tous pertinents. La recherche par fenêtrage retourne tous les résultats pertinents pour une requête à ε près mais il est possible de n'obtenir aucun résultat ou un ensemble de résultats bien trop volumineux.

Une approche différente pour la recherche des plus proches voisins est la recherche approximative. Soit un ensemble de vecteurs $\Omega = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, la recherche approximative ou à ε près des plus proches voisins est définie par : trouver un vecteur $\mathbf{x} \in \Omega$ tel que $\mathbf{x}$ est un plus proche voisin à ε près du vecteur requête $\mathbf{q}$ si $\forall \mathbf{x}' \in \Omega$, $d(\mathbf{x}, \mathbf{q}) \leq (1+\varepsilon)d(\mathbf{x}', \mathbf{q})$. La recherche approximative des plus proches voisins permet de faire un compromis entre la rapidité des résultats retournés et la qualité de ces derniers.

Nous verrons dans ce chapitre que les structures d'indexation existantes ne sont pas toujours adaptées aux différents types de recherche par similarité.

2.3.2 Recherche par similarité partielle

Nos travaux sont basés sur la recherche exacte et globale, cependant avec l'augmentation de la dimension, de nombreuses méthodes de recherche se sont orientées vers la recherche par similarité partielle. Tout d'abord, nous présentons un exemple qui permet de comprendre les motivations des recherches par similarité partielle avant de donner un aperçu de quelques méthodes de ce domaine. Soit le tableau 2.1 donnant les coordonnées de quatre vecteurs $\mathbf{x}_i$ et une requête $\mathbf{q}$.

Soient deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ dans $\mathbb{R}^d$, nous définissons la distance euclidienne globale :

$$DE_g(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^{d} |\mathbf{x}_j - \mathbf{y}_j|^2}. \tag{2.4}$$

Nous calculons la distance euclidienne globale DE_g entre chacun des vecteurs $\mathbf{x}_i$ et $\mathbf{q}$, nous obtenons les valeurs présentées dans le tableau 2.2 avec en dernière ligne le plus proche voisin

de la requête **q**, qui est le vecteur $\mathbf{x}_4$ pour DE_g. Le but de la recherche par similarité partielle est de proposer un ensemble de résultats dont la plupart des caractéristiques (dimensions du vecteurs) sont similaires à celles de la requête. Si on se base sur les 9 dimensions les plus proches de la requête pour chaque vecteur, le calcul de la distance euclidienne DE_9 propose un ordre des plus proches voisins différent de celui obtenu avec DE_g, et certains vecteurs se retrouvent très proches de la requête. Pour 8 dimensions, DE_8, $\mathbf{x}_3$ est égale à **q**.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
$\mathbf{x}_1$	1	1	1	1	1	100	1	1
$\mathbf{x}_2$	100	2	2	1	1	1	2	1
$\mathbf{x}_3$	1	10	2	10	1	2	100	100
$\mathbf{x}_4$	20	20	20	20	20	20	20	20
q	1	10	2	10	1	2	1	2

TAB. 2.1 – Base de données exemple

$DE_g(\mathbf{x}_i, \mathbf{q})$	$DE_9(\mathbf{x}_i, \mathbf{q})$	$DE_8(\mathbf{x}_i, \mathbf{q})$
98,8	12,8	9,1
99,7	12,1	8,2
139,3	98	0
47,5	43,5	39,1
$\mathbf{x}_4$	$\mathbf{x}_2$	$\mathbf{x}_3$

TAB. 2.2 – Distances euclidiennes

L'un des principaux inconvénients de ces méthodes est le choix du paramètre déterminant le nombre de dimensions à sélectionner. Par exemple, [GLC02] propose une mesure de similarité DPF (dynamic partial function) pondérée sur un ensemble de n dimensions. Leur mesure ne vérifie pas l'inégalité triangulaire, ce qui limite les possibilités d'utilisation. La recherche par similarité partielle peut être appliquée avec une méthode de classification par projeté dans des sous espaces que nous développons dans la section 2.6.3. La recherche de la classe la plus proche se limite aux caractéristiques (dimensions restreintes ou pondérées) du sous espace de chacune des classes. Plus récemment, [TZKO06], propose un algorithme *frequent k-n-match* de plus proches voisins qui se base sur la similarité partielle pour une fenêtre de dimensions. La structure d'indexation utilisée pour cette algorithme est un ensemble de d tableaux de couples (id, $coord$) triés pour chacune des d dimensions. Les k plus proches voisins les plus fréquents pour les valeurs de n dimensions choisies sont les résultats retournés pour une requête q. Néanmoins, leur algorithme est assez complexe et peut s'avérer lent si la fenêtre des dimensions choisies est mal adaptée aux données.

2.4 Quelques méthodes de fouilles de données vidéo

Nous présentons dans cette section un aperçu des différentes méthodes de fouille de vidéos. Ce domaine répond à de nombreuses applications dont les plus importantes sont :

- la gestion d'une base de données vidéo, ce qui inclut l'indexation, la classification et la recherche de vidéos ;
- la protection des vidéos contre le vol ou la copie.

L'accès aux contenus des vidéos peut être scindé en deux grandes familles : les contenus sémantiques et les contenus automatiques des vidéos. Les contenus dits "sémantiques" sont les descriptions d'une vidéo faite par un utilisateur, par exemple des mots clés, des résumés de séquence, etc. Les contenus dits "automatiques" se réfèrent à l'ensemble des descripteurs qui peuvent être extraits via la vidéo elle même. Beaucoup de ces descripteurs sont communs avec les descripteurs automatiques du contenu d'une image. De nombreux travaux de recherche ont abordé le problème de recherche par le contenu d'une image, plusieurs états de l'art de ces méthodes existent dont [RHC99, SWS+00, MF02]. Les vidéos apportent une dimension temporelle par rapport aux images. De nombreux descripteurs automatiques sont alors ajoutés tels que la hiérarchisation en séquences, plans et frames ou encore la notion de mouvement des objets ou de la caméra (voir [PPC06]). Les vidéos comme la musique ont une dimension sonore : par exemple, Zhang et al. ([ZDC06]) se sont basés sur les descripteurs automatiques audio pour mettre en valeur des séquences vidéo sur le sport. La majorité de ces descripteurs automatiques du contenu des vidéos sont inclus dans la norme Mpeg-7 [Mpe, Mar04], norme de descriptions des contenus multimédia et audiovisuel utilisant le langage XML. L'ensemble de ces descriptions est stocké dans une base de fichiers annexes aux vidéos. La fouille de données vidéos s'effectue alors dans cette base de documents Mpeg-7.

Nous présentons dans la suite de cette section quelques méthodes récentes d'extraction automatique de caractéristiques de séquences vidéo pour le regroupement, l'indexation et la recherche.

Une première approche intéressante est celle présentée par Kim et al. [KV05]. Cette approche est basée sur une combinaison pondérée par un paramètre de deux dissimilarités entre séquences vidéo : temporelle et spatiale. La dissimilarité spatiale utilisée découpe les images

de la vidéo en plusieurs parties et crée une matrice de rang de taille 4. La matrice de rang pour la représentation d'une séquence vidéo a été proposée par Mohan [Moh98], cette matrice est créée en découpant chaque image de la vidéo en parties égales, chaque partie reçoit une valeur en fonction de sa couleur moyenne ordonnée par rapport aux autres parties de l'image. La dissimilarité spatiale entre deux séquences vidéo se calcule à l'aide des différences entre l'ensemble des matrices de rang représentant les images d'une séquence vidéo. La dissimilarité temporelle est basée sur l'évolution des matrices de rang des paires d'images consécutives de la séquence vidéo.

Lee et al. [LOH05] ont proposé une approche se basant sur l'extraction d'objets et leurs caractéristiques temporelles pour indexer une base de données vidéo. Leur approche se décompose en deux grandes étapes : tout d'abord ils utilisent un algorithme de segmentation d'images pour construire un graphe de régions connectées. Ensuite, ils proposent de lier les graphes d'images consécutives d'une vidéo en connectant les noeuds similaires (les régions segmentées des images). L'indexation se fait ensuite par l'utilisation d'une technique de regroupement des graphes d'objets similaires, basée sur l'algorithme de classification Expectation Maximization EM. Une structure d'indexation est ensuite construite en fonction de la hiérarchisation des classes trouvées.

Zhu et al. [ZAF+03] présentent une indexation de documents vidéo basée sur un regroupement hiérarchique à plusieurs niveaux. Les images sont regroupées en plans à l'aide de la distance entre les histogrammes de couleurs de deux images consécutives. Les plans sont regroupés en groupes de plans qui sont eux aussi regroupés en classes en utilisant une mesure de similarité basée sur l'histogramme de couleur et les caractéristiques de textures des images. Les auteurs utilisent ensuite des algorithmes de détection de visage, de peau ou de couleur rouge sang pour déterminer des évènements particuliers (dialogue, opération chirurgicale) de leurs vidéos à caractère médical. Les auteurs proposent dans [ZEX+05] une amélioration de leur méthode en utilisant les caractéristiques de mouvement dans les séquences vidéo pour une meilleure détection des plans et pour l'extraction des images clés des séquences vidéo.

Cheung et al [CZ03] ont proposé une représentation des vidéos par signature appelé *ViSig*. L'idée de base est de résumer chaque vidéo par un petit ensemble d'images représentatives de la séquence vidéo, ensemble appelé *signature* de la vidéo. Ensuite, ils estiment le pourcentage

d'images similaires entre deux séquences vidéo en se basant seulement sur les images de la signature *Visig*. Sheung et al. [SOZ05] proposent une autre représentation des séquences vidéo qu'ils nomment *ViTri* (Video Triplet). Une séquence vidéo est représentée par un ensemble de classes qui contiennent des images similaires. Chaque classe est alors représentée par une hypersphère indexée par un triplet *Vitri*. Ensuite, ils proposent de gérer l'indexation des vidéos dans un arbre B+ (nous développons cette méthode dans la section 3). La mesure de similarité entre images n'est pas dans l'optique de leur recherche, leur but est d'optimiser la représentation des données vidéo et de les indexer. Dans [ZZS07], ils proposent une amélioration de leurs travaux antérieurs qui ne prenaient pas en compte la dimension temporelle apportée par la vidéo. L'idée principale est de décrire les vidéos de la base d'une manière plus compacte que par l'ensemble des caractéristiques de chaque image composant la vidéo. Pour cela, ils proposent de regrouper les images similaires d'une vidéo en plusieurs classes. Puis, la vidéo est reconstruite en remplaçant chaque image par l'identifiant de la classe de l'image. Ainsi, une vidéo est décrite par une chaîne ordonnée d'identifiants de classe, ce qui permet de garder la dimension temporelle.

Moënne-Loccoz et al. [MLBM05] utilisent les descripteurs dynamiques locaux des vidéos pour indexer leur moteur de recherche interactif. Les descripteurs dynamiques locaux permettent d'estimer les mouvements de caméras et les mouvements d'objets dans la séquence vidéo à partir des points caractéristiques des séquences d'images formant la vidéo. Ayache et al. [AGQ06] présentent leur système de détection de frontières de séquences vidéo basé sur les différences entre images voisines et les descripteurs de mouvement. Ils proposent aussi une catégorisation des séquences vidéo par machines à support vectoriel (SVM) à partir de descripteurs haut niveaux des séquences vidéo. Ces descripteurs hauts niveaux sont obtenus par la combinaison et la fusion de descripteurs bas niveaux (couleur, texture), des données audio, de données textuelles et de contextes conceptuels. Ils exploitent leur propositions dans un moteur de recherche vidéo par le contenu.

Très récemment, quelques laboratoires français ont lancé ARGOS [JBPKQ07, CIEI+07], une campagne d'évaluation d'outils de surveillance de contenus vidéo. L'objectif général est de comparer différents outils d'analyse du contenu vidéo. Ce projet se divise en deux phases. Une première phase concerne les techniques d'extraction des descripteurs bas niveaux des

vidéos, par exemple la segmentation en plans ou l'identification des mouvements de caméras. La deuxième phase concerne l'évaluation de méthodes basées sur la fusion ou la combinaison des résultats des méthodes de la première phase.

Pour compléter cet aperçu des différentes méthodes d'accès aux contenus vidéo pour l'indexation et la recherche, nous citons la thèse d'Alexis Joly [Jol06] qui présente ses recherches dans ce domaine avec une application particulière sur la détection de copies par signatures de vidéo. Un état de l'art résume l'ensemble des méthodes d'accès aux contenus vidéo qui ont un rapport avec la détection de copie. Il propose une mesure de similarité, pour détecter les copies de vidéos, basée sur les signatures locales des vidéos et une mesure globale sur l'ensemble de la base de données. Une méthode de recherche approximative par plus proches voisins est ensuite proposée pour les descripteurs numériques d'une vidéo basée sur des requêtes statistiques : les régions pertinentes de l'espace sont déterminées par un modèle de distorsion théorique.

Nous avons proposé dans cette section un aperçu de quelques méthodes récentes d'indexation et de recherche liées aux contenus de vidéo. La plupart de ces méthodes sont basées sur une représentation des vidéos comme une suite d'images contenant de nombreux descripteurs extraits automatiquement. Nos recherches dans ce mémoire n'utilisent pas cette représentation. Notre base de données contient des annotations manuelles pour chaque séquence vidéo, annotations réalisées par un expert métier selon un ensemble de descripteurs Mpeg-7 choisis. Une séquence vidéo est donc représentée par un document Mpeg-7 contenant des descripteurs sémantiques et automatiques.

2.5 Structure d'Indexation Multidimensionnelle

L'étude des structures d'indexation et les algorithmes qui les manipulent est un sujet fondamental pour l'indexation en général. Le but étant de mémoriser, accéder et manipuler les données sous une forme ou une autre. Selon le type de données utilisées, une méthode peut être plus appropriée qu'une autre. Il est donc important de bien choisir la structure de données qui apportera les meilleurs résultats. Nous nous intéressons particulièrement aux bases de données spatiales ainsi qu'aux différentes méthodes de recherche par similarité utilisées. Nous détaillons ensuite les structures existantes d'indexation uni-dimensionnelle et enfin nous

développons les structures d'indexation multidimensionnelle.

2.5.1 Indexation unidimensionnelle

Nous présentons dans cette section, les structures d'indexation unidimensionnelle fortement utilisées dans de nombreuses applications pour leur simplicité d'implémentation et leur efficacité dans l'accès aux données. Tout d'abord nous présentons rapidement les tables de hachage unidimensionnelle puis nous développons l'indexation par structures arborescentes.

2.5.1.1 Hachage unidimensionnel

Une fonction de hachage permet de convertir une donnée en une clé ou un index. Soit la fonction $h : \Omega \rightarrow \Phi$, $\forall \mathbf{x} \in \Omega$ on définit $h(\mathbf{x})$ comme la valeur de hachage de $\mathbf{x}$. Une fonction de hachage n'est pas injective. Une propriété d'une fonction de hachage est : "Étant donnés $\mathbf{x}, \mathbf{y} \in \Omega$, si $h(\mathbf{x}) \neq h(\mathbf{y}) \Rightarrow \mathbf{x} \neq \mathbf{y}$". Lorsque deux données ont la même valeur de hachage, on parle de *collision*. Une table de hachage est une structure de données qui utilise une fonction de hachage pour indexer les données dans un tableau. On accède à une donnée via son index de hachage en un temps de $O(1)$ en général et $O(n)$ dans le pire des cas, i.e. toutes les données ont la même valeur de hachage. Les tables de hachage sont très rapides mais elles ne sont pas toujours adaptées à la recherche souhaitée. Dans la section 2.5.2.4, nous présentons des structures de hachage multidimensionnel.

2.5.1.2 Indexation arborescente

Au milieu des années 70, apparaît une méthode de structuration pour le stockage et l'accès aux données, le Balanced-Tree ou Bayer Tree (B-Tree) [BM72]. C'est une structure linéaire qui repose sur l'existence d'un ordre total sur le domaine de la clé représentant une donnée. Il a pour avantage une extension facile du système pour un accès efficace aux données spatiales et un accès possible à travers le langage de requêtes. La figure 2.5 nous montre un arbre B d'ordre 2 où chaque noeud sauf la racine contient k clés avec $2 \leq k \leq 4$ et la racine contient $1 \leq k \leq 4$ clés. Il est important de noter qu'un arbre B est équilibré, toutes les feuilles sont au même niveau.

L'indexation d'un arbre B consiste à diviser les possibilités de choix en plusieurs parties égales et établir des pointeurs vers les sous-blocs. De cette façon, pour rechercher un élément, on le compare d'abord avec l'élément de la tête de l'arbre. Si l'élément recherché est celui-là, la recherche est terminée, sinon, soit elle est plus grande, soit plus petite ; dans l'un ou l'autre cas on continue la recherche dans le sous-bloc correspondant en utilisant le pointeur approprié.

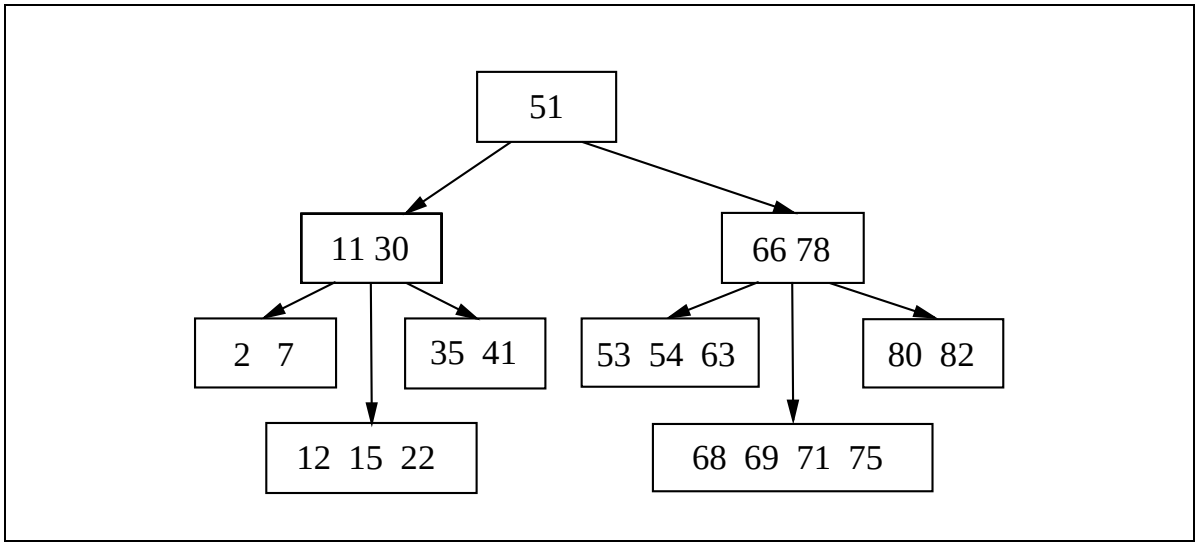

FIG. 2.5 – Arbre B

L'Arbre B+ est la structure d'indexation la plus répandue, c'est une amélioration de l'arbre B. Elle est dynamique, procure un ajustement de la structure balancée de l'index à chaque insertion/suppression. Cette méthode permet une recherche efficace par intervalle de valeur ou sur une condition d'inégalité. Un arbre B+ comprend un index organisé en arbre B et un ensemble de feuilles contenant les clés. La recherche se fait donc jusqu'aux feuilles. En général, on a un taux de remplissage moyen des noeuds de 67%. La structure d'arbre est idéale pour une recherche par intervalles, comme pour une recherche par valeurs. Les inconvénients d'un arbre B+ sont la nécessité de traverser l'index en profondeur pour un accès sélectif et une mise à jour coûteuse pour l'éclatement ou la fusion des noeuds. Nous présentons dans la figure 2.6, un exemple d'arbre B+ créé avec les mêmes index que l'arbre B présenté au dessus. Il est important de noter que la recherche d'un élément dans un arbre B+ est l'une des plus rapides. Elle est de l'ordre de $O(\log N)$, N étant le nombre d'index.

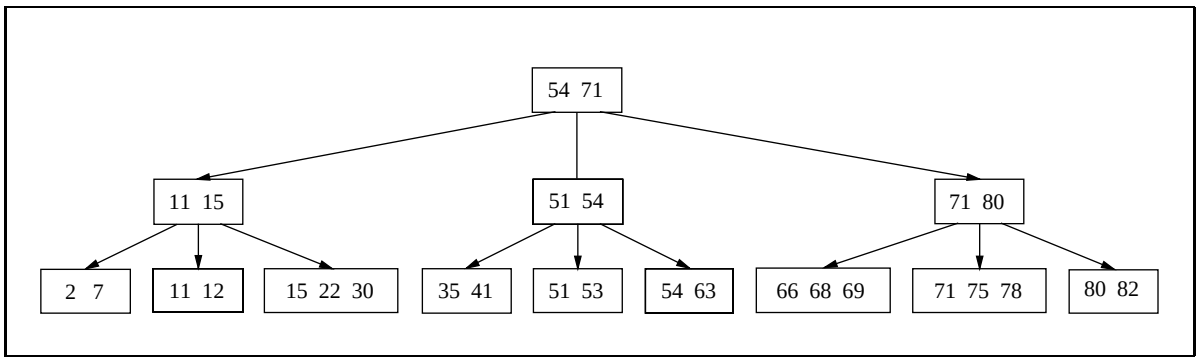

FIG. 2.6 – Arbre B+

Les arbres B+ sont aujourd'hui encore très souvent utilisés. La longévité de cette méthode d'indexation à une dimension est l'une des meilleures preuves de son efficacité. L'étude de [OT02] nous montre les différentes formes ainsi que les domaines d'applications de l'arbre B ou B+. Mais pour certains types de données, l'indexation par un arbre de la famille de l'arbre B ne suffit pas. On utilise alors des structures d'indexation multidimensionnelle.

2.5.2 Indexation multidimensionnelle

L'indexation spatiale a de multiples applications, son but principal est d'accélérer l'accès à une grande collection de données par leur position dans un espace multidimensionnel. Les avantages d'un index spatial sont l'efficacité en temps et en espace pour la recherche d'objets et la mise à jour dynamique et efficace. Les données peuvent être placées sans organisation particulière autre que leur stockage par page de mémoire. Comme le montre la figure 2.7, une indexation multidimensionnelle permet une recherche par similarité à partir de documents complexes.

Les premiers travaux de recherche sur les structures d'indexation spatiales dans un espace à plusieurs dimensions ont été réalisés entre 1979 et 1984. En effet, durant ces années, deux méthodes vont constituer les bases de ce domaine : le *Kd-Tree* [Ben79] et le *R-Tree* [Gut84]. Ces deux méthodes sont à l'origine des deux grandes familles de l'indexation multidimensionnelle : celle basée sur le partitionnement des données et celle basée sur le partitionnement de l'espace.

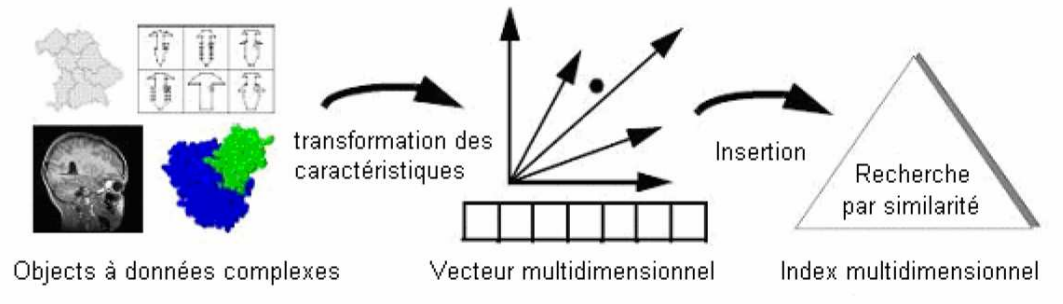

FIG. 2.7 – Principe général de l'indexation multidimensionnelle et de la recherche par similarité

2.5.2.1 Indexation par partitionnement des données

Ces techniques dérivent toutes du R-Tree [Gut84]. Elles procèdent par regroupement des vecteurs selon leur proximité dans l'espace. Chaque groupe de vecteurs est englobé dans une forme géométrique simple à manipuler (hyper-sphère, hyper-rectangle,etc.). Le tout est organisé sous forme d'un arbre dans lequel les vecteurs sont stockés dans les feuilles alors que les formes englobantes sont stockées dans les noeuds internes. On trouve dans cette famille des méthodes comme le R*-Tree [BKSS90], le X-Tree [BKK96], le SS-Tree [WJ96], le SR-Tree [KS97] et le M-Tree [CPZ97]. Nous en détaillerons seulement quelques unes.

La famille du *R-Tree* :

Les arbres de la famille du R-Tree indexent un espace multidimensionnel de points par un découpage hiérarchique équilibré en hyper-rectangles. Le R-Tree [Gut84] est l'extension directe du B-Tree [BM72] aux espaces multidimensionnels. Le R-Tree, voir figure 2.8, est un arbre équilibré dans lequel chaque noeud n est associé à un rectangle englobant minimum (REM), ce dernier est le REM de tous les rectangles de ses fils. Les feuilles de l'arbre contiennent une liste d'entrées de type (REM, oid), où REM est le rectangle minimum englobant de l'objet identifié par son oid

Le *R*-Tree* [BKSS90] est une variante du R-Tree. Les algorithmes d'insertion et de découpage ont évolué. Au lieu de découper un noeud surchargé, les entrées supplémentaires sont

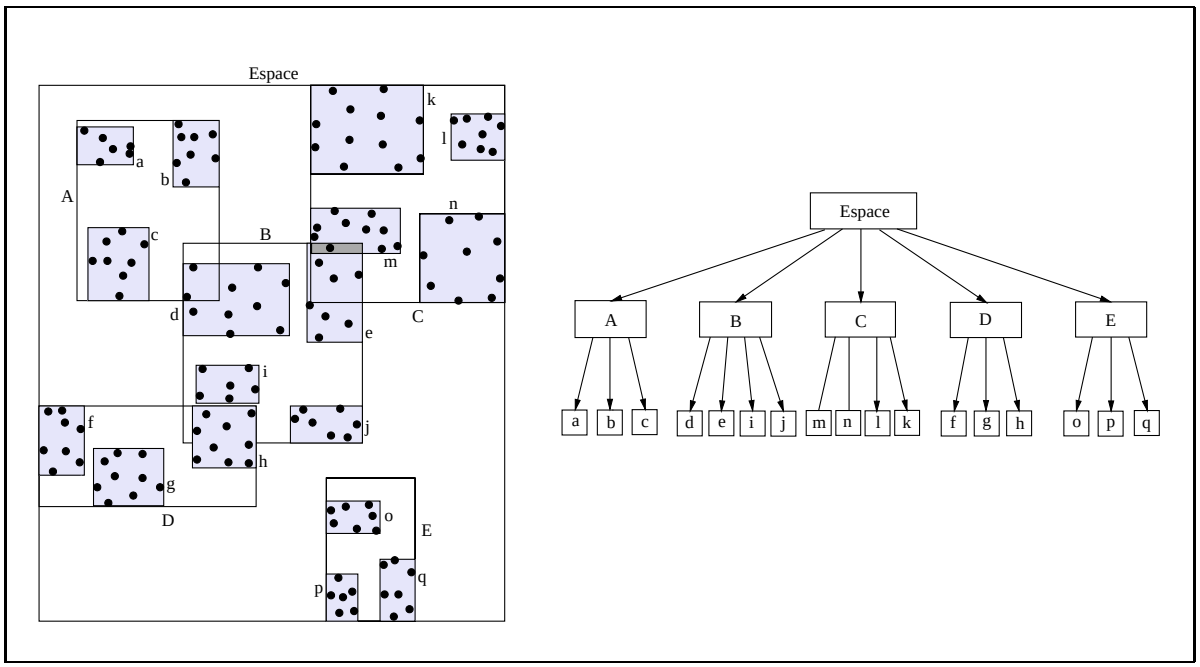

FIG. 2.8 – Structure du R-Tree

retirées et réinsérées dans l'arbre au même niveau. On a donc une réorganisation dynamique de l'arbre. Les arbres R*-Tree présentent des objectifs de dédoublement différents : le chevauchement entre les régions de noeuds, la longueur du périmètre des régions et le volume couvert par les noeuds doivent être réduits au maximum.

En Résumé, la famille des techniques d'indexation multidimensionnelle dérivées du R-Tree possède les propriétés suivantes :

- les cellules sont des hyper-rectangles englobants minimaux (REMs) ; elles sont organisées en hiérarchie où un niveau supérieur englobe les REMs du niveau inférieur ;
- les REMs permettent d'optimiser les règles de filtrage par l'utilisation d'une distance précise ;
- les formes englobantes se recouvrent, ce qui diminue l'efficacité des procédures de recherche ; ce problème s'amplifie lorsque la dimension des données croît ;
- les REMs se représentent par deux points multidimensionnels, ce qui pose des problèmes sur la quantité d'éléments que l'on peut stocker lorsque que la dimension augmente, ce

qui peut entraîner la construction d'arbres de grande hauteur ;

– L'algorithme de construction du R-Tree par insertions successives est non déterministe : le même ensemble de données ne sera pas représenté par le même arbre selon l'ordre d'insertion dans l'arbre.

Le X-Tree [BKK96] est une variante du R*-Tree. Le X-Tree, voir figure 2.9, gère le problème de recouvrement des REMs en utilisant deux concepts : l' "overlap-free split" (découpage à recouvrement libre) et le concept de super noeuds avec une capacité agrandie. Le X-Tree minimise ainsi les recouvrements entre REMs, ce qui, par conséquence, améliore la recherche d'un élément. Le X-Tree utilise un super noeud pour que l'arbre reste équilibré après l'algorithme de "split". Les performances de l'arbre X-Tree se sont révélées meilleures que celles du R*-Tree lorsque la dimension augmente mais moins bonnes que la recherche séquentielle si la dimension devient trop grande [GG98].

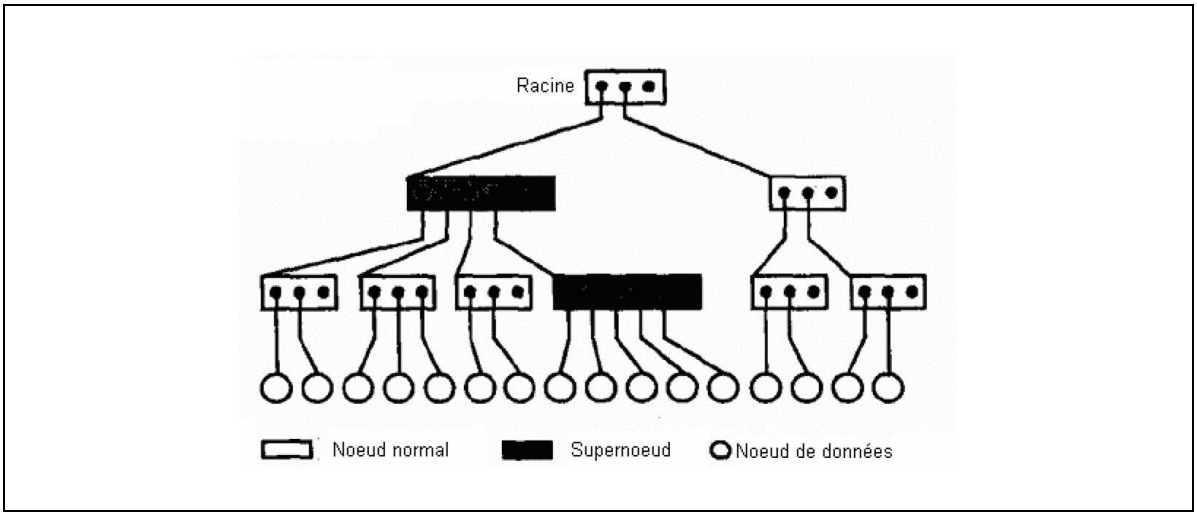

FIG. 2.9 – Structure du X-Tree

Le *SS-Tree* [WJ96], figure 2.10 est une amélioration du R*-tree qui fonctionne mieux pour la recherche des plus proches voisins. Cette structure emploie des frontières sphériques : les centres des sphères englobantes sont utilisés dans la construction du SS-Tree. Cette méthode requiert moitié moins de place de stockage qu'une méthode utilisant les REMs et utilise une réorganisation dynamique lors de l'insertion de données.

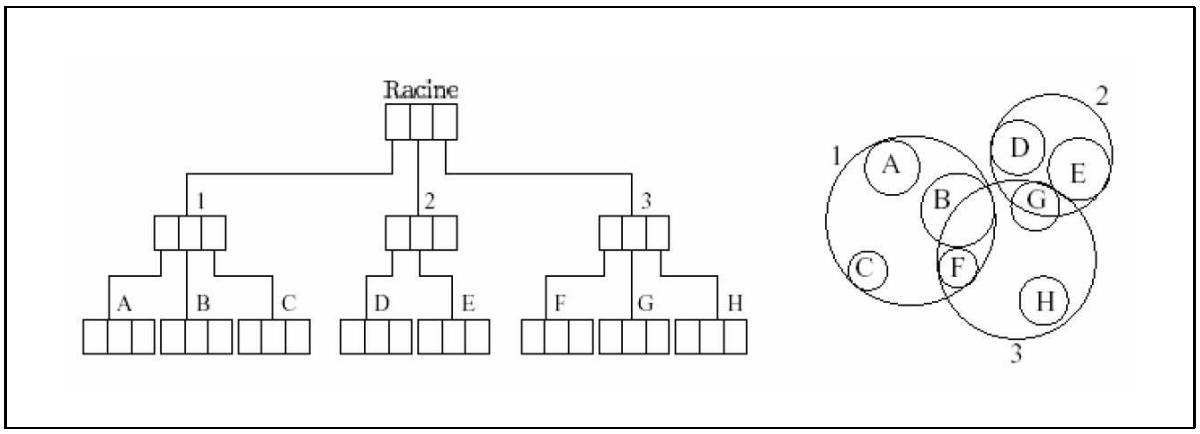

FIG. 2.10 – Structure du SS-Tree

Le *SR-Tree* [KS97] est un mélange du SS-Tree et du R*-Tree. Les sphères sont performantes dans les algorithmes des plus proches voisins et des requêtes d'approximation car elles utilisent une distance euclidienne. En contrepartie, elles créent des régions de recouvrement plus grandes que les rectangles. Le SR-Tree, comme le montre la figure 8, combine des sphères et des rectangles englobants minimums pour récupérer les avantages des deux méthodes précédentes. L'un des désavantages de l'arbre SR-Tree est qu'il a l'une des descriptions de régions les plus complexes mixant les REMs et les SEMs (Sphères englobantes minimales) car la dimension des index n'est pas la même ($2d$ pour les REMs et $d+1$ pour les SEMs). Les performances du SR-Tree sont équivalentes à celles du X-Tree.

2.5.2.2 Indexation par partitionnement de l'espace

Contrairement aux techniques présentées dans la section précédente dont le principal inconvénient est le chevauchement entre les cellules, nous allons présenter dans cette section d'autres méthodes qui se basent sur le principe de partitionnement de l'espace. Ces techniques ont l'avantage d'être simples à gérer et aucun chevauchement n'existe entre les cellules qu'elles manipulent.

Elle reposent sur le principe qui consiste à partitionner à priori l'espace en cellules (régions ou cases) plus ou moins régulières sans prendre en compte la distribution des données. Ceci permet d'avoir des structures d'indexation faciles à construire et simples à gérer. De plus le

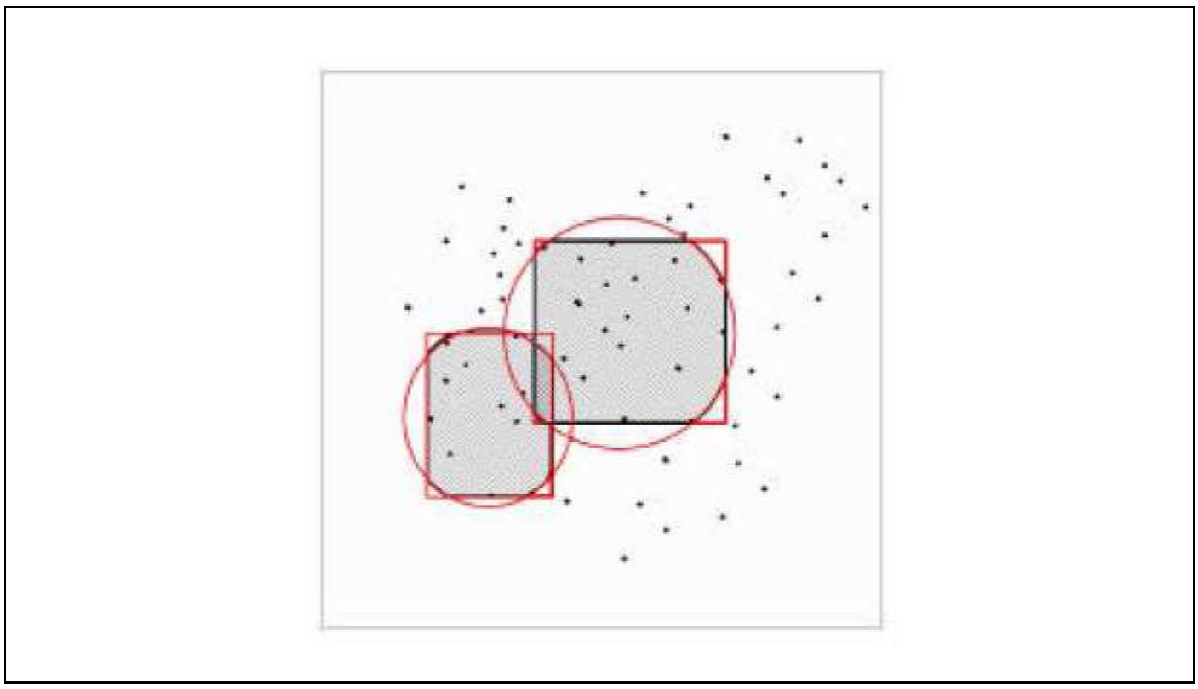

FIG. 2.11 – Structure du SR-Tree

chevauchement entre les cellules est évité. Ces techniques sont toutes dérivées du k-d-Tree [Ben79].

Le KDB-Tree [Rob81] est une structure d'indexation qui se base sur le partitionnement de l'espace en régions organisées dans une arborescence similaire au R-Tree. Comme le montre la figure 2.12, les noeuds et les feuilles correspondent à des régions de l'espace. Les régions d'un même niveau sont toutes disjointes. Les régions des noeuds fils regroupées forment la région du noeud père.

L'absence de chevauchement accroît les performances des algorithmes de recherche. Cependant, si un point requête est proche d'une frontière, il est nécessaire de visiter les régions voisines qui peuvent être nombreuses. Le partitionnement de l'espace n'étant pas en fonction de la répartition des données, le KDB-Tree ne garantit pas un taux minimal d'utilisation de l'espace alloué. De plus le respect d'un arbre équilibré peut créer des noeuds ou feuilles vides.

Le LSD-Tree [HSW89] et le LSDh-Tree [Hen98] sont deux structures d'indexation multidimensionnelle, le LSDh-Tree, voir figure 2.13, étant une amélioration du LSD-Tree. Le

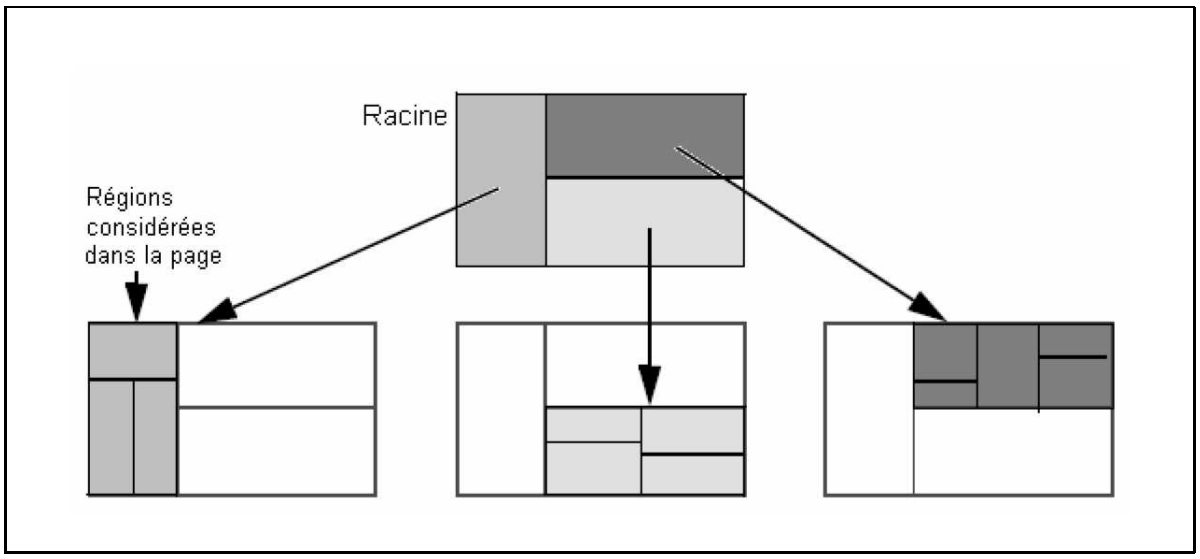

FIG. 2.12 – Partitionnement du KDB-Tree

LSDh-Tree est une variante du KDB-Tree : La description des régions est plus complexe et aboutit à une réduction du besoin d'espace pour ces descriptions. Un codage binaire des régions de données permet de réduire la taille de l'arbre et évite de gérer les zones vides de l'espace.

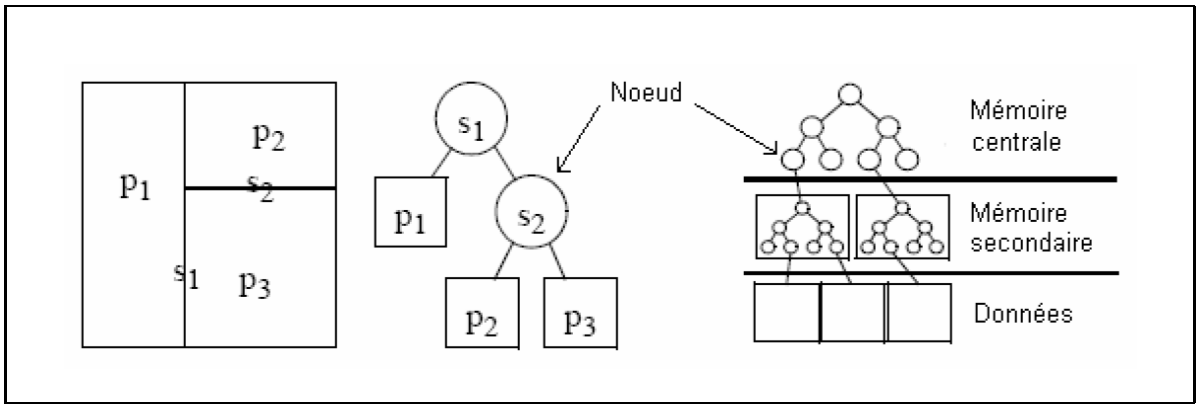

FIG. 2.13 – LSDh-Tree

Certains niveaux en haut de l'arbre sont stockés dans la mémoire principale dans des répertoires dits internes à l'opposé des répertoires externes pour les autres niveaux sujets à des mis

à jour fréquentes. Pour une dimension 16 par exemple, cette méthode requiert 10 fois moins de place que le R-Tree. Pour l'insertion d'un point, il existe toujours une unique région de données. Les performances de cet arbre sont assez similaires à celles du X Tree au final.

Synthèse :

Ci-dessous, le tableau 2.3 récapitule les avantages et inconvénients des différentes méthodes présentées précédemment. Un état de l'art complet sur ces structures a été fait par Gaede et al.[GG98] :

Méthode	Avantages	Inconvénients
R*-Tree	Formes englobantes permettant d'affiner le filtrage	Fort taux de chevauchement ; éclatement des noeuds problématiques
X-Tree	Limite le chevauchement entre régions	Paramètres de construction difficile à fixer : seuil max de chevauchement
SS-Tree	Petites tailles d'index	Chevauchement important
SR-Tree	Formes englobantes plus adaptées à de grandes dimensions	Complexité des formes ; recherche coûteuse ; taille d'index importante
Kdb-Tree	Pas de chevauchement	Fractionnements récursifs coûteux et arbitraires ; faible taux d'utilisation de l'espace alloué
LSDh-Tree	Codage binaire des régions : taille de l'arbre réduite et zones vides non gérées	Organisation mémoire complexe

TAB. 2.3 – Résumé des avantages et inconvénients des index multidimensionnels cités [Ber04]

2.5.2.3 Malédiction de la dimension

Pourtant, la plupart de ces méthodes subissent ce qui est communément appelé la malédiction de la dimension ("The curse of dimensionality") : plusieurs travaux, par exemple [WSB98] ainsi que la thèse de Berrani [Ber04], ont étudié les difficultés rencontrées pour des données

de dimension élevée. Pour une dimension supérieure à 16, il est souvent plus rapide et efficace d'utiliser un algorithme de recherche séquentiel que l'une des méthodes d'indexation multidimensionnelle présentées.

Les effets d'une dimension élevée sont nombreux, on a par exemple une augmentation exponentielle des volumes englobants et par conséquence des volumes de recouvrement ; les algorithmes de découpage de l'espace deviennent complexes car de nombreuses possibilités de découpage existent, etc. La thèse de Berrani présente en détails les phénomènes observés pour des données de dimension élevée. L'un de ces phénomènes va nous intéresser particulièrement : l'évolution de la géométrie usuelle en grande dimension.

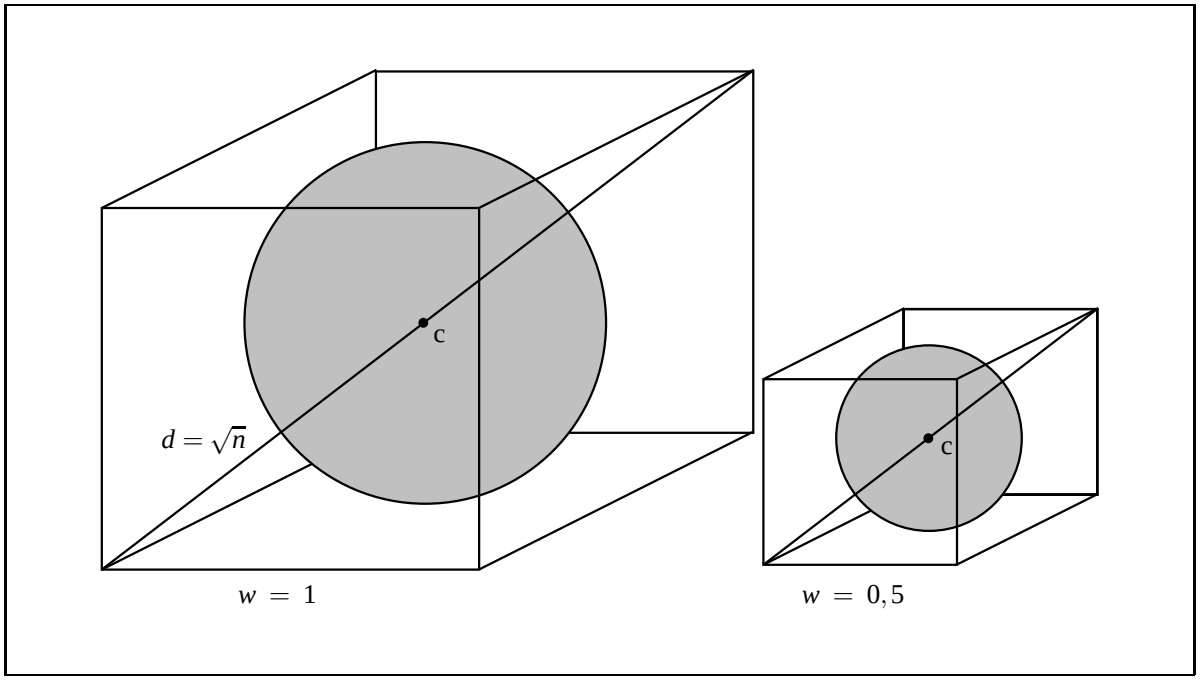

FIG. 2.14 – Hypercubes exemples

Prenons par exemple deux hyper cubes HC_1 et $HC_{0,5}$, l'un normalisé ($w = 1$) et l'autre de côté $w = 0,5$ (voir figure 2.14). Le tableau 2.4 présente quelques exemples de valeurs pour les volumes des hypercubes, la longueur de leur diagonale, ainsi que le volume de l'hypersphère inscrite dans chaque hyper-cube. Les formules mathématiques générales sont données ci dessous pour un espace de dimension d :

– volume d'un hypercube de largeur w :

$$V(HC_w) = w^d \tag{2.5}$$

– diagonale d'un hypercube de largeur w :

$$Dg(HC_w) = w \times \sqrt{d} \tag{2.6}$$

– volume d'une hyper-sphère inscrite dans un hypercube de largeur w :

$$V(HS_w) = \frac{\pi^{\frac{d}{2}}(w/2)^d}{\Gamma(\frac{d}{2}+1)} \tag{2.7}$$

où la fonction gamma est égale à $\Gamma\left(\frac{d}{2}+1\right) = \sqrt{\pi}\frac{d!!}{2^{(d+1)/2}}$, $d!!$ est le double factoriel de d.

Les valeurs exemples du tableau montrent quelques phénomènes intéressants. Tout d'abord, on observe que la limite du volume d'une hyper-sphère lorsque le nombre de dimensions tend vers l'infini est nulle. Si nous prenons un espace où les données ont une répartition homogène dans l'espace, l'observation précédente implique que la grande majorité des données se situent à l'extérieur de la sphère et donc proche des hyper-faces de l'hypercube. Nous remarquons aussi que la valeur de la diagonale nous donne la distance Euclidienne maximale entre deux points inclus dans l'hypercube. Cette valeur montre bien la différence de distance qu'il peut exister entre deux points dans un hypercube et deux points dans une hyper-sphère, phénomène que nous retrouvons dans les volumes de recouvrement des méthodes telles que la famille du R-Tree, incluant le SS-Tree et SR-Tree.

Nous présentons, dans les sections suivantes, quelques nouvelles techniques qui contrairement aux index multidimensionnels traditionnels décrits précédemment, gèrent les problèmes liés aux grandes dimensions. Ainsi toutes ces techniques proposent des idées originales leur permettant de contrecarrer à la malédiction de la dimension.

d	$V(HC_1)$	$V(HS_1)$	$Dg(HC_1)$	$V(HC_{0,5})$	$V(HS_{0,5})$	$Dg(HC_{0,5})$
2	1	π	$\sqrt{2} \approx 1,41$	0,25	$\approx 0,39$	$\approx 0,71$
3	1	$\frac{4\pi}{3} \approx 4,19$	$\sqrt{3} \approx 1,73$	0,125	$\approx 0,52$	$\approx 0,87$
4	1	$\approx 4,93$	2	0,0625	$\approx 0,31$	1
10	1	$\approx 2,55$	$\sqrt{10} \approx 3,16$	$\approx 0,001$	≈ 0	$\approx 1,58$
20	1	$\approx 0,01$	$\sqrt{20} \approx 4,47$	$\approx 10^{-7}$	≈ 0	$\approx 2,24$
100	1	≈ 0	10	$\approx 10^{-31}$	≈ 0	5

TAB. 2.4 – Evolution de la géométrie des hypercubes et hypersphères selon la dimension d

2.5.2.4 Techniques basées sur un hachage multidimensionnel

Le principe des techniques de hachage multidimensionnel est de créer plusieurs fonctions de hachage représentant au mieux la répartition des données dans l'espace. Ainsi les données ayant les mêmes valeurs de hachage ont une très forte probabilité d'être similaires et les données éloignées ont une forte probabilité de ne pas avoir la même valeur de hachage. Soit une collection Ω de données et une fonction de similarité $sim(\mathbf{x},\mathbf{y})$, on définit une famille Φ de fonctions de hachage sensibles localement sur Ω telle que $\forall \mathbf{x},\mathbf{y} \in \Omega$, $\forall r_1, r_2 \in \mathbb{R}^n, r_1 < r_2$ et $\forall P_1, P_2 \in [0,1], P_1 > P_2$ si

- $d(\mathbf{x},\mathbf{y}) \leq r_1 \rightarrow \text{Proba}(\text{h}(\mathbf{x}) = \text{h}(\mathbf{y})) > \text{P}_1$
- $d(\mathbf{x},\mathbf{y}) > r_2 \rightarrow \text{Proba}(\text{h}(\mathbf{x}) = \text{h}(\mathbf{y})) \leq \text{P}_2$

Indyk et al. [IM98] sont les premiers à proposer LSH (pour Locality Sensitive Hashing), une méthode de hachage multidimensionnel. Une amélioration a ensuite été proposée dans [GIM99]. Leur méthode se base sur la distance de Hamming qui transforme les coordonnées de chaque vecteur en suite de valeurs binaires. Les fonctions de hachage proposées concatènent les coordonnées binaires dans un sous ensemble de dimensions propres à la fonction de hachage. Une amélioration à LSH a été proposée par Datar et al. [DIIM04], utilisant une loi de distribution p-stable pour définir les intervalles de hachage, par exemple la loi de distribution Gaussienne définie par la fonction de densité $g(\mathbf{x}) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ est 2-stable. La famille de fonctions de hachage est alors définie par $h_{\mathbf{a},b}(\mathbf{v}) = [\frac{\mathbf{a}.\mathbf{v}+b}{w}]$, où $\mathbf{a}$ est un vecteur aléatoire et b un nombre réel compris entre $[0,w]$. Une comparaison des temps de hachage et temps

de recherche, ainsi que l'implémentation des derniers algorithmes utilisant les fonctions LSH sont disponibles sur [AI].

L'un des inconvénients principaux du LSH est le nombre de tables de hachage requis, ce qui demande un espace mémoire important. Pour de grandes bases de données, la technique de LSH n'est pas envisageable pour récupérer la plupart des plus proches voisins selon Lv et al.[LJW+06]. Ces derniers proposent de perturber les valeurs de hachage de la requête pour récupérer plusieurs ensembles de données par table de hachage. Selon leurs expérimentations, cette technique permet de réduire le temps d'une requête et le nombre de tables à garder en mémoire pour obtenir des résultats similaires à ceux des techniques LSH.

2.5.2.5 Techniques basées sur une approximation de l'espace

Le VA-File [WSB98] est une méthode basée sur la compression des données, mais elle est aussi considérée comme une amélioration de la recherche séquentielle. Le principe du VA-File repose sur la gestion en deux ensembles de données : un fichier qui contient tous les vecteurs de la base, et un autre qui contient des approximations géométriques de ces vecteurs, une technique de codage simple à réaliser, voir figure 2.15.

Une approximation du vecteur requête est calculée lors de la requête. Pour une recherche exacte, il suffit de lire l'ensemble des requêtes ayant le même code d'approximation. Pour une requête par similarité les plus proches voisins sont directement donnés avec cette technique de codage. Pour agrandir cette recherche, il suffit de prendre en compte les approximations voisines de celles de la requête. Cette technique est particulièrement adaptée aux données distribuées uniformément dans l'espace. Les résultats présentés par la méthode du VA-File [WSB98] prouvent son efficacité dans les grandes dimensions.

Le LPC-File [CZPC02] est une amélioration du VA-File. Cette méthode propose d'enrichir les informations codées en introduisant les coordonnées polaires dans le calcul des approximations. Cette information supplémentaire permet d'améliorer le taux de filtrage des vecteurs non pertinents lors de la recherche. L'inconvénient de cette méthode est le calcul très coûteux des distances entre la requête et les approximations de données. Les résultats présentés sont meilleurs à ceux du VA-File. Berchtold et al. [BBJ+00] ont proposé le IQ-Tree pour *Independent Quantization Tree*. Le IQ Tree est constitué de trois niveaux d'indexation : un premier

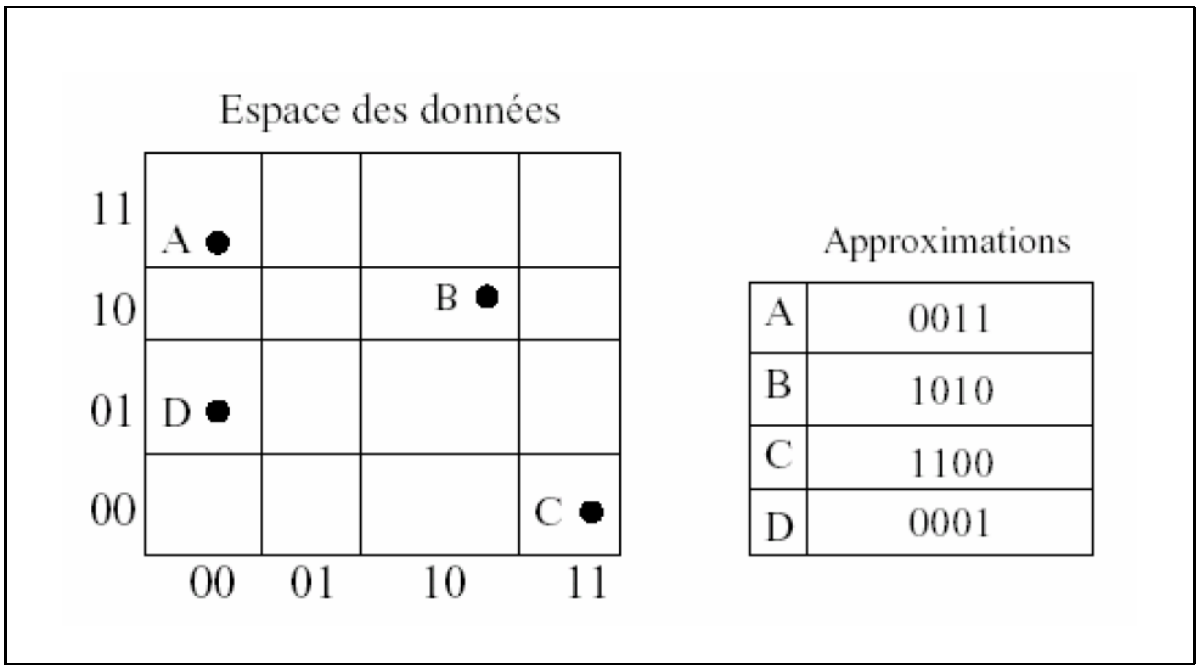

FIG. 2.15 – Technique de codage du VA-File

niveau répartit les données dans une structure hiérarchique de type R*-Tree, le second niveau conserve une approximation du style VA-File pour chaque rectangle englobant minimum (REM), enfin le dernier niveau correspond aux données. Le nombre d'accès aux données et le temps de recherche sont considérablement réduits par rapport aux performances de VA-File. Très similaire du IQ-Tree, le A-Tree [SYUK00] propose une indexation hiérarchique similaire au R*-Tree. La différence se fait dans l'approximation du REM. Cette dernière est relative au REM père et est donc plus précise que sur l'espace entier comme le fait VA-File. Les expérimentations du A-Tree montrent que cette approximation relative permet de limiter le nombre d'accès et d'accélérer la recherche par rapport à VA-File.

Plus récemment, le GC-tree [CC02] propose de partitionner de manière hiérarchique l'espace en hyper-cubes selon la répartition des données. De plus, le GC Tree est construit dynamiquement : un noeud est divisé lorsque sa taille mémoire est supérieure à la capacité d'une page du disque. Un paramètre τ de densité est fixé par l'utilisateur pour déterminer si la région représentée par l'hyper-cube est une classe ou non. Les données non classées sont considérées

comme données "bruit" et sont stockées ensemble. Le GC-Tree utilise la même représentation des données que LPC-File dans ses noeuds feuille.

2.5.2.6 Techniques basées sur une découpage géométrique de l'espace

Parmi les nouvelles approches étudiées, la *Technique de la Pyramide* [BBK98] est celle qui nous a le plus intéressés. La technique de la Pyramide indexe les points d'un espace de dimension d avec une clé à une dimension, puis utilise un arbre B+ pour ranger ces clés. Cette méthode découpe l'espace en $2 \times d$ pyramides puis affecte aux données un numéro de pyramide et sa hauteur jusqu'au sommet de la pyramide. Ces valeurs constituent la clé d'indexation pour le stockage dans l'arbre B+. Les requêtes subissent les mêmes transformations pour être utilisables. Il n'y a pas de transformation inverse à effectuer. L'algorithme est compliqué mais sa programmation est assez simple. Cette méthode est l'une des premières à ne pas subir la malédiction de la dimension. La figure 2.16 nous montre la méthode de partitionnement de l'espace.

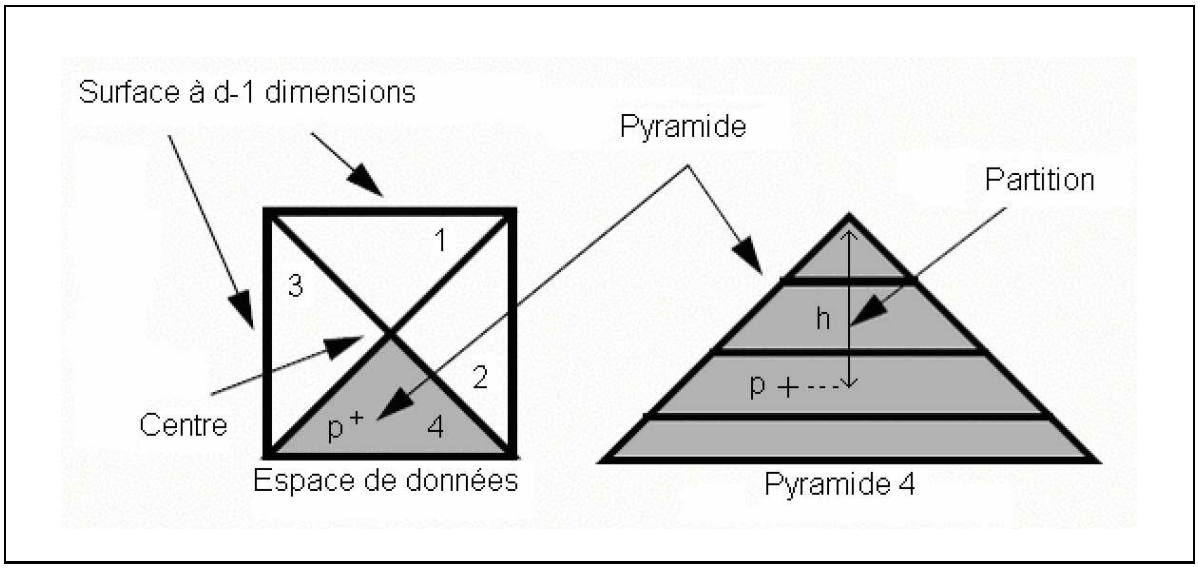

FIG. 2.16 – La technique de la Pyramide

La figure 2.16 prend l'exemple d'un point P dans l'espace de données. Ce point se situe dans la 4ème Pyramide et à une hauteur h par rapport au sommet de sa pyramide. Ce point P

peut alors être indexé dans l'arbre B+ avec comme clé d'indexation $(4+h)$. Toutes les données d'une base seront ainsi indexées dans un arbre B+ à l'aide de cette méthode.

L'algorithme de recherche utilise la même technique d'indexation pour placer le point "requête" dans une pyramide et ensuite dans l'arbre B+ contenant les données. Une recherche par intervalles est alors possible et permet de retourner les données les plus similaires à la requête.

Les performances de la technique de la Pyramide se dégradent lentement lorsque la dimension des données augmente. Cependant, elles dépendent fortement de la distribution des données et de la position de la requête dans l'espace. En effet, lorsque la distribution des données n'est pas uniforme, le choix du sommet des pyramides n'est pas significatif, ce qui détériore la qualité d'indexation et la rapidité de la recherche. De même, lorsqu'une requête est proche de la base (hyperplan) d'une pyramide. En effet, une telle requête peut générer des accès inutiles à un ensemble de données n'ayant pas forcément de similarité avec la requête. Ce qui risque d'affecter considérablement les performances de la recherche.

La technique de la Pyramide a servi de référence à plusieurs récentes méthodes développées : IminMax [OTYB00] se base sur un paramètre θ, à adapter selon la distribution des données, et utilise une fonction simple de transformation des données peu coûteuse en calcul. Cette technique est bien adaptée aux requêtes WQ, et elle est plus dynamique que la technique Pyramidale. Cependant, le choix du paramètre θ n'est pas évident et les performances en dépendent fortement. Idistance [YOTJ01] découpe l'espace de données en plusieurs partitions et choisit un point référence pour chacune de ces partitions. L'algorithme se base principalement sur des calculs de distance entre points. Cette technique est assez bien adaptée pour les requêtes KNN. P+Tree [ZOT04] répond aux deux types de requêtes KNN et WQ. Cette approche associe une méthode de division de l'espace basée sur *Bissecting K-means* [SBBG02] et la technique de la Pyramide. Les performances de P+Tree dépendent fortement des caractéristiques des sous espaces résultant de la méthode de division.

Idistance et IminMax ne sont adaptées qu'à un seul type de requête WQ et KNN respectivement. P+tree peut être considéré comme une amélioration de la technique Pyramidale. Cependant, cette technique n'est pas suffisamment efficace. En effet, pour résoudre le problème lié à la distribution non homogène des données, il est nécessaire de réduire l'espace de données à accéder lors d'une recherche et particulièrement lorsque la requête approche un coin ou

un hyperplan d'une pyramide. La division de l'espace en sous espaces apparaît alors comme une solution évidente. Cela permet d'appliquer la technique Pyramidale sur des espaces plus restreints.

L'efficacité d'une telle solution dépend naturellement du partitionnement des données et de la sélection des points références. P+Tree divise l'espace en hyper rectangles, puis applique la technique de la Pyramide à chacun de ces sous espaces. Leur méthode de division de l'espace est basée sur l'algorithme de *Bissecting K-means*, mais n'applique ce dernier qu'à une seule dimension. Leur découpage de l'espace ne tient pas suffisamment compte de la distribution des données. De ce fait, les sommets des pyramides ne constituent plus des points références. Si en pratique les requêtes sont supposées suivre la distribution des données, elles ne sont donc pas localisées près des sommets des pyramides, ce qui entraîne plus de calculs et par conséquent réduit les performances de P+Tree.

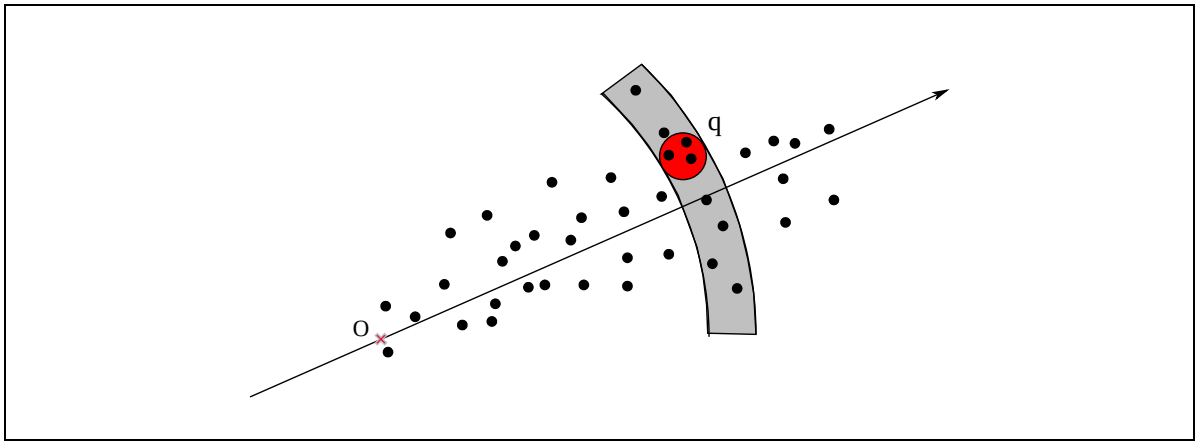

FIG. 2.17 – Représentation géométrique de l'indexation *vitri*

Plus récemment, Shen et al. [SOZ05] ont proposé une nouvelle structure d'indexation *ViTri Indexing* sous forme d'arbre B+. Tout d'abord, une classification binaire partitionne dynamiquement et récursivement l'espace en k classes. La classification s'arrête lorsque chaque classe a un rayon $r < \varepsilon$. Les index des données sont ensuite calculés en fonction de leur distance par rapport à O, O étant le projeté le plus à l'extérieur sur la composante principale (voir figure 2.17). L'espace accédé lors d'une requête est ainsi très restreint, voir exemple de la requête q

sur la figure. Les performances en temps de réponse et nombre d'accès aux données sont par conséquent meilleures que pour les autres méthodes citées.

Le tableau 2.5 donne les caractéristiques de chaque méthode d'indexation selon différents critères : la dimension, le nombre et la distribution des données traitées, ainsi que leur adaptation aux deux types de requêtes (KNN et WQ). Le tableau 2.6 synthétise les avantages et inconvénients de chacune de ces méthodes.

Méthodes	Dimension	Volume de données	Requêtes KNN	Requêtes WQ	Distribution des données
Pyramide	Peu élevée	Moyen	Non	Oui	Homogènes
IMinMax	Élevée	Moyen	Non	Oui	Homogènes
IDistance	Elevée	Grand	Oui	Non	Homogènes
P+Tree	Elevée	Grand	Oui	Oui	Peu Hétérogènes
ViTri Indexing	Très élevée	Grand	Oui	Oui	Hétérogènes

TAB. 2.5 – Avantages et Inconvénients des différentes méthodes d'indexation

2.6 Classification

La classification des données est un domaine très vaste et actif dans la fouille de données. Les algorithmes de classification y sont présents pour de nombreuses applications ([CUDW02, ZSD02, Dje03]), incluant la fouille de données multimédia. L'état de l'art exhaustif de ce domaine est exposé dans plusieurs articles de synthèse apparaissant avec une certaine régularité au fil des ans (voir [JD88, JMF99, Ber02, TSK06]). Deux grandes familles de classification automatique existent : la classification automatique "supervisée" qui nécessite la connaissance au préalable d'une classification pour classer de nouvelles données. La classification automatique "non supervisée" où aucune information est connue à l'avance pour former les différentes classes. Nous nous intéressons particulièrement dans nos recherches à la classification automatique non supervisée. Nous proposons tout d'abord de définir quelques notions importantes relatives à la classification. Ensuite nous présentons un aperçu général des méthodes de classification non supervisée. Enfin, nous détaillerons les méthodes basées sur la classification de

Méthodes	Inconvénients	Avantages
VA File	Codage lourd pour de grandes dimensions ; Non adaptée à des données hétérogènes	Implémentation simple de l'indexation et la recherche ; recherche séquentielle améliorée
Technique Pyramidale	Non adaptée pour des données hétérogènes ; Mauvaise gestion des requêtes KNN	Peu affectée par l'augmentation de la dimension Performante pour des données homogènes
iMinMax	Non adaptée pour des requêtes KNN	Peu affectée par l'augmentation de la dimension
iDistance	Non adaptée pour des requêtes WQ	Peu affectée par l'augmentation de la dimension
P+Tree	Algorithme de division binaire de l'espace peu performant ; Performances aléatoires pour des données hétérogènes	Peu adaptée à des requêtes K-NN ; Adaptée à des requêtes WQ Non affectée par l'augmentation de la dimension
ViTri Indexing	Algorithme de division binaire de l'espace peu performant ; Adaptée à des classes petites, denses et sphériques	Adaptée à des requêtes WQ ; Adaptée à des requêtes K-NN Non affectée par l'augmentation de la dimension

TAB. 2.6 – Caractéristiques des différentes méthodes d'indexation

sous espaces vectoriels.

2.6.1 Classification : généralités

La *classification des données* est un processus qui permet de regrouper un ensemble Ω de N données dans un ensemble de classes ayant un sens logique (similarité). Une partition de Ω est un ensemble de K classes $C_i, 0 < i \leq K$ telles que $\forall i, \bigcup C_i = \Omega$ et $\forall j, 0 < j \neq i \leq K, C_i \cap C_j = \oslash$.

Une *classe* contient un ensemble de données similaires deux à deux. La description d'une classe est représentative de l'ensemble des données incluses dans la classe. Plusieurs descriptions d'une classe existent dont les plus connues sont définies ci dessous. Soit $\mathbf{x}$ un vecteur de

l'espace, $\mathbf{x}_i$ un vecteur appartenant à C_i, N_i le nombre de vecteurs appartenant à C_i, $0 < i \leq K$, on définit par exemple :

- le centre de gravité ou le centroïde $\mathbf{c}_i$ de la classe C_i tel que $\mathbf{c}_i = \frac{1}{N_i}\sum \mathbf{x}_i$;
- le médoïde de la classe comme le vecteur $\mathbf{x}_i$ le plus proche du centre de gravité $\mathbf{c}_i$;
- l'inertie intra classe $I_i = \frac{1}{N_i}\sum(\mathbf{c}_i - \mathbf{x}_i)^2$;
- le rectangle minimum englobant REM_i de la classe C_i ;
- une formule de densité $dens_i = \frac{N_i}{REM_i}$;
- etc.

La qualité d'une classification peut être définie par une similarité intra classes élevée et inter classes basse. Plusieurs distances inter classes sont souvent utilisées dans les algorithmes de classification :

- le *lien simple* est la distance minimale entre les points de deux classes ;
- le *lien complet* est la distance maximale entre les points de deux classes ;
- l'*inertie* inter classes, avec $\mathbf{c}$, centre de gravité de l'ensemble des vecteurs, $I_intra = \frac{1}{N}\sum(\mathbf{c} - \mathbf{x})^2$;
- etc.

Les différents états de l'art existants sur la classification de données [JD88, JMF99] définissent un ensemble de conditions requises à une "bonne" méthode de classification :

- Extensibilité à un grand volume de données ;
- Habilité à traiter différents types de données ;
- Découverte de classes ayant des formes arbitraires ;
- Habilité à traiter les données multidimensionnelles ;
- Connaissances requises (paramètres de l'algorithme) ;
- Habilité à traiter les données bruitées et isolées ;
- Dépendance à l'ordre des données traitées ;
- Interprétation possible des résultats.

2.6.2 Aperçu général

Les méthodes de classification non supervisée peuvent se répartir en plusieurs grandes familles. La figure 2.18 propose une répartition répandue. Nous divisons l'ensemble des méthodes en quatre familles :

- les méthodes hiérarchiques ascendantes et descendantes ;
- les méthodes de type K-Means ;
- les méthodes basées sur la densité des données ;
- les méthodes probabilistes.

Les méthodes de classification hiérarchique construisent une hiérarchie de classes. Elle est représentée sous forme d'un arbre appelé dendogramme. Une classification est obtenue en délimitant un niveau de l'arbre, par exemple pour obtenir le nombre de classes désiré. Les méthodes hiérarchiques se répartissent en deux catégories :

- Les méthodes hiérarchiques ascendantes ou agglomératives : elles s'initialisent en utilisant chaque donnée comme une classe. Chaque étape regroupe les deux classes les plus similaires. La hiérarchie de classe est complète lorsque qu'il n'existe plus qu'une seule classe.
- Les méthodes hiérarchique descendantes ou divisives : elles s'initialisent en groupant l'ensemble des données en une seule classe. A chaque étape de l'algorithme, une des classes est divisée en deux classes. La hiérarchie est complète lorsque chaque point représente une classe.

Ces méthodes présentent certains avantages : un niveau de granularité peut être défini après avoir complété la hiérarchie pour trouver la classification adéquate ; elles s'adaptent à tous types de données tant qu'une mesure de similarité ou une distance est établie pour l'ensemble de la base de données. Néanmoins, la plupart des méthodes hiérarchiques supportent très mal la classification de grands volumes de données. En effet, la construction de l'arbre est très coûteuse. De plus, le groupement ou la division des classes est définitive, une fois qu'une action est réalisée, l'algorithme n'effectue pas une deuxième "passe" pour améliorer la classification.

Une deuxième famille d'algorithmes se base sur un partitionnement des données utilisant la densité de voisinage. Ces méthodes permettent de trouver des classes ayant des formes

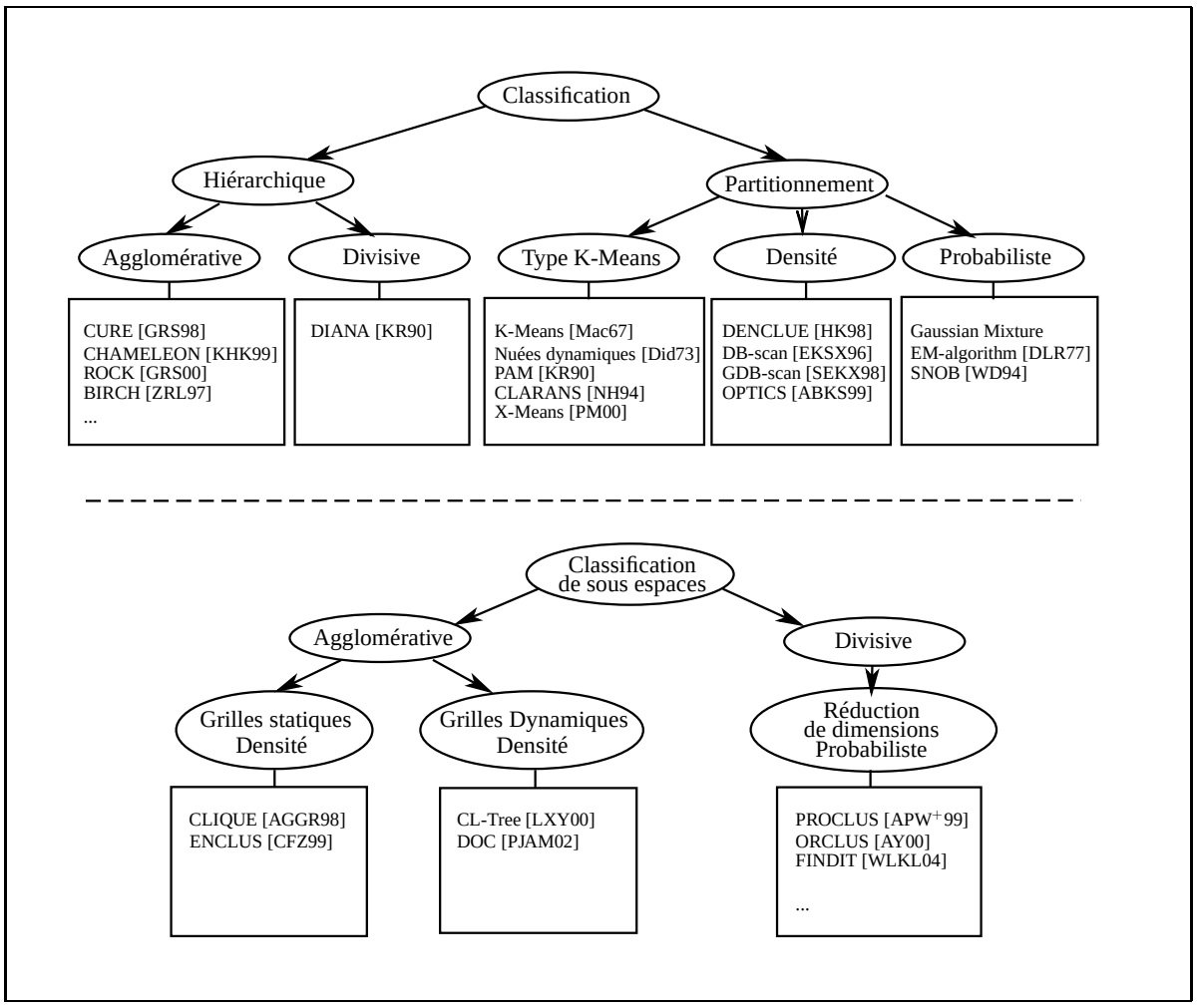

FIG. 2.18 – Répartition des méthodes de classification

arbitraires et de gérer les données « bruit ». Elles sont basées sur la notion de similarité de voisinage. Une classe va inclure les points voisins si ces derniers ne font pas diminuer la densité de la classe en dessous d'une valeur limite (souvent paramètre de l'algorithme). Ces algorithmes basés sur la densité sont souvent complexes et sont peu adaptables à des grands volumes de données de dimension élevée.

Une des familles d'algorithmes les plus connus et populaires est la famille k-Means. Cette famille d'algorithme par partitionnement a pour but de trouver une partition de k classes op-

timisant une fonction de similarité ou de distance, en réalisant plusieurs passes sur la base de données. L'algorithme k-Means utilise le centre de gravité pour représenter chaque classe, l'algorithme k-Médoïdes utilise l'objet de la classe le plus proche du centre de gravité (appelé médoïde) et l'algorithme des Nuées dynamiques se base sur un noyau d'objets pour représenter chaque classe. Ces méthodes sont simples et efficaces. Néanmoins elles ont pour inconvénient majeur le choix du paramètre *k* de l'algorithme même si certaines versions de ces algorithmes proposent l'utilisation de critères statistiques pour déterminer le nombre de classes automatiquement (voir X-Means [PM00]). Elles sont très sensibles aux données « bruit » et ne découvrent que des classes de forme convexe. Enfin, la partition finale des classes est dépendante de l'initialisation de l'algorithme.

Très proche de la famille de k-Means, la famille des algorithmes de classification basés sur une mesure probabiliste de type Expectation-Maximisation [DLR77]. Pour chaque donnée, l'espérance d'appartenir à chaque centre est calculée, ensuite les centres sont recalculés pour trouver le maximum de vraisemblance du modèle probabiliste choisi. L'algorithme prend fin lorsqu'il a atteint un maximum local de vraisemblance. Les algorithmes de cette famille sont coûteux. La partition finale est dépendante de l'initialisation du modèle probabiliste. Néanmoins, la représentation des classes est différente d'un ensemble de données, l'algorithme peut donc être arrêté à tout moment et être facilement interprétable. Dans la figure 2.18, nous proposons quelques références non exhaustives pour chacune des catégories présentées.

2.6.3 Méthodes basées sur la classification de sous espaces vectoriels

Nous proposons dans cette section, de développer une famille de méthodes de classification apparue récemment pour répondre aux problèmes liés aux grandes bases de données de dimension élevée. Il s'agit des méthodes basées sur la classification de sous espaces vectoriels (*Subspace Clustering*). Ces méthodes apparaissent dans de nombreux domaines avec différentes applications, en particulier le domaine de la bio informatique avec la classification de gènes et le domaine des données multimédia pour la classification d'images, de vidéos ou d'objets 3D. Toutes les méthodes présentées dans cette section combinent des idées des grandes familles de la section précédente. Elles sont toutes basées sur la réduction de la dimension de l'espace pour

obtenir une classification des données. Cette famille de méthodes basées sur la classification de sous espaces vectoriels peut être divisée en plusieurs sous familles (voir la figure 2.18).

Il est reconnu que la classification des données dans des espaces de petite dimension est plus simple que dans des espaces de dimension élevée. Par exemple dans un espace unidimensionnel, les régions de forte densité sont facilement identifiables par une simple recherche linéaire. Avec l'augmentation de la dimensionnalité, le problème devient plus complexe. La notion de classification par projection dans des sous espaces est alors introduite par Agrawal et al. dans [AGGR98]. Ces derniers ont observé que les points dans des sous espaces de plus faible dimensionnalité sont plus facilement regroupés en classes que dans l'ensemble de l'espace d'origine $\mathbb{R}^n$, idée qu'ils ont utilisée dans leur algorithme CLIQUE. L'algorithme CLIQUE sert de référence pour plusieurs algorithmes. Cette famille d'algorithmes utilise un algorithme hiérarchique agglomératif basé sur un regroupement APRIORI selon une mesure de similarité choisie pour déterminer les régions denses de l'espace.

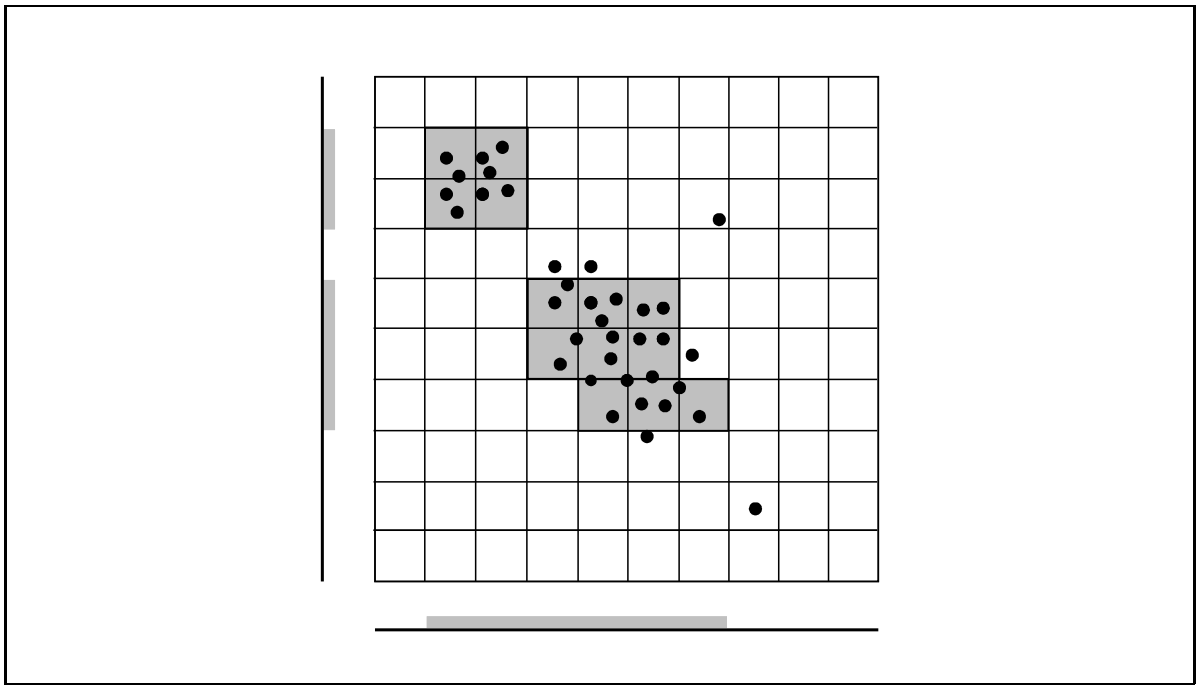

FIG. 2.19 – CLIQUE : grille et densité

Nous détaillons l'algorithme CLIQUE et mentionnons ensuite quelques autres méthodes de la même famille. CLIQUE utilise un découpage de l'espace basé sur une grille statique (voir figure 2.19). Chaque dimension est alors découpée en régions unitaires. Pour un espace de dimension k, $k \in [1,d]$ où d est la dimension de l'espace vectoriel d'origine, les régions unitaires contenant une population dense de données sont utilisées pour former un nouvel ensemble de régions de dimension $k+1$. Ces dernières sont analysées pour connaître leur densité et déterminer si l'algorithme les garde en mémoire pour la récurrence suivante. L'algorithme récursif se termine pour $k = d$. Les régions denses obtenues sont alors étudiées deux à deux ; l'algorithme regroupe deux régions denses en une seule si elles ont un hyper-plan en commun. Le nouvel ensemble de régions denses obtenu à la fin de l'algorithme est considéré comme la classification finale des données.

Basée principalement sur CLIQUE, la méthode ENCLUS [CFZ99] n'utilise pas de mesure de densité mais une mesure d'entropie de la classe. ENCLUS se base sur le fait que des sous espaces vectoriels contenant au moins une classe ont une entropie plus faible que des sous espaces ne contenant pas de classe. L'algorithme MAFIA présente une amélioration de CLIQUE. Cet algorithme, proposé par [NGC99], est basé sur un découpage dynamique de chaque dimension en fonction de la densité des données avant de combiner les cellules denses. Cette méthode a des performances fort dépendantes des deux paramètres utilisés : le nombre de cellules par dimension et la densité minimale d'une cellule. De plus MAFIA propose de parallèliser son algorithme pour calculer les régions denses.

La méthode CL-Tree [LXY00] adapte un algorithme d'arbre de décisions pour détecter les régions denses de chaque dimension avant de les regrouper. Cet algorithme a aussi deux paramètres influant sur ses performances : la densité minimale pour fusionner deux cellules et le nombre minimal de données que doit contenir une région pour être considérée comme une classe. *CBF* pour *CellbasedClusteringMethod* est un autre algorithme proposé dans [CJ02] de la même famille que CL-Tree. Cette méthode propose une adaptation optimale du découpage en régions unitaires en fonction du nombre de données ainsi que de leur répartition dans le sous espace étudié. Une autre approche appelée *DOC* est présentée par [PJAM02]. Elle détermine mathématiquement une partition optimale dans les sous espaces. Leur méthode n'est pas basée sur les grilles statiques ou dynamiques mais sur la densité d'hyper-rectangles de largeur fixée.

La méthode SUBCLU présentée par [KKK04] utilise une version hiérarchique ascendante de l'algorithme DB-scan. Il commence par trouver les classes dans chaque sous espace unidimensionel, puis construit à chaque itération les classes d'espace vectoriel de dimension supérieure. Cet algorithme présente les avantages observés des méthodes de classification par densité de voisinage : habilité à trouver les classes de formes arbitraires, n'est pas affecté par les données "bruit". Néanmoins, cet algorithme présente une complexité élevée par rapport à tous les autres algorithmes.

Parallèlement, une autre famille d'algorithmes est apparue et s'est développée. Cette famille d'algorithmes commence par déterminer un ensemble de classes approximatives dans l'espace d'origine pour ensuite améliorer récursivement l'ensemble des classes formées par des mesures inter et/ou intra classes modifiant pour certaines d'entre elles le nombre de dimensions ou l'importance de ces dernières. Nous pouvons trouver dans cette famille les algorithmes PROCLUS [APW+99] et ORCLUS [AY00]. L'algorithme ORCLUS qui est une amélioration de PROCLUS, utilise un ensemble de points choisis aléatoirement comme centres de classe (appelés aussi médoïdes des classes). Ensuite, l'algorithme améliore récursivement la classification en diminuant petit à petit le nombre de dimensions et le nombre de classes (par fusion ou suppression des classes) pour former k classes dans q dimensions, k et q étant des paramètres utilisateur de l'algorithme. La réduction de dimension se fait en choisissant les dimensions les plus représentatives pour chaque classe par un calcul des valeurs propres minimales de la matrice de covariance de chaque classe. Les médoïdes sont recalculés dans le nouvel sous espace formé de dimension moins élevée qu'à l'étape précédente. L'algorithme FINDIT [WLKL04] est une variante de ces algorithmes. Les dimensions choisies correspondent à celles qui ont le pourcentage de données proche du médoïde le plus élevé.

Les recherches de classification par sous espaces dans le domaine de la bio informatique sont devenues très actives ces dernières années. Plusieurs méthodes ont été présentées. La technique de classification HARP [YCN04] est basée principalement sur la classification hiérarchique agglomérative. A la différence des méthodes précédentes, la méthode HARP utilise des paramètres dynamiques. Le nombre de dimensions et la pertinence des dimensions choisies sont déterminées dynamiquement. Leur mesure de pertinence sur une dimension est en fonction de la variance interne de la classe et la variance globale de l'ensemble des données.

Le nombre de classes désiré reste un paramètre possible pour éviter de construire la hiérarchie complète.

La méthode LAC [DPGM04] est basée sur une classification des données du type K-means. En effet, elle se base sur le centre de gravité de la classe. Un vecteur de poids est associé à chaque centre de gravité. Le poids associé à un centre de gravité pour une dimension précise est inversement proportionnel à la dispersion des données de la classe sur la dimension.

La classification de sous espaces vectoriels est un domaine en pleine expansion. Les données de dimension élevée deviennent incontournables, et le volume des bases de données ne cesse d'augmenter surtout dans le domaine du multimédia. Les méthodes présentées ci-dessus présentent toutes l'avantage de répondre à la malédiction des grandes dimensions. Un article de synthèse [PHL04] résume la majorité des méthodes et essaie de comparer les performances de certaines d'entre elles. Pour résumer, les méthodes agglomératives évaluent la classification dans des sous espaces de dimension faible, élèvent petit à petit la dimension pour obtenir la classification dans l'espace originel. Elles présentent l'avantage de trouver des classes de formes arbitraires. L'inconvénient majeur de ces méthodes est le paramètre de densité minimale. Pour les méthodes divisives, la classification s'initialise avec l'ensemble des données pour être ensuite améliorée en pondérant ou supprimant des dimensions. Ces méthodes requiert le nombre de classes à obtenir et la dimension minimale des sous espaces à étudier. Les plus récentes tentent de limiter le nombre de paramètres par des mesures statistiques sur les classes. Les classes formées sont par nature de forme hyper-sphériques (basée sur le centre de gravité) et les données "bruit" sont souvent mal gérées.

2.7 Conclusion

Ce chapitre présente un aperçu de l'état de l'art des différents domaines liés aux travaux de recherches présentés dans les sections suivantes. Nous avons pu remarquer l'étendu du domaine de la fouille de documents multimédia. Nous avons choisi un premier axe de recherche orienté vers les structures d'indexation multidimensionnelle. La plupart des structures d'indexation existantes ne sont pas adaptées à de grandes bases de données de dimension élevée. Nous avons montré aussi que la distribution hétérogène des données détériorait rapidement les

performances des techniques les plus récentes. Ce qui nous a conduit à étudier les méthodes de regroupement dans des sous espaces vectoriels. Ces dernières présentent souvent un nombre de paramètres difficiles à déterminer par l'utilisateur même expert de la base. A partir de ces différentes remarques sur les méthodes existantes, nous avons pu nous positionner et proposer plusieurs structures d'indexation multidimensionnelle et une technique de classification que nous développons dans les chapitres suivants.

Bibliographie

[Abd07] Hervé Abdi. Distance. In Neil J. Salkind, editor, *Encyclopedia of Measurement and Statistics*, pages 280–284. Sage, 2007.

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics : Ordering points to identify the clustering structure. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 49–60, Philadelphia, Pennsylvania,USA, 1999. ACM Press.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM-SIGMOD Int. Conf. Management of Data,*, pages 94–105, 1998.

[AGQ06] S. Ayache, J. Gensel, and G. M. Quénot. Clips-lsr experiments at trecvid 2006. In *TRECVID'2006 Workshop*, Gaithersburg, MD, USA, 2006.

[AI] LSH Algorithm and Implementation. http ://web.mit.edu/andoni/www/LSH/index.html.

[APW+99] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 61–72, Philadelphia, Pennsylvania,USA, 1999. ACM Press.

[AY00] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 70–81, Dallas, Texas, USA, 2000. ACM.

[BBJ+00] S. Berchtold, C. Böhm, H. V. Jagadish, H. P. Kriegel, and J. Sander. Independant quantization : An index compression technique for high-dimensional data spaces. In *Proceedings of 16-th Int. Conf. on Data Engineering, IEEE ICDE*, pages 577–588, San diego, California, USA, 2000.

[BBK98] S. Berchtold, C. Bohm, and H. P. Kriegel. The pyramid technique : Towards breaking the curse of dimensionality. In *Proceedings of ACM SIGMOD Int. Conf on Management of Data*, pages 142–153, Seatle,Washington,USA, 1998.

[Ben79] J. L. Bentley. Multidimensional binary search trees in database applications. *Proceedings of IEEE Transactions on Software Engineering*, SE-5 :333–340, 1979.

[Ber02] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA, 2002.

[Ber04] Sid-Ahmed Berrani. Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu., 2004.

[BHR00] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *SPIRE*, pages 39–48, 2000.

[BKK96] S. Berchtold, D.A. Keim, and H.P. Kriegel. The x-tree : An index structure for high-dimensional data. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 322–331, Bombay, Inde, 1996. Morgan Kaufmann.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r*-tree : An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD International conference on Management of Data*, pages 322–331, Atlantic City, NJ, USA, 1990. ACM Press.

[BM72] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1 :173–189, 1972.

[CC02] Guang-Ho Cha and Chin-Wan Chung. The gc-tree : a high-dimensional index structure for similarity search in image databases. *IEEE Transactions on Multimedia*, 4(2) :235–247, 2002.

[CCS04] Gerardo Canfora, Luigi Cerulo, and Rita Scognamiglio. Measuring xml document similarity : A case study for evaluating information extraction systems. In *10th IEEE International Software Metrics Symposium (METRICS 2004)*, pages 36–45, Chicago, IL, USA, 2004. IEEE Computer Society.

[CFZ99] C. H. Cheng, A. W.-C. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *5th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 84–93. ACM, 1999.

[CIEI+07] CLIPS-IMAG, EURECOM, INA, IRIT, LABRI, LIP6, NOVELTIS, and SFRS. Campagne d'évaluation d'outils de surveillance de contenus vidéo, 2007. http ://www.irit.fr/argos.

[CJ02] J.-W. Chang and D.-S. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC)*, pages 503–507, Madrid, Spain, 2002. ACM.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree : An efficient access method for similarity search in metric spaces. In *Proceedings of 23rd International Conference on Very Large Data Bases*, pages 426–435, Athenes, Grece, 1997. Morgan Kaufmann.

[CUDW02] A. B. Chaudri, R. Unland, C. Djeraba, and W.Lindner, editors. *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops*, volume LNCS 2490 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2002.

[CZ03] S.-C. S. Cheung and A. Zakhor. Efficient video similarity measurement with video signature. *IEEE Trans. Circuits Syst. Video Techn.*, 13(1) :59–74, 2003.

[CZPC02] G.-H. Cha, X. Zhu, P. Petkovic, and C.-W. Chung. An efficient indexing method for nearest neighbor searches in high-dimensional image databases. *IEEE Transactions on Multimedia*, 4(1) :76–87, 2002.

[Did73] Edwin Diday. The dynamic clusters method and optimization in non-hierarchical clustering. In *5th Conference on Optimization Techniques*, volume 3 of *Lecture Notes in Computer Science*, pages 241–258, Rome, Italy, 1973. Springer.

[DIIM04] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In J. Snoeyink and J. Boissonnat, editors, *Proceedings of the 20th ACM Symposium on Computational Geometry*, pages 253–262, Brooklyn, New York, USA, 2004. ACM.

[Dje03] C. Djeraba, editor. *Multimedia Mining - A Highway to Intelligent Multimedia Documents*. Kluwer, Boston, 2003.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the *em* algorithm. *Journal of the Royal Statistical Society*, 39(1) :1–38, 1977.

[DPGM04] C. Domeniconi, D. Papadopoulos, D. Gunopulos, and S. Ma. Subspace clustering of high dimensional data. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, Lake Buena Vista, Florida, USA, 2004. SIAM.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, 1996.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2) :170–231, 1998.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, pages 518–529, Edinburgh, Scottland, UK, 1999. Morgan Kaufmann.

[GLC02] K. Goh, B. Li, and E. Chang. Dyndex : a dynamic and non-metric space indexer. In *MULTIMEDIA '02 : Proceedings of the tenth ACM international conference on Multimedia*, pages 466–475, Juan-les-Pins, France, 2002. ACM Press.

[GRS98] S. Guha, R. Rastogi, and K. Shim. Cure : An efficient clustering algorithm for large databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 73–84, Seattle, Washington, USA, 1998. ACM Press.

[GRS00] S. Guha, R. Rastogi, and K. Shim. Rock : A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5) :345–366, 2000.

[Gut84] A. Guttman. R-trees : a dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, USA, 1984. ACM Press.

[Hen98] A. Henrich. The lsd^h-tree : An access structure for feature vectors. In *Proceedings of the Fourteenth International Conference on Data Engineering (ICDE)*, pages 362–369, Orlando, Florida, USA, 1998. IEEE Computer Society.

[HK98] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 58–65, 1998.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer. The lsd tree : Spatial access to multidimensional point and nonpoint objects. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 45–53, Amsterdam, Pays-Bas, 1989. Morgan Kaufmann.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors : Towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[JBPKQ07] Philippe Joly, Jenny Benois-Pineau, Ewa Kijak, and Georges Quénot. The argos campaign : Evaluation of video analysis and indexing tools. *Image Commun.*, 22(7-8) :705–717, 2007.

[JD88] A. K. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : A review. *ACM Computing Surveys*, 31 :264–323, 1999.

[Jol06] Alexis Joly. Recherche par similarité statistique dans une grande base de signatures locales pour l'identification rapide d'extraits vidéo., 2006. Doctorat de l'université de La Rochelle.

[KHK99] G. Karypis, E.-H. Han, and V. Kumar. Chameleon : Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8) :68–75, 1999.

[KKK04] Peer Kröger, Hans-Peter Kriegel, and Karin Kailing. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, Lake Buena Vista, Florida, USA, 2004. SIAM.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data : An Introduction to Cluster Analysis.* John Wiley, 1990.

[KS97] N. Katayama and S. Satoh. The sr-tree : An index structure for high-dimensional nearest neighbor queries. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 369–380, Tucson,Arizona,USA, 1997. ACM Press.

[KV05] C. Kim and B. Vasudev. Spatiotemporal sequence matching for efficient video copy detection. *IEEE Transaction on Circuits and Systems for Video Technology*, 15(1) :127–132, 2005.

[LJW+06] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. A time-space efficient locality sensitive hashing method for similarity search in high dimensions. Technical report, Princeton University, New Jersey, USA, 2006.

[LOH05] J. Lee, J.-H. Oh, and S. Hwang. Strg-index : Spatio-temporal region graph indexing for large video databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 718–729, Baltimore, Maryland, USA, 2005.

[LXY00] B. Liu, Y. Xia, and P. S. Yu. Clustering through decision tree construction. In *9th international conference on Information and knowledge management*, pages 20–29. ACM, 2000.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, 1967. University of California Press.

[Mar04] José M. Martínez. Mpeg-7 overview, 2004. http ://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

[MF02] Oge Marques and Borko Furht. *Content-Based Image and Video Retrieval*. Springer, 2002.

[MLBM05] N. Mo
ënne-Loccoz, E. Bruno, and S. Marchand Maillet. Interactive retrieval of video

sequences from local feature dynamics. In *Proceedings of the 3rd International Workshop on Adaptive Multimedia Retrieval, AMR
05*, Glasgow, UK, July 2005.

[Moh98] Rakesh Mohan. Video sequence matching. In *Int. Conf. on Audio, Speech and Signal Processing (ICASSP)*, volume 6, pages 3697–3700, 1998.

[Mpe] Mpeg7. http ://www.chiariglione.org/mpeg/.

[NGC99] H. Nagesh, S. Goil, and A. Choudhary. Mafia : Efficient and scalable subspace clustering for very large data sets. Technical report, Northwestern University, 1999.

[NH94] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases*, pages 144–155, Santiago, Chile, 1994. Morgan Kaufmann Publishers.

[OT02] B. C. Ooi and K.-L. Tan. B-trees : Bearing fruits of all kinds. In Xiaofang Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, volume 5, Melbourne, Victoria, Australie, 2002. Australian Computer Society.

[OTYB00] B. C. Ooi, K. L. Tan, C. Yu, and S. Bressan. Indexing the edges - a simple and yet efficient approach to high-dimensional. In *Proceedings of 19-th ACM SIGMOD SIGACT SIGART Symposium on Principles of Database Systems*, pages 166–174, Dallas, Texas, USA, 2000.

[PHL04] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data : a review. *SIGKDD Explorations*, 6(1) :90–105, 2004.

[PJAM02] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 418–427, Madison, Wisconsin, USA, 2002. ACM.

[PM00] D. Pelleg and A. W. Moore. X-means : Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International*

Conference on Machine Learning (ICML 2000), pages 727–734, Stanford University, Standord,CA, USA, 2000. Morgan Kaufmann.

[PPC06] V. Parshin, A. Paradzinets, and L. Chen. Multimodal data fusion for video scene segmentation. In *Visual Information and Information Systems, 8th International Conference, VISUAL 2005*, volume 3736 of *Lecture Notes in Computer Science*, Amsterdam, The Netherlands, 2006. Springer.

[RHC99] Y. Rui, T. Huang, and S. Chang. Image retrieval : current techniques, promising directions and open issues. *Journal of Visual Communication and Image Representation*, 10(4) :39–62, april 1999.

[Rob81] J. T. Robinson. The k-d-b-tree : A search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, Michigan, USA, 1981. ACM Press.

[SBBG02] S. M. Savaresi, D. Boley, S. Bittanti, and G. Gazzaniga. Cluster selection in divisive clustering algorithms. In *Proceedings of the Second SIAM International Conference on Data Mining*, Arlington, VA, USA, 2002. SIAM.

[SEKX98] J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Density-based clustering in spatial databases : The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery, an International Journal*, 2 :169–194, 1998.

[SOZ05] H. T. Shen, B. C. Ooi, and X. Zhou. Towards effective indexing for very large video sequence database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 730–741, Baltimore, Maryland, USA, 2005. ACM.

[SPK07] I. Song, J. Paik, and U. Kim. Semantic-based similarity computation for xml document. *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, 0 :796–803, 2007.

[SWS+00] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12) :1349–1380, 2000.

[SYUK00] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The a-tree : An index structure for high-dimensional spaces using relative approximation. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 516–526. Morgan Kaufmann, 2000.

[Tek] Kardi Teknomo. http ://www.people.revoledu.com/kardi/tutorial/Similarity/.

[TSK06] P. N ; Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson/Addison-Wesley, Boston, 2006.

[TZKO06] A. K. H. Tung, R. Zhang, N. Koudas, and B.C. Ooi. Similarity search : A matching based approach. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 631–642. ACM press, 2006.

[UAH76] J. D. Ullman, A. V. Aho, and D. S. Hirschberg. Bounds on the complexity of the longest common subsequence problem. *J. ACM*, 23(1) :1–12, 1976.

[W3C] W3C. http ://www.w3.org/.

[WD94] S. WAllace and D. L. Dowe. Intrinsic classification by mml - the snob program. In *the 7 Australian Joint Conference on Artificial Intelligence*, pages 37–44, Singapour, 1994.

[WF74] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1) :168–173, 1974.

[WJ96] D. A. White and R. Jain. Similarity indexing with the ss-tree. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, pages 516–523, New Orleans, Louisiana, USA, 1996. IEEE Computer Society.

[WLKL04] K.-G. Woo, J.-H. Lee, M.-H. Kim, and Y.-J. Lee. Findit : a fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology*, 46(4) :255–271, March 2004.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of 24rd International Conference on Very Large Data Bases*, pages 194–205, New York city, New York, USA, 1998. Morgan Kaufmann.

[YCN04] K. Y. Yip, D. W. Cheung, and M. K. Ng. Harp : A practical projected clustering algorithm. *IEEE Transactions on Knowledge Data Engineering*, 16(11) :1387–1397, 2004.

[YL99] Y. Yang and X. Liu. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49, Berkley,USA, August 1999.

[YOTJ01] C. Yu, B. C. Ooi, K. L. Tan, and H. V. Jagadish. Indexing the distance : an efficient method to knn processing. In *Proceedings of 27-th Int. Conf. on Very Large Data Bases*, pages 421–430, Rome,Italy, 2001.

[ZAF+03] X. Zhu, W. G. Aref, J. Fan, A. C. Catlin, and A. K. Elmagarmid. Medical video mining for efficient database indexing, management and access. In *Proceedings of the 19th International Conference on Data Engineering*, pages 569–580. IEEE Computer Society, 2003.

[ZDC06] B. Zhang, W. Dou, and L. Chen. Combining short and long term audio features for tv sports highlight detection. In *Advances in Information Retrieval, 28th European Conference on IR Research, ECIR 2006*, Lecture Notes in Computer Science, pages 472–475, London, UK, 2006. Springer.

[ZEX+05] X. Zhu, A. K. Elmagarmid, X. Xue, L. Wu, and A. C. Catlin. Insightvideo : toward hierarchical video content organization for efficient browsing, summarization and retrieval. *IEEE Transactions on Multimedia*, 7(4) :648–666, 2005.

[ZLCZ02] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity metric for xml documents. In *Workshop on Knowledge and Experience Management, FGWM 2003*, Karlsruhe, Germany, 2002.

[ZOT04] R. Zhang, B. C. Ooi, and K. L. Tan. Making the pyramid technique robust to query types and workloads. In *Proceedings of 20-th Int. Conf. on Data Engineering, IEEE ICDE*, pages 313–324, Boston, USA, 2004.

[ZRL97] T. Zhang, R. Ramakrishnan, and M. Livny. Birch : A new data clustering algorithm and its applications. *Data Mining Knowledge Discovery*, 1(2) :141–182, 1997.

[ZS89] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6) :1245–1262, 1989.

[ZSD02] O. R. Zaïane, S. Simoff, and C. Djeraba, editors. *Mining Multimedia and Complex Data*, volume LNAI 2797 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 2002.

[ZZS07] X. Zhou, X. Zhou, and H. T. Shen. A new similarity measure for near duplicate video clip detection. In *Advances in Data and Web Management, Joint 9th Asia-Pacific Web Conference, APWeb 2007, and 8th International Conference, on Web-Age Information Management, WAIM 2007*, volume 4505 of *Lecture Notes in Computer Science*, pages 176–187, Huang Shan, China, 2007. Springer.

Chapitre 3

Indexation Multidimensionnelle

Sommaire

L'indexation multidimensionnelle est un domaine de recherche apparu pour gérer l'augmentation de la complexité des données. Dans ce domaine les données multimédia (images, vidéos, objets 3D, etc.) présentent de nombreuses caractéristiques. Nos recherches s'orientent vers une base de séquences vidéo dont le contenu est décrit par un document Mpeg-7 [Mpe], norme de description audiovisuelle et multimédia utilisant le langage XML. Le principal but est d'accéder à l'information contenue dans les données le plus rapidement possible. Ces données peuvent être indexées sous leur forme originelle par une méthode de classification ou d'indexation de documents XML. Il est aussi possible de les transformer en vecteurs multidimensionnels, avec un modèle de représentation vectorielle, pour ensuite les indexer dans une structure d'indexation. Dans ce chapitre, nous présentons tout d'abord une description du cadre général de notre approche, puis nous développons notre première méthode Kpyr dans le domaine de l'indexation multidimensionnelle, enfin nous terminons par la présentation de KpyrRec, une amélioration de Kpyr.

3.1 Cadre général

Le but de notre approche est d'apporter des outils d'indexation et de recherche performants pour un moteur de recherche de séquences vidéo permettant ainsi un accès efficace et rapide à l'information contenue dans les séquences vidéo. Notre base de données, composée exclusivement de films d'entreprise, a été annotée par des experts métier. Nous récupérons ainsi des documents Mpeg-7 contenant la description du contenu audiovisuel des séquences vidéo de tous les films. La figure 3.1 présente un aperçu général de notre approche. Sur fond vert, les étapes que nous détaillons dans ce chapitre, sur fond gris, l'interface utilisateur décrite à la section 6.2. Notre approche comporte deux grandes phases de traitement :

– Un traitement off-line qui consiste à indexer les contenus vidéo.
– Un traitement on-line correspondant à l'interrogation de la base de films.

Dans le traitement off-line, nous avons élaboré plusieurs structures d'indexation multidimensionnelle optimisée pour la recherche : Kpyr, KpyrRec et RPyR (détaillées dans les sections et chapitre suivants). Par souci d'implantation, nous utilisons une structure classique de type table de hachage pour des informations exactes telles que les noms propres des acteurs,

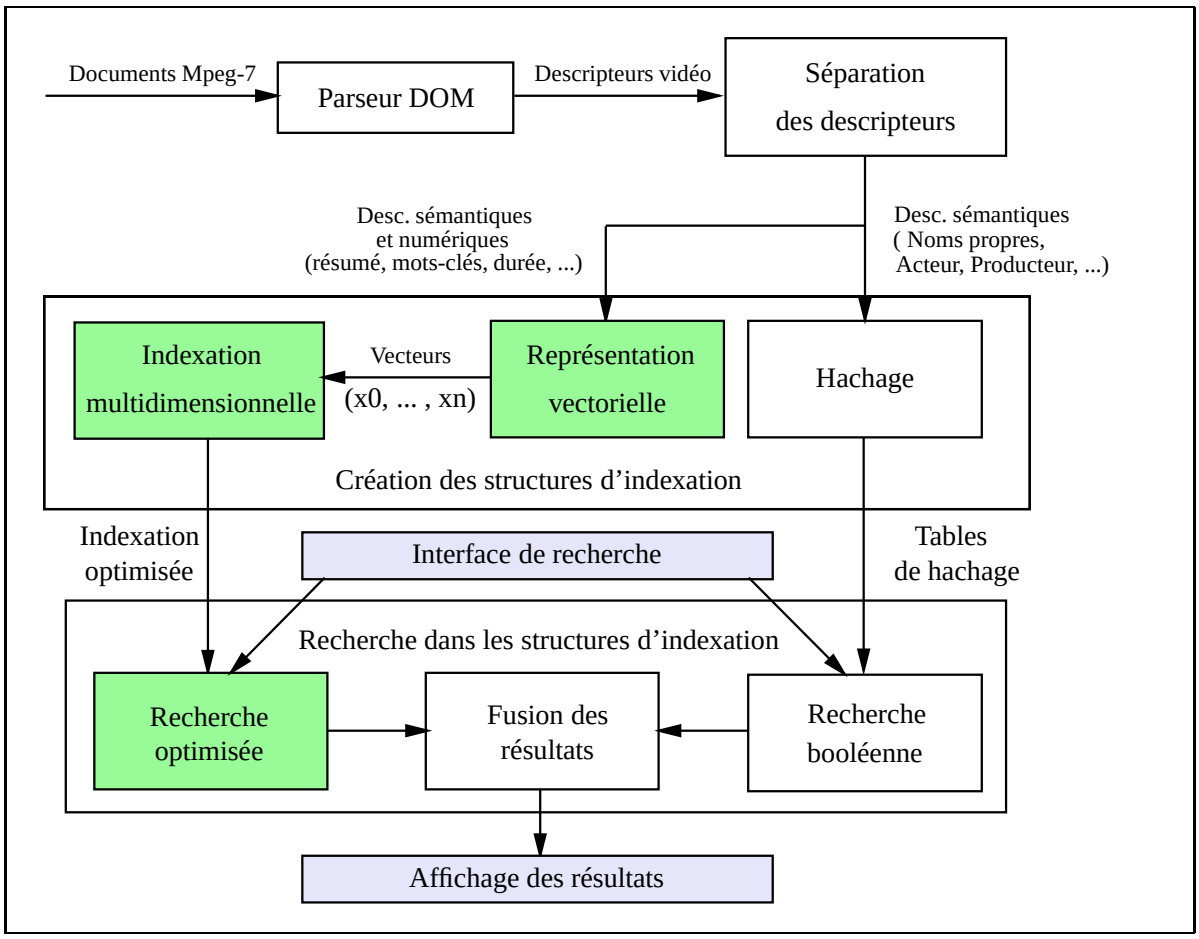

FIG. 3.1 – Aperçu de l'architecture de notre approche

réalisateurs, etc. La construction d'une telle structure nécessite l'utilisation d'un modèle de représentation vectorielle qui transforme les informations (mots clés, résumé, durée,etc.) de tous les autres descripteurs en vecteurs multidimensionnels. Nous présentons notre modèle dans la section 6.1.

La partie on-line correspond à l'accès aux informations via nos structures d'indexation en fonction des requêtes faites par un utilisateur à l'aide de notre interface de recherche. Nous utilisons les tables de hachage pour la recherche booléenne sur l'ensemble des séquences vidéo concernant des informations exactes. Nous avons élaboré une méthode qui transforme la

requête en un point multidimensionnel dans l'espace vectoriel et effectue une mise en correspondance avec les données existantes contenues dans la structure d'indexation afin d'accéder le plus rapidement possible aux séquences vidéo les plus pertinentes. Par ailleurs, un algorithme de fusion des résultats permet de combiner les ensembles de résultats obtenus avant de les retourner à l'interface utilisateur pour leur affichage.

3.2 Kpyr, structure d'indexation multidimensionnelle

Dans Kpyr, notre idée de base consiste à réunir les conditions favorables à l'application de la technique de la Pyramide [BBK98] et ce, en réorganisant l'espace en sous espaces homogènes et en introduisant une nouvelle structure arborescente qui gère ces espaces.

3.2.1 Algorithmes

L'**algorithme général d'indexation de Kpyr** se divise en trois étapes :

1. Pour une meilleure indexation des données, nous utilisons l'algorithme de classification K-means [Mac67]. Nous obtenons à la fin de cette étape K classes homogènes plus adaptées à une bonne utilisation de la technique de la Pyramide.
2. Pour chaque classe, nous effectuons un changement de base du sous espace correspondant en un hyper-cube unitaire. Ensuite, nous appliquons la technique de la Pyramide sur chaque classe et obtenons un arbre B+ par classe.
3. Enfin, un algorithme, détaillé plus loin dans cette section, détermine les frontières des régions de l'espace représentées par chaque classe. Cet algorithme crée à partir de ces frontières un "arbre Espace" binaire dont les noeuds feuilles sont les arbres B+ de chaque classe et dont les autres noeuds représentent les frontières déterminées entre les régions des classes.

La figure 3.2 présente un exemple simple en deux dimensions d'une classification avec K-means en quatre classes distinctes. Sur chacune de ces classes, nous appliquons la technique de la Pyramide. Nous prenons l'exemple précis du vecteur P de la figure dont les coordonnées en

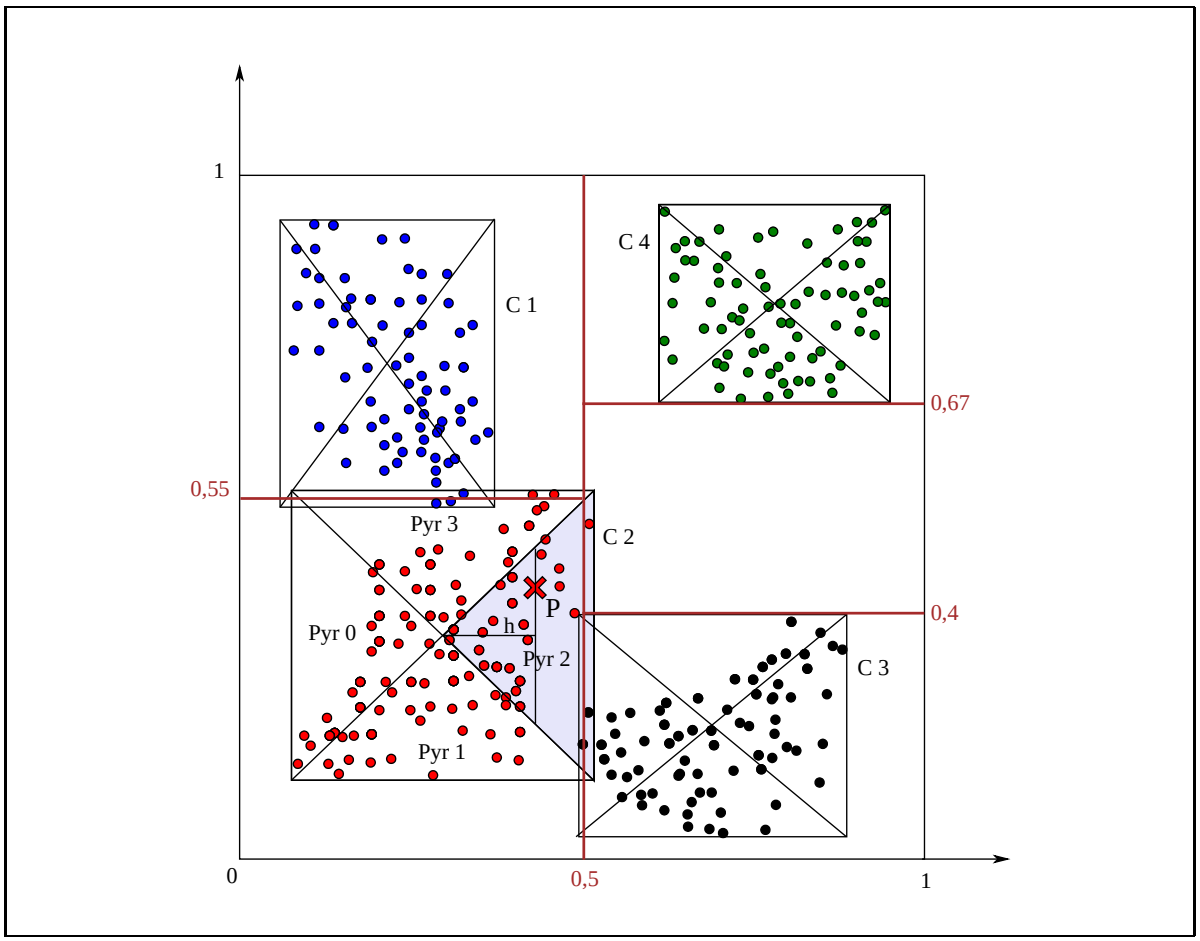

FIG. 3.2 – Représentation géométrique de *Kpyr*

deux dimensions sont $P(0,43;0,4)$. Il se situe dans la classe C_2. Il subit donc le changement de base de cette classe et devient $P_{C_2}(0,8;0,7)$. Nous pouvons alors déterminer que P appartient à la pyramide $Pyr2$ de la classe C_2. Dans la pyramide $Pyr2$, le point P se situe à une hauteur h égale à 0,3 par rapport au sommet de la pyramide. Nous pouvons en déduire la valeur de l'index I_P du vecteur P.

$I_P = pyramide + h(sommetPyramide) = 2 + 0,3 = 2,3$

Le vecteur P, comme l'ensemble des autres vecteurs de l'espace, reçoit un index uni-dimensionnel. Cet index représente l'emplacement du point P dans l'arbre B+ de la classe C_2.

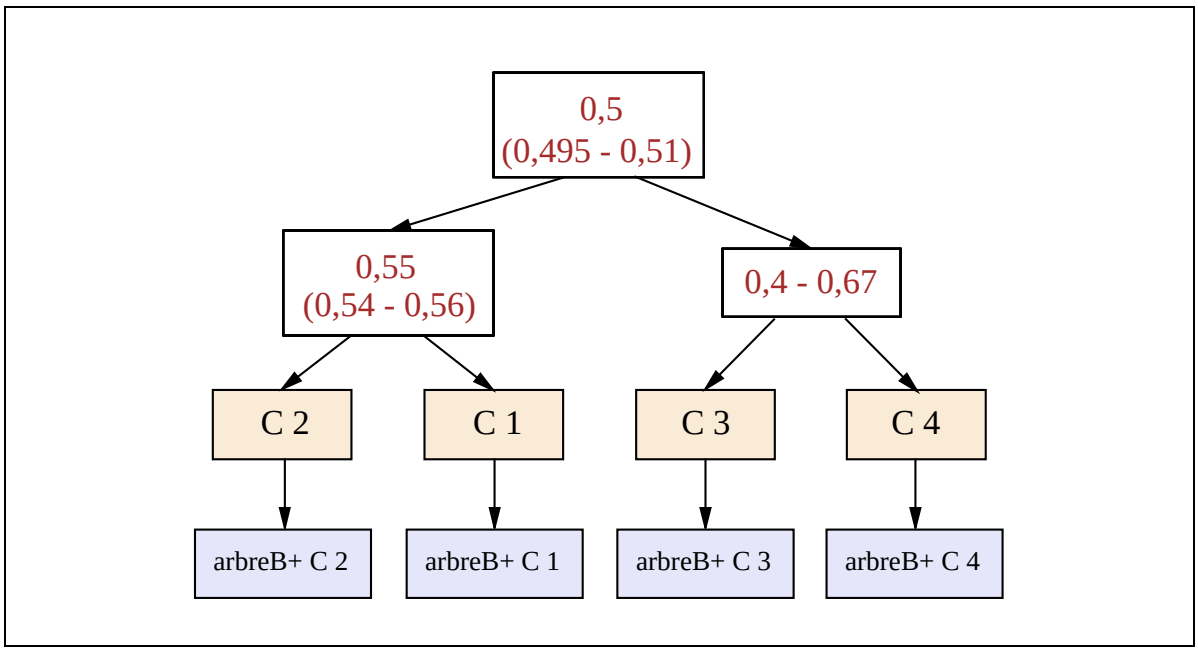

FIG. 3.3 – Exemple d'un arbre Espace

Pour terminer la construction de notre structure d'indexation optimisée, un "arbre Espace" est créé à partir des K arbres B+ obtenus représentant les classes C_i, $0 \leq i < K$. Cet "arbre Espace" est un arbre binaire (voir figure 3.3), ses feuilles pointent vers les K arbres B+ et les autres noeuds sont les meilleures frontières entre les régions des classes. Le but d'une telle structure arborescente est d'optimiser l'accès aux bonnes classes lors d'une recherche. La profondeur de l'arbre Espace est calculée en fonction du nombre de classes obtenues par l'algorithme de classification et de la répartition de ces dernières dans l'espace. Nous obtenons K classes par l'algorithme de classification, ce qui permet de construire un arbre Espace binaire d'une profondeur variant entre $\log_2(K)$ et K.

L'algorithme de création de l'arbre Espace se base sur un *phénomène d'occultation* inter classes que nous détaillons avec précision dans la section 4.1.2. Ce phénomène prouve qu'il existe une probabilité de trouver un espace vide dans l'une des dimensions entre deux classes "consécutives". Nous montrons dans la section 4.1.2 que plus le nombre de dimensions augmente, plus la probabilité d'avoir un espace vide est forte. Nous utilisons ces espaces vides

pour déterminer nos frontières inter classes et créer notre arbre Espace binaire. Si aucun espace vide n'est visible alors nous déterminons l'espace contenant le moins de données possible au milieu de deux classes consécutives. L'algorithme est composé de quatre grandes étapes :

1. Tant que chaque classe n'est pas seule dans une région délimitée par les frontières, alors l'algorithme recherche l'espace vide entre deux classes consécutives le plus grand pour chaque dimension ; sinon étape 4 ;
2. Les noeuds "frontière" sont créés. Deux cas possibles : soit il existe un plus grand espace vide pour une dimension choisie, dans ce cas seules les deux limites de cet espace vide dans la dimension choisie sont conservées dans le noeud. Notons que ces deux limites sont les bordures dans la même dimension des rectangles englobants minimums (REMs) des deux classes bordant l'espace vide ; soit il n'existe aucun espace vide, alors une seule valeur frontière est choisie, au milieu de deux classes consécutives pour une dimension à l'endroit le moins dense. Néanmoins les limites des bordures des rectangles englobants minimums dans cette dimension sont conservées dans le même noeud pour certains cas spéciaux de requête ;
3. Les classes sont réparties dans les nouvelles régions de l'espace créées, ensuite retour à l'étape 1 ;
4. Les feuilles de l'arbre Espace sont créées pointant vers les arbres B+ de chaque classe.

Notre exemple proposé figure 3.3 montre l'arbre binaire créé à partir des 4 classes de la figure 3.2. Le choix de la première frontière est problématique. En effet, il n'existe aucun espace vide entre les classes consécutives, c'est à dire entre les classes ordonnées C_1, C_2, C_3 et C_4 pour la dimension x et C_3, C_2, C_1 et C_4 pour la dimension y. nous choisissons donc l'espace le moins dense qui s'avère être l'espace entre les classes consécutives C_2 et C_3 sur x. Un premier noeud frontière est créé dans notre arbre Espace avec pour valeur 0,5, le noeud garde en mémoire les deux limites (0,495 - 0,51) des REMs des classes concernées. Ensuite, il faut une autre frontière pour séparer les classes C_1 et C_2 ainsi que pour C_3 et C_4. La frontière entre C_1 et C_2 est une nouvelle fois problématique, l'espace le moins dense est celui entre ces classes sur y, le noeud aura la valeur 0,55 et les valeurs (0,54 - 0,56). Pour séparer C_3 et C_4, un espace vide existe, les deux limites, 0,4 et 0,67, de cet espace vide servent à délimiter

les régions contenant les deux classes. L'algorithme est alors terminé pour notre exemple car chaque classe C_i, $0 \leq i < K$, est isolée dans une région délimitée par l'arbre Espace.

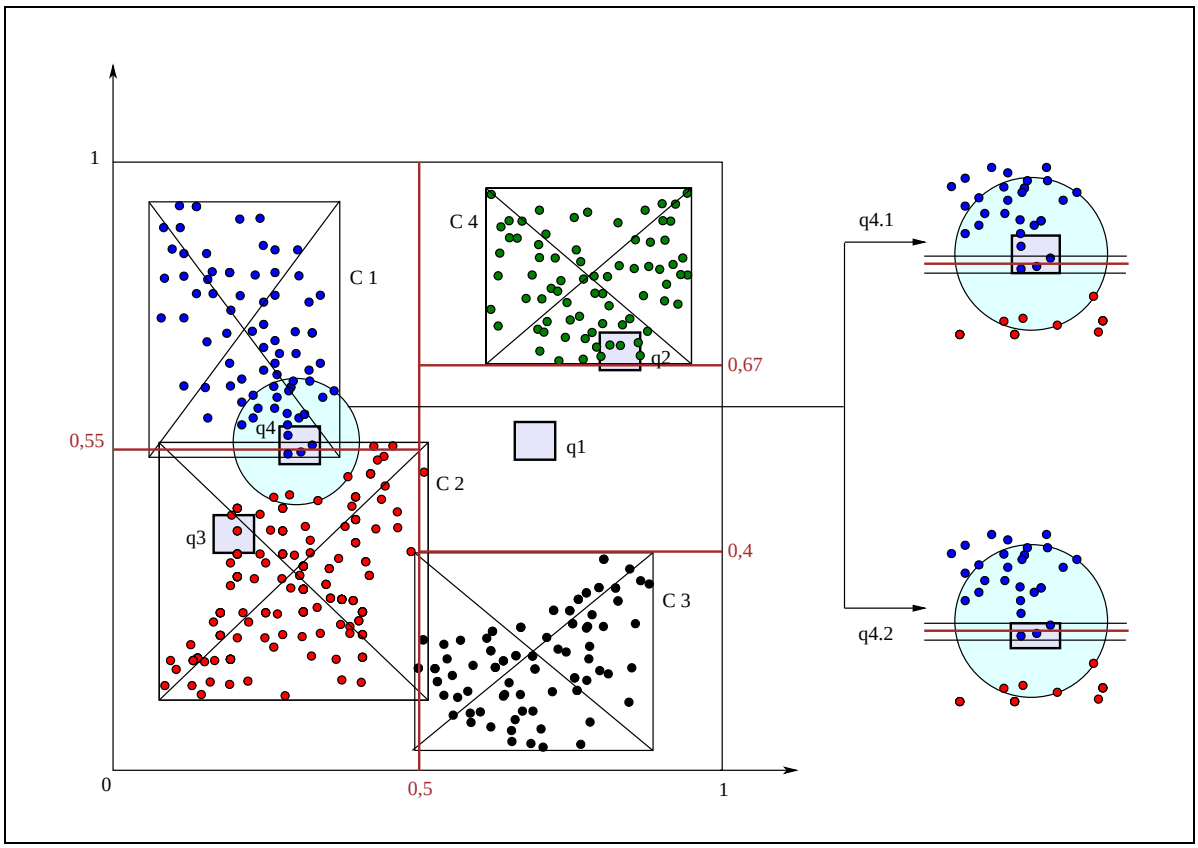

FIG. 3.4 – Exemple de requêtes dans Kpyr

Deux types de recherche exacte existent, comme nous l'avons détaillé section 2.3. La recherche par fenêtrage est pour notre structure d'indexation, la recherche la plus rapide car elle est très bien adaptée à l'accès à l'information dans un arbre B+. La recherche des plus proches voisins est plus lente pour un même nombre de résultats retournés. Cette recherche peut être vue, dans notre cas, comme plusieurs recherches par fenêtrage pour lesquelles la taille de la fenêtre augmente petit à petit afin d'obtenir les plus proches voisins. Nous présentons les modifications nécessaires pour obtenir notre algorithme de recherche des plus proches voisins à partir de la recherche par fenêtrage dont les trois grandes étapes sont présentées ci dessous :

1. A partir d'une requête par fenêtrage q, l'algorithme détermine la, ou les, régions concernées grâce à notre "arbre Espace". Les classes à atteindre par la requête sont alors déduites. Plusieurs cas possibles sont détaillés plus loin par l'exemple.
2. La requête obtenue charge en mémoire les arbres B+ de chaque classe concernée. Ensuite, grâce aux coordonnées géométriques de la requête par fenêtrage q, les feuilles à accéder de chaque arbre B+ sont déterminées. Cette action retourne un ensemble restreint de résultats possibles.
3. Les résultats possibles sont ensuite analysés un par un pour déterminer s'ils correspondent à la requête q.

Notre algorithme de recherche des k plus proches voisins d'une requête q se décompose ainsi :

1. tant que les k plus proches voisins n'ont pas été trouvés, l'algorithme de fenêtrage est utilisé avec un ε plus grand ;
2. les k plus proches voisins de q accédés par la requête par fenêtrage sont conservés en mémoire. Les bornes des index à cette étape sont conservées pour éviter des accès supplémentaires aux données lors de l'itération suivante ;
3. si la distance du $k^{\text{ième}}$ plus proche voisin est inférieure à la largeur de la requête par fenêtrage alors l'algorithme est fini sinon retour à l'étape 1.

Par nature récursive, cet algorithme de recherche des plus proches voisins est plus lent qu'une recherche par fenêtrage qui n'accède qu'une seule fois à la structure d'indexation. Quelque soit l'algorithme de recherche choisi, différentes situations de requête peuvent être observées. Nous présentons quelques cas de requêtes possibles, représentés sur la figure 3.4. Les requêtes $q1$ et $q3$ ne posent pas de problème : $q1$ n'atteint pas de classe, l'arbre Espace le détermine directement grâce aux différentes valeurs gardées en mémoire dans les noeuds sans devoir interroger les arbres B+ et $q3$ est complètement à l'intérieur de la classe C_2. La requête $q2$ se situe sur une limite d'un espace vide et d'une classe, elle est alors redimensionnée à la limite de la classe C_4. Enfin la requête $q4$ se situe sur une frontière déterminée par faible densité, la requête est alors divisée en deux et chaque partie de $q4$, $q4.1$ et $q4.2$ est redimensionnée en fonction des limites des REMs gardées en mémoire (voir figure 3.4 à droite). Pour simplifier

la compréhension de notre exemple, nous utiliserons la requête $q3$ à l'intérieur de la classe C_2. Géométriquement la requête $q3$ accède aux points compris dans les tranches pyramides grisées sur la figure 3.5 à droite. Les données appartenant à ces tranches sont analysées et celles comprises dans la requête $q3$ sont conservées comme résultats.

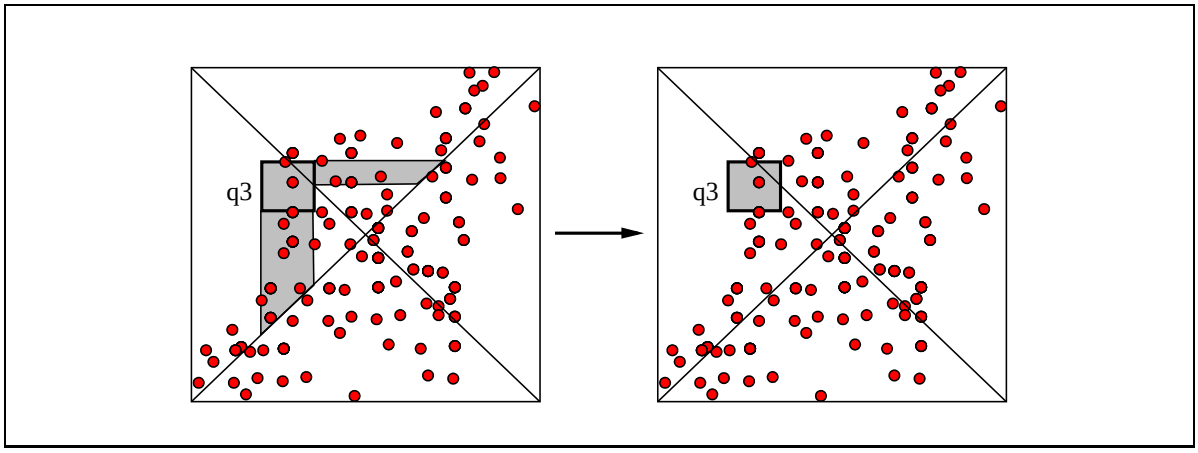

FIG. 3.5 – Exemple de la requête $q3$ dans une pyramide

La contribution de notre approche est de réunir de bonnes conditions d'utilisation de la technique de la Pyramide :

- L'indexation résultante est plus efficace. En effet, le nombre de données et l'espace à indexer par la technique Pyramidale est réduit. Les classes sont homogènes, leur centre est le sommet de chaque pyramide, un point référence pour les données de la classe.
- Le calcul des frontières s'effectue après la classification, ce qui donne une meilleure division de l'espace. Ces frontières permettent de déterminer les classes atteintes par une requête, elles jouent donc un rôle important dans la réduction des accès mémoire et par conséquent dans la réduction du temps de réponse.

3.2.2 Expérimentations

Les expérimentations menées ont pour but de comparer notre structure d'indexation Kpyr à d'autres structures de l'état de l'art. L'implémentation de Kpyr a été réalisée en langage Java.

L'évaluation des performances est faite sur un ordinateur AMD 3Ghz avec 1Go de RAM. Nous avons évalué les performances de notre approche à partir d'une base de données réelles de documents vidéo. Chaque séquence vidéo de notre base est transformée en un vecteur multidimensionnel selon le modèle de représentation vectoriel présenté dans la section 6.1. Ensuite, nous enrichissons l'ensemble des vecteurs obtenu avec des données synthétiques selon la distribution des données réelles.

Les structures d'indexation multidimensionnelles sont donc évaluées sur des bases de vecteurs multidimensionnels. Les jeux de données créés sont stockés en mémoire pour évaluer les performances des différentes structures en utilisant une moyenne de plusieurs requêtes. Les requêtes utilisées sont aléatoirement issues des vecteurs multidimensionnels représentant les séquences vidéo de notre base. L'ensemble des requêtes est identique pour chaque structure d'indexation. Nous avons mesuré dans nos résultats le temps de réponse total pour une requête par fenêtrage sous différentes caractéristiques de la base de données : la taille de la base de données, la dimension des données et la sélectivité.

Nous proposons dans nos expérimentations d'utiliser la *sélectivité* d'une requête, paramètre fixé par l'utilisateur que nous définissons comme la largeur de la fenêtre requête par rapport à l'espace de données normalisé. Par exemple, pour une sélectivité égale à $0,02$ à partir d'une requête $\mathbf{q}(q_1,...,q_d) \in \mathbb{R}^d$ dans un espace normalisé, la recherche retourne l'ensemble des données comprises dans la requête par fenêtrage $\mathbf{Wq}([q_1-0.01\ ;\ q_1+0.01]\ ,\ ...\ ,\ [q_d-0.01\ ;\ q_d+0.01])$. L'ensemble des données comprises dans $\mathbf{Wq}$ est retourné comme résultat puis ordonné en fonction de la distance de chacune des données à la requête $\mathbf{q}$.

Les expériences montrent qu'en général Kpyr présente de meilleurs temps de réponse que P+Tree, la technique de la pyramide et la recherche séquentielle. Un exemple est présenté figure 3.6 pour 500 000 données de 50 dimensions, dont la distribution est hétérogène et répartie en 4 classes. Pour des requêtes à large sélectivité, la recherche séquentielle devient plus rapide. En revanche, ces requêtes ne correspondent à aucun besoin réel de recherche d'information. Les résultats sont trop nombreux et non pertinents par manque de précision dans la sélectivité (20% des données de la base retournées comme résultats). Nous avons aussi testé notre méthode avec des recherches par plus proches voisins. Les résultats obtenus montrent que le temps de recherche est en moyenne triplé entre une recherche par fenêtrage qui retourne k

résultats et la recherche des *k* plus proches voisins. Il est aussi intéressant de noter que les performances de la recherche séquentielle des plus proches voisins sont moins bonnes que pour Kpyr avec un nombre *k* inférieur à 10% de la base de données.

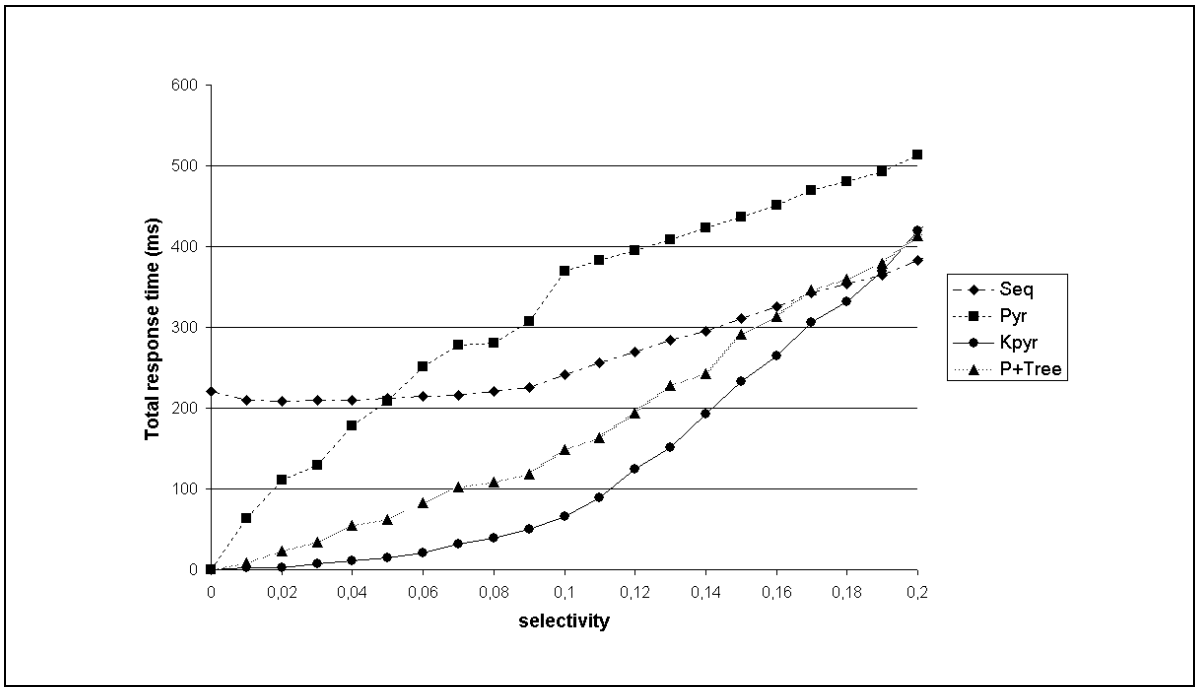

FIG. 3.6 – Effet de la sélectivité avec 500 000 données

Par ailleurs, nous avons obtenu des résultats particulièrement intéressants répondant aux questions suivantes : le temps de réponse est-il affecté par la quantité de données analysées lors d'une recherche ? Est-il affecté par le nombre de classes atteintes ?

La figure 3.7 montre l'évolution du temps de réponse en fonction de variations légères de la sélectivité pour 500 000 données et 8 classes. Nous remarquons que les courbes de la recherche séquentielle et de Kpyr sont linéaires, alors que celles de la technique de la Pyramide et de P+Tree montrent quelques augmentations significatives du temps de réponse pour certaines valeurs de sélectivité. Nous expliquons ces augmentations importantes par un accès inutile à un surplus de données. Par exemple, pour une sélectivité de 0,06, le temps de réponse de la pyramide augmente de 50%. La requête par fenêtrage a certainement atteint un hyperplan de la pyramide contenant un grand nombre de données ne répondant pas à la requête demandée. Ce

phénomène s'accentue pour la méthode de la Pyramide de part la répartition très hétérogène des données, il est moins important pour la méthode P+Tree. Quant à notre méthode, elle n'est pas affectée par ce problème puisque la courbe de résultats de Kpyr croît linéairement en fonction de la sélectivité.

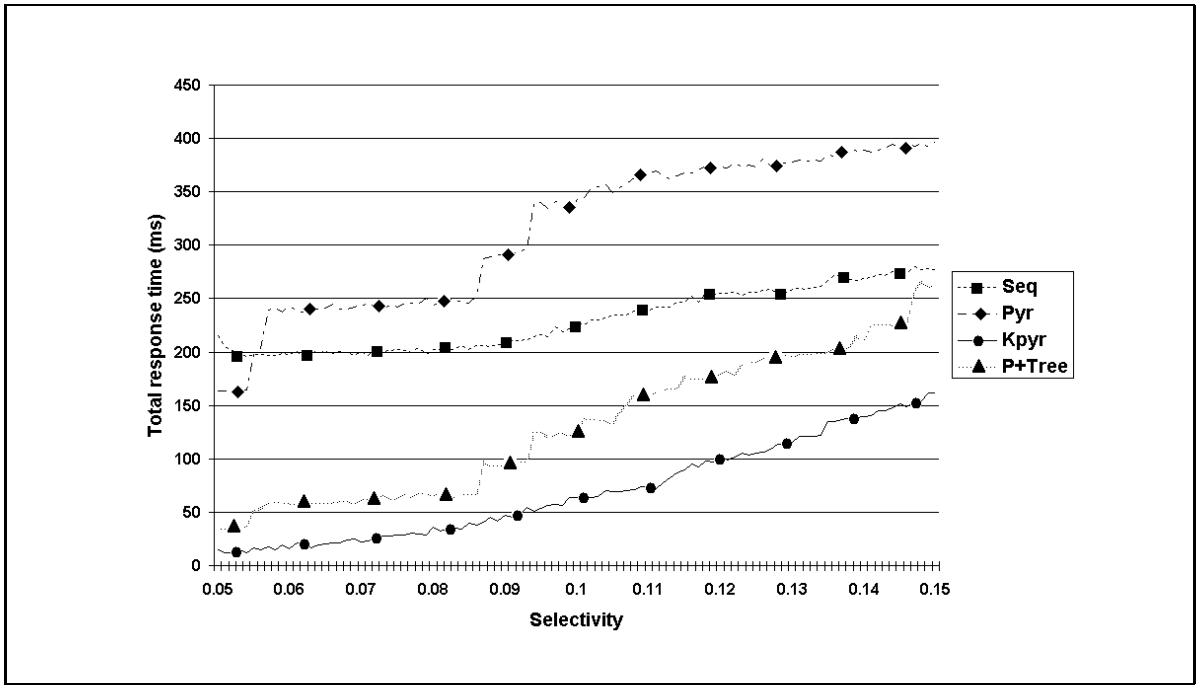

FIG. 3.7 – Effet de la sélectivité (précise) avec 500 000 données et 8 classes

Pour mieux comprendre ce phénomène, nous montrons, dans la figure 3.8, l'évolution du nombre de données accédées et du nombre de classes atteintes en fonction de la sélectivité pour les méthodes P+Tree et Kpyr. Nous remarquons qu'entre les valeurs de sélectivité 0,06 et 0,08, le nombre de classes atteintes par P+Tree augmente, alors que le nombre de données croit linéairement comme le temps de réponse de la figure 3.7. Des phénomènes similaires peuvent être observés pour notre approche. Nous en déduisons que l'effet du nombre de classes atteintes est négligeable par rapport au nombre de données accédées. Ces courbes nous montrent que notre méthode Kpyr accède à beaucoup moins de données pour une même requête que P+Tree, ce qui explique en partie le gain en temps de notre méthode par rapport à P+Tree.

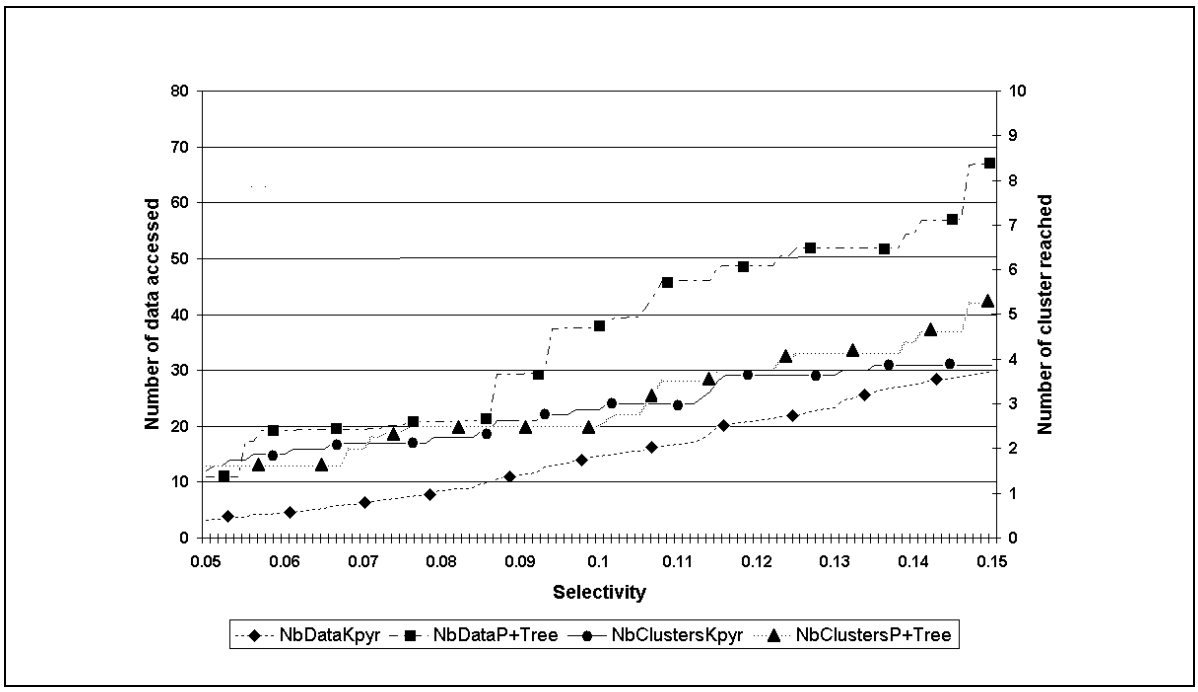

FIG. 3.8 – Effet de la sélectivité (précise) en fonction du nombre de données accédées et du nombre de classes atteintes

3.2.3 Conclusions

Nous retenons en conclusion des expérimentations réalisées sur Kpyr :

1. Kpyr a en général de meilleurs résultats comparés à ceux des autres méthodes présentées ;
2. les performances de Kpyr (et des autres méthodes également) sont plus affectées par le nombre de données traitées lors d'une requête. Le fait d'accéder à un grand nombre de données inutiles lors d'une requête dégrade considérablement les temps de réponse.

Le principe de l'indexation par la technique de la pyramide ne tient aucunement compte du surplus de données inutiles à la requête mais qui sont tout de même traitées. La recherche dans notre arbre Espace binaire et dans les arbres B+ concernés a un coût asymptotique de $O(\log N)$, où N est le nombre total d'index dans les arbres B+, ce qui est équivalent au nombre de données de notre base. Cette recherche est donc rapide mais retourne un ensemble de résul-

tats possibles, à analyser ensuite pour savoir si ces résultats répondent réellement à la requête. Cette analyse a un coût de $O(d \times N_p)$, où d est le nombre de dimensions et N_p est le nombre de résultats possibles, qui dans le pire des cas vaut N. De ce fait, nous nous sommes concentrés sur la possibilité de réduire considérablement l'ensemble des résultats possibles à analyser afin de retourner plus rapidement les résultats finaux à la requête. Nous avons réduit, avec notre première approche Kpyr, ce nombre en proposant une classification à priori des données, comme le montre la figure 3.9.

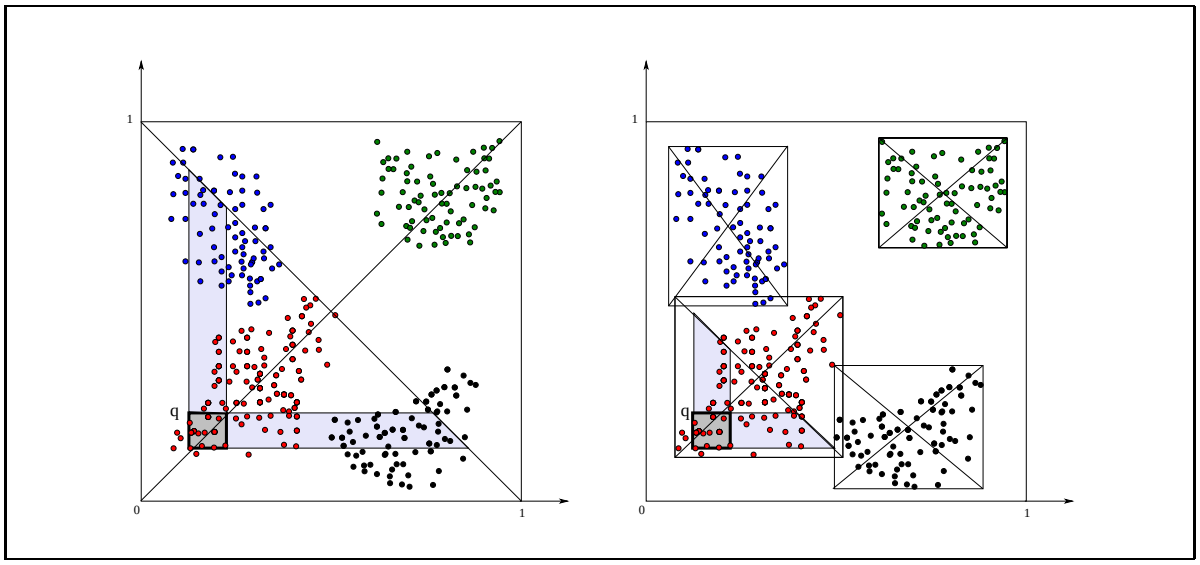

FIG. 3.9 – Problème d'accès inutiles aux données

Néanmoins, ce problème persiste et s'aggrave dans les espaces de dimension élevée. En effet, plus la dimension augmente plus les tranches qui sont proches de la base des pyramides sont volumineuses même après classification de la base de données. Une requête placée proche de ces tranches accéde à un nombre trop important de données inutiles. Nous présentons une deuxième méthode d'indexation *KpyrRec* qui a pour but principal d'atténuer ce phénomène.

3.3 KpyrRec

Nous présentons dans cette section KpyrRec, une structure d'indexation multidimensionnelle basée sur les conclusions de Kpyr. La différence fondamentale se situe au niveau de l'algorithme d'indexation. En effet, après avoir obtenu la classification de k-means, nous proposons une nouvelle méthode *PyrRec* effectuant un découpage différent de l'espace diminuant ainsi le nombre d'accès inutiles à des données lors d'une recherche.

3.3.1 PyrRec : l'idée de la division de l'espace

Une étude approfondie de la technique de la pyramide nous a conduit à réfléchir sur la façon de réunir les conditions favorables à une meilleure indexation. Nous proposons donc notre méthode PyrRec qui organise efficacement l'espace en sous espaces homogènes et introduit une nouvelle structure qui gère ces espaces.

L'idée principale de PyrRec est de faire un découpage récursif de l'espace en tranches de pyramide de volume égal par niveau de récurrence. L'algorithme de *PyrRec* se divise en plusieurs étapes :

1. un découpage de l'espace en $2 \times d$ pyramides est appliqué, où d est le nombre de dimensions de l'espace ;
2. chaque pyramide est ensuite découpée ensuite en Nt tranches de volume égal ; ces tranches sont déterminées en calculant les hauteurs respectives h_i et h_{i+1} de la pyramide.

$$h_i = \frac{h * (\sqrt[d]{i} - \sqrt[d]{i-1})}{\sqrt[d]{Nt}}, \tag{3.1}$$

   avec $1 \leq i < Nt$, d : nombre de dimensions, h : hauteur des pyramides
3. l'algorithme effectue une récurrence du découpage sur les tranches contenant un nombre de données supérieur à un seuil fixé λ. Un changement de base est alors effectué pour obtenir un espace de dimension $d-1$, la dimension permettant de déterminer la tranche étant supprimée ; puis retour à l'étape 1 pour effectuer la récurrence si nécessaire.

En général, le nombre de récurrences ne dépasse pas 3 même avec un petit seuil λ. Par

exemple, pour 1.000.000 données de dimension 100, Nt=10, nous avons en moyenne 5.000 données par pyramide et donc 500 données par tranche de pyramide. Avec le seuil λ égal à 10, une première récurrence s'effectue sur la plupart des tranches. Ensuite selon la répartition des données, d'autres récurrences sont nécessaires pour les régions de l'espace de forte densité.

Le nombre de récurrence est en fonction du nombre de données de la base et des paramètres Nt et λ. Un nombre de tranches Nt par pyramide élevé diminue en moyenne le nombre de données par tranche et par conséquent, diminue la probabilité d'avoir une récurrence dans chaque tranche. Le paramètre λ permet de contrôler le nombre de récurrences ; une valeur élevée de λ diminue considérablement le nombre de récurrences, et par conséquent, le nombre de sous-arbres B+ dans la structure d'indexation. Il permet ainsi de contrôler la taille mémoire requise par la structure.

Il est difficile d'évaluer une bonne valeur de seuil λ, nos différents tests pour fixer cette valeur λ ont montré des résultats similaires pour $\lambda \in [1,20]$, oscillant ainsi le nombre de récurrences moyen entre 2 et 3 (i.e. 3 ou 4 index par point) selon la base de données utilisée. Une valeur λ élevée réduit trop le nombre de récurrences, les performances ressemblent alors à celles de Kpyr. Dans les expérimentations réalisées et présentées à la fin de ce chapitre, nous avons fixé la valeur de λ à 10. Notre principal objectif est d'observer le gain en temps de recherche apportée par une structure d'indexation récursive.

La figure 3.10 nous montre un exemple d'utilisation d'une récurrence. Le point $P1$ est dans une tranche qui ne dépasse pas le seuil fixé du nombre de données pour effectuer une nouvelle récurrence. Le point $P1$ aura le même index que toutes les autres données comprises dans cette tranche, donc au maximum un nombre égal au seuil λ. $P2$ et $P3$ sont dans une tranche qui subit une récurrence car le seuil λ est dépassé. L'algorithme de découpage en tranches de pyramide s'effectue une seconde fois dans un espace de dimension $d-1$ (partie droite de la figure 3.10), nous observons alors que les points $P2$ et $P3$ n'appartiennent pas à la même pyramide dans ce nouvel espace. Ils seront donc indexés avec des valeurs différentes dans notre arbre B+.

L'algorithme PyrRec requiert la construction d'un arbre B+ approprié pour ranger des index contenant les informations des récurrences. Pour construire l'arbre B+ contenant les index des points de l'espace de données, PyrRec procède comme suit :

1. Pour chaque point p_i, $0 \leq i < N$, la pyramide cp_i qui contient le point est déterminée en

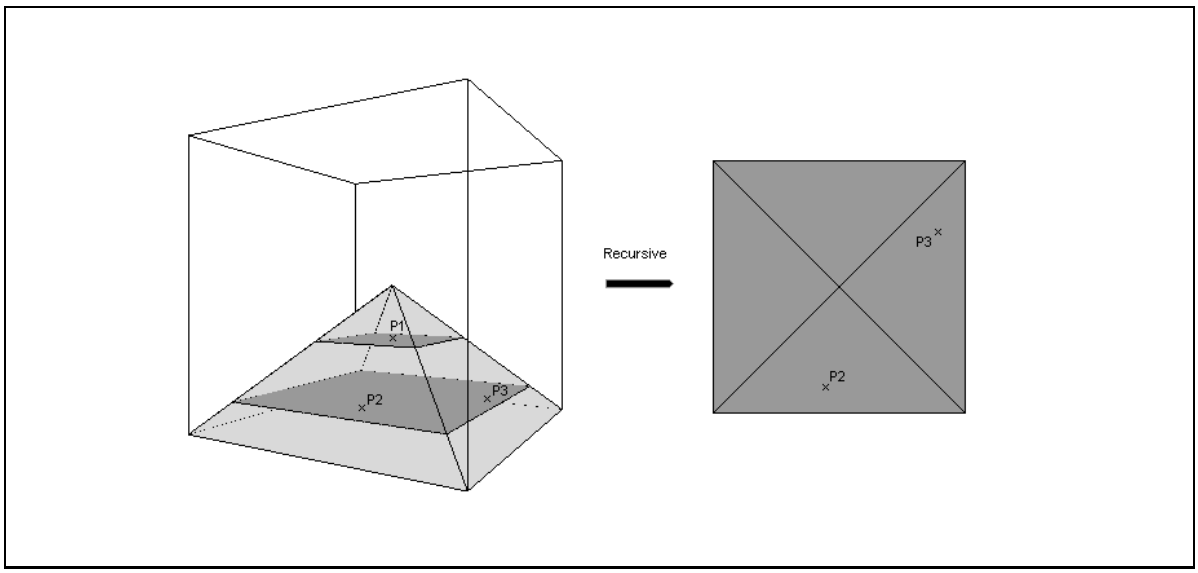

FIG. 3.10 – Représentation géométrique de *PyrRec*

se basant sur la distance minimum entre le point et les centres des bases des pyramides.

2. Ensuite, la hauteur du point p_i par rapport au sommet de sa pyramide détermine la tranche tp_i de la pyramide le contenant. A partir des valeurs de cp_i et tp_i, un premier index est créé pour le point p_i.

3. L'ensemble des points de l'espace reçoit un premier index ; les points appartenant à la même tranche ont alors le même index.

4. Pour chaque tranche contenant des points, un noeud dans l'arbre B+ est créé avec l'index des points de la tranche. Si le nombre de points d'une tranche dépasse le seuil λ alors une récurrence est effectuée sur le nouvel espace de la tranche concernée : retour à l'étape 1.

L'exemple de la figure 3.11 montre une partie de la construction de l'arbre B+ de notre algorithme PyrRec avant d'effectuer une récurrence. Nous observons par exemple que la tranche 2 de la pyramide 1 (index 1,2) ne contient que 3 points, donc inférieur au seuil λ, ce qui implique que cette tranche ne nécessite pas d'être affinée par une récurrence de notre algorithme. Les tranches indexées 8,7 et 20,5 contiennent 50 et 25.000 points, au moins une récurrence est alors nécessaire. La figure 3.11 montre la suite de la construction de notre arbre B. Les

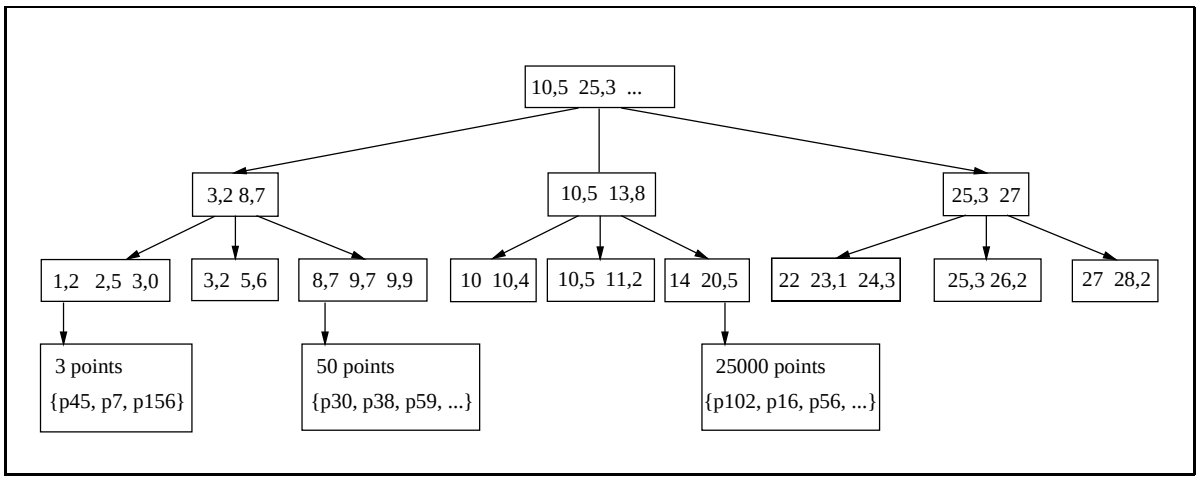

FIG. 3.11 – Construction de l'arbre B+ de PyrRec : 1ère récurrence

50 points de la tranche 7,8 ont été répartis dans de nouvelles feuilles de l'arbre B+ après la récursion. La tranche 20,5 aura nécessité une deuxième récurrence pour 2 tranches créées par la première récurrence pour répartir l'ensemble de ces points dans des feuilles de l'arbre B+.

A la fin de l'algorithme de PyrRec, nous obtenons un arbre B+, composé au moins de $\frac{N}{\lambda}$ index (chaque index contient λ points) et au plus de N index (un point par index), N étant le nombre total de points. La hauteur de cet arbre peut varier selon le nombre de récurrences à faire pour indexer les points. Notre structure peut être définie comme une composition d'arbres B+ selon la répartition des données dans l'espace.

3.3.2 Comparaison entre PyrRec et technique de la pyramide

Afin de justifier l'amélioration apportée par notre structure PyrRec face à la technique de la pyramide, nous avons comparé les deux méthodes. Nous présentons dans un premier temps une visualisation graphique du gain de l'espace des données retournées comme résultats possibles. Puis nous donnons une comparaison des coûts des algorithmes de recherche dans les deux structures concernées. Enfin nous prouvons par des résultats expérimentaux les performances de notre structure PyrRec.

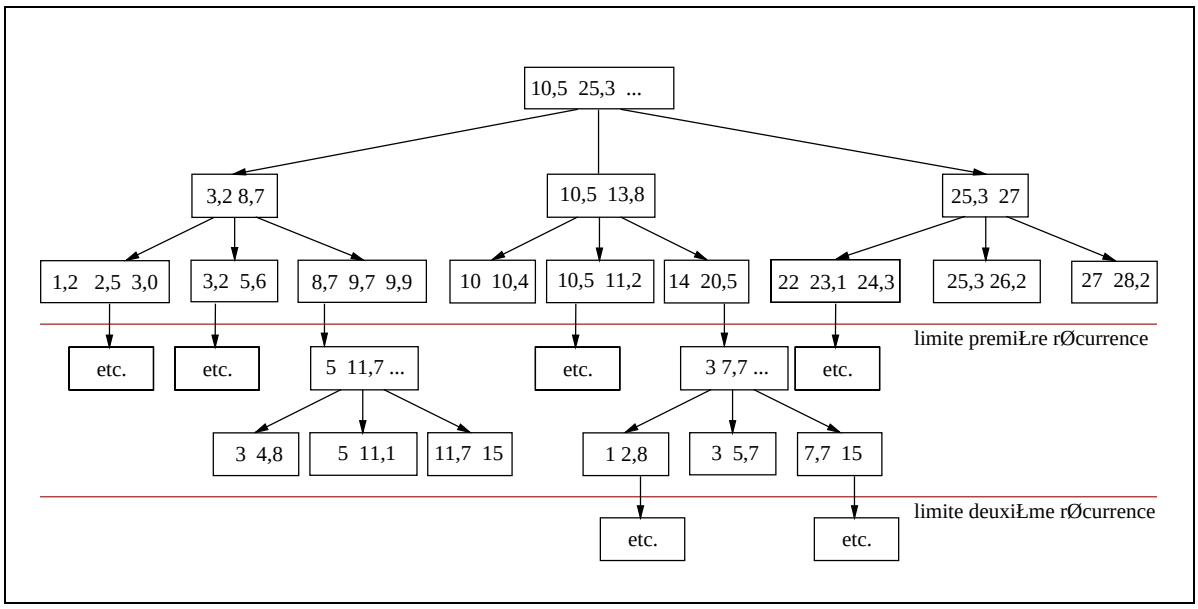

FIG. 3.12 – Construction de l'arbre B+ de PyrRec : récurrences suivantes

3.3.2.1 Espace de données accédées

L'idée principale de la création de PyrRec est venue du problème d'accès inutiles à un volume trop important de données, décrit dans la section 3.2.3. Nous montrons sur la figure 3.13 un exemple simple (par souci de présentation) de requête q dans la classe C_2. Nous avons représenté la figure en 3D afin de pouvoir constater l'utilité d'une récurrence lors de la requête. La technique de la pyramide retourne tous les points compris entre la hauteur minimale et maximale de la requête q dans la dimension de la pyramide. Pour la même requête dans la structure PyrRec (à droite), nous accédons aux tranches de pyramides (représentées par un ou plusieurs noeuds dans notre structure arborescente) contenant la requête q puis nous retournons les points compris dans les tranches encadrant la requête dans la ou les pyramides récurrentes. Nous pouvons constater visuellement la différence entre les régions de l'espace accédées (en rouge ou gris foncé sur la figure 3.13) par les deux méthodes pour une même requête.

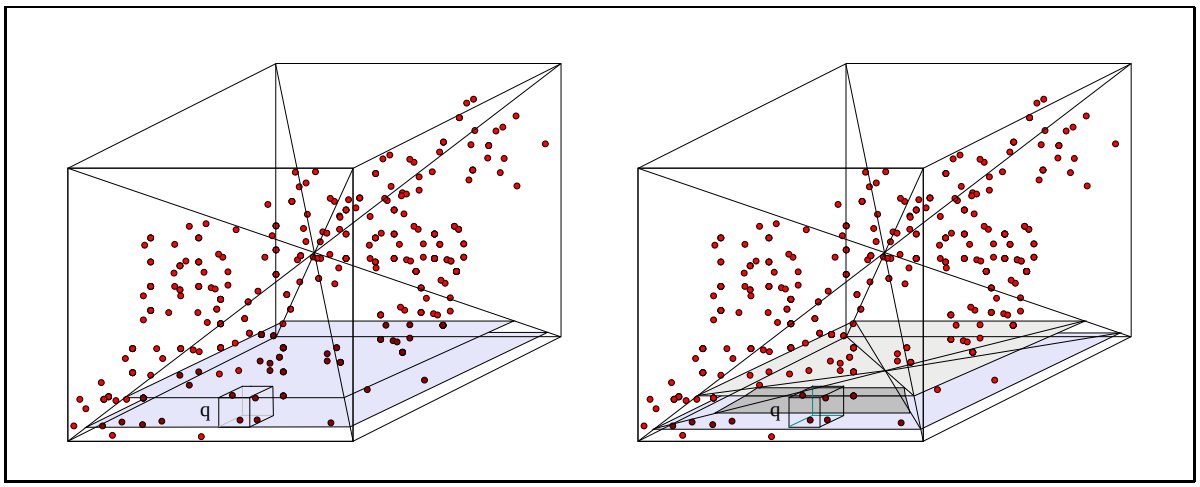

FIG. 3.13 – Comparaison des espaces accédés par une requête q pour les deux méthodes : à gauche, par la technique de la pyramide et à droite par PyrRec

3.3.2.2 Coût d'une recherche et coût mémoire

Nous proposons dans cette section de comparer dans un premier temps les coûts asymptotiques d'une recherche par la technique de la pyramide et par notre algorithme PyrRec. Ensuite, nous nous intéressons à l'espace mémoire occupé par ces structures d'indexation.

La recherche dans la structure d'indexation créée par la technique de la pyramide, que nous utilisons dans Kpyr, peut se simplifier à l'accès aux feuilles comprises entre une valeur minimale et une maximale. Il est possible que cette recherche soit située sur une limite de pyramide et nécessite une division en deux parties. Il faut alors trouver deux valeurs minimales dans l'arbre B+ et ensuite se déplacer de feuilles en feuilles jusqu'aux valeurs maximales. Pour construire un arbre B+ il est nécessaire de fixer une valeur f, *fanout* de l'arbre B+, représentant le nombre maximal d'index que peut contenir un noeud ou une feuille de l'arbre. Dans la figure 2.6, le fanout de l'arbre présenté est de 3. Avec la valeur f de l'arbre et le nombre de données à indexer N, la recherche d'un index a un coût asymptotique de $O(\log_f N)$.

Pour notre algorithme PyrRec, la structure arborescente utilisée se base sur le principe de construction de l'arbre B+ pour chaque récurrence. En considérant le même fanout f que dans l'exemple précédent, nous pouvons donner un coût asymptotique de $O(\log_f I_r)$, où I_r

est le nombre d'index de la récurrence en cours. Pour une récurrence, nous avons une valeur maximale de $I_r = 2 \times d * nt$ avec d la dimension des données et nt le nombre de tranches utilisées pour découper les pyramides. Pour approximer les calculs, nous ne tenons pas compte du fait que la dimension de l'espace diminue lorsque nous effectuons une récurrence. Toute tranche indexée dans l'arbre B+ doit contenir au minimum un point. Le coût asymptotique d'une recherche dans notre structure d'indexation est donc égale à $O((n_r + 1) \times \log_f I_r)$, où n_r est le nombre maximal de récurrences effectuées. Nous avons montré théoriquement que la valeur de n_r ne dépassait pas 2, d'où un coût maximal égal à $O(3\log_f I_r)$.

Avec la comparaison des deux coûts de recherche, nous pouvons facilement en déduire que pour l'accès aux index des points à retourner comme résultats possibles, la technique de la pyramide est plus rapide que PyrRec. Cette différence est négligeable par rapport à celle du volume de données retournées à analyser pour savoir si réellement elles répondent à la requête. Cette analyse se fait en $O(R_p)$, où R_p est le nombre des données retournées comme résultat possible par la structure arborescente. Comparer théoriquement R_p est très difficile, ce nombre dépend fortement de la répartition des données dans l'espace et de l'emplacement de la requête. C'est pourquoi, juste après avoir détaillé les différences de coût mémoire, nous proposons une étude expérimentale comparant le temps de réponse des deux méthodes pour une même requête.

Les structures arborescentes utilisées par les deux méthodes comparées utilisent la même structure pour les noeuds et les feuilles de l'arbre. Chaque noeud (non feuille) contient plusieurs index selon la valeur du fanout de l'arbre f. Les noeuds feuille contiennent f index. Dans le cas de la technique de la pyramide, les index sont associés aux identifiants des données. Pour notre structure, les index des feuilles représentent la tranche d'une pyramide contenant entre 1 et λ identifiants de point. Au final, nous gardons en mémoire dans les deux structures les N identifiants des N points de notre base de données complète. La différence ne se fait donc pas sur ces identifiants de points mais sur le nombre d'index et surtout leur taille. En effet, le nombre d'index est compris entre $\frac{N}{\lambda}$ et N. Dans le pire des cas, nous avons besoin d'autant d'index que la technique de la pyramide. En considérant un fanout équivalent pour les deux structures, le nombre de noeuds et de feuilles à utiliser est inférieur pour notre structure. La plus grande différence se fait au niveau de la taille des index. Par exemple, pour 100 dimen-

sions (donc 200 pyramides) et 10 tranches par pyramide, l'index le plus élevé sera 199,9. Nous avons utilisé cette forme d'index dans ce chapitre pour aider le lecteur à la compréhension. Cependant cette forme peut être facilement transformée en "short" (16 octets en java) par une multiplication par 10, ce qui donne des index compris entre 0 et 1999. La technique de la pyramide utilise des index de type "float" ou "double"(32 ou 64 octets en java). Pour N= 1.000.000, la différence en mémoire est immédiate, du simple au double (voir quadruple).

3.3.2.3 Comparaison expérimentale

Afin de confirmer les avantages apportés par notre structure PyrRec par rapport à la technique de la pyramide, nous avons testé expérimentalement les deux méthodes séparément et avons évalué le gain en temps pour des recherches sur un ensemble de données homogènes. La figure 3.14 présente le temps de réponse total en fonction de la sélectivité pour 1 000 000 données de dimension 80. Pour information, PyrRec a utilisé, pour ces courbes de résultats, une seule récurrence au maximum ; la répartition des données étant homogène, aucune région n'a été assez dense pour effectuer une deuxième récurrence. La réduction du temps de réponse est assez significative, allant jusqu'à 50% pour certaines valeurs de sélectivité. Ce qui montre notre intérêt à remplacer la technique de la pyramide par PyrRec dans l'algorithme KPyrRec.

3.3.3 Algorithmes d'indexation et de recherche de KpyrRec

Notre algorithme de construction de la structure d'indexation de KpyrRec suit les grandes lignes suivantes :

- l'algorithme de classification K-means [Mac67] est utilisé pour répartir les données de l'espace en K classes ;
- l'algorithme PyrRec est ensuite appliqué sur chaque classe obtenue. Cet algorithme retourne une structure arborescente décrite dans la section 3.3.1 ;
- enfin, des frontières entre les différentes classes sont déterminées, elles permettent la répartition des K arbres B+ correspondants dans un arbre Espace binaire. Cette partie est similaire à celle décrite dans la section 3.2.

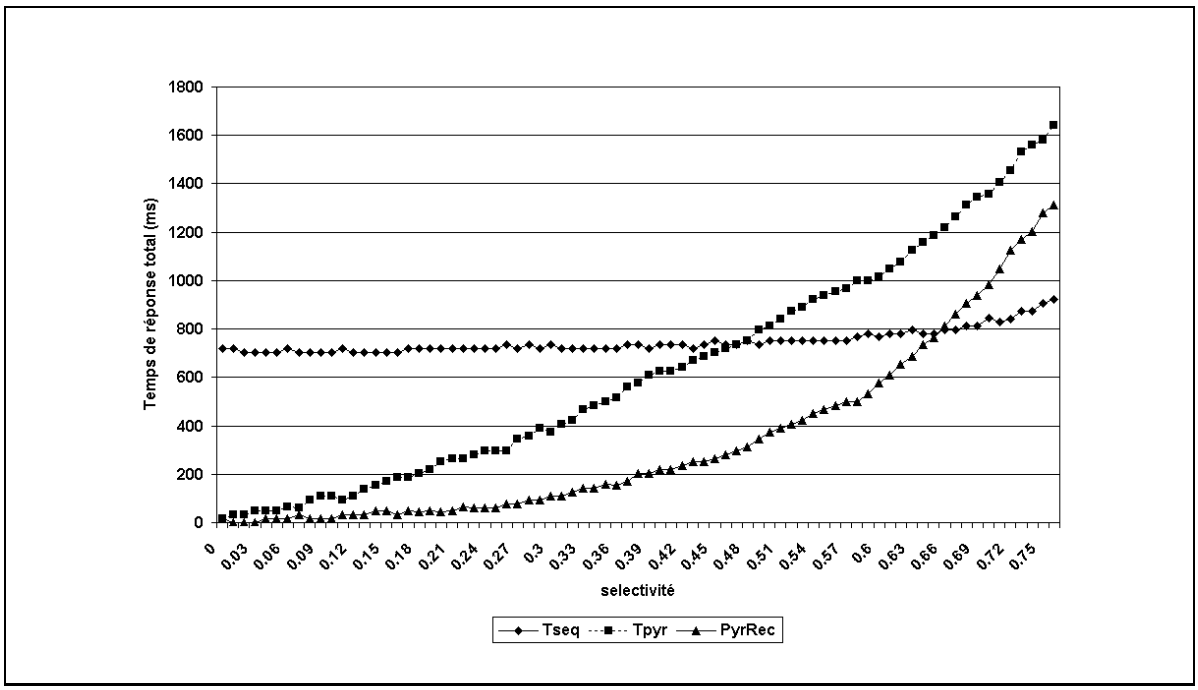

FIG. 3.14 – Effet de la sélectivité avec 1 000 000 données

Notre algorithme de recherche par fenêtrage utilisant la structure d'indexation KpyrRec procède comme suit :

- à partir d'un point requête *q*, une requête par fenêtrage est créée en fonction de la sélectivité choisie ;
- les classes (ou régions de l'espace) atteintes par cette requête sont récupérées à l'aide de l'arbre Espace ;
- pour chaque classe atteinte, l'algorithme place la requête dans son nouvel espace et détermine les tranches des pyramides qu'il faut scanner ;
- ensuite, un accès aux feuilles de l'arbre B+ permet de retourner les résultats possibles appartenant aux tranches accédées ;
- enfin, les résultats possibles sont analysés pour déterminer ceux qui répondent correctement à la requête *q*.

L'algorithme de recherche des plus proches voisins est similaire à celui présenté dans la section 3.2. Il est cependant nécessaire de mettre plus d'informations en mémoire selon le nombre de

récurrences effectuées. Nous remarquons que les opérations nécessaires lors d'une recherche dans notre structure arborescente créée par PyrRec ne sont pas plus compliquées que pour Kpyr ; le phénomène de récurrence n'a pas d'influence sur l'algorithme de recherche. Les expérimentations réalisées corroborent cette remarque et montrent les performances de KpyrRec par rapport à Kpyr entre autres.

3.3.4 Expérimentations

L'environnement utilisé pour réaliser les tests proposés est le même que celui décrit dans la section 3.2.2. Nous avons réalisé quelques expériences sur des données hétérogènes qui nous permettent de montrer que notre nouvelle structure d'indexation KpyrRec obtient de bonnes performances en temps de réponse. La figure 3.15 nous montre que pour un espace à 50 dimensions et 1 000 000 de données, KpyrRec obtient un meilleur temps de réponse, quel que soit la sélectivité de la requête par fenêtrage, comparé aux autres structures d'indexation : recherche séquentielle, Pyramide, P+Tree et Kpyr. Nous retrouvons un gain de temps d'environ 20% de KpyrRec par rapport à Kpyr. Notons que ce gain de temps est moins important que dans la figure précédente, ce phénomène s'explique par l'existence de classes (ici 4), ce qui implique un nombre moins important de données par pyramide.

Les courbes de la figure 3.16 sont moins attrayantes mais montrent des résultats qui restent tout de même intéressants. Elle mesure les temps de réponses des différentes structures d'indexation pour 1 000 000 données de dimension 50. Les courbes de KpyrRec et Kpyr sont similaires à la figure précédente avec un gain de temps pour KpyrRec quel que soit la sélectivité. Nous remarquons que pour une certaine valeur de sélectivité, les courbes de Kpyr et KpyrRec présentent une forte augmentation des valeurs du temps de réponse. Elles dépassent nettement celles de la courbe de P+Tree qui subissent la même augmentation pour une sélectivité plus élevée. Ces résultats montrent que pour des requêtes à large sélectivité, retournant comme résultats plus de 15% de l'ensemble des données, le temps de réponse de nos structures (Kpyr et KpyrRec) peut devenir aléatoire par rapport à P+Tree. Néanmoins, de telles valeurs de sélectivité ne sont pas significatives, elles correspondent à des requêtes trop générales (imprécises).

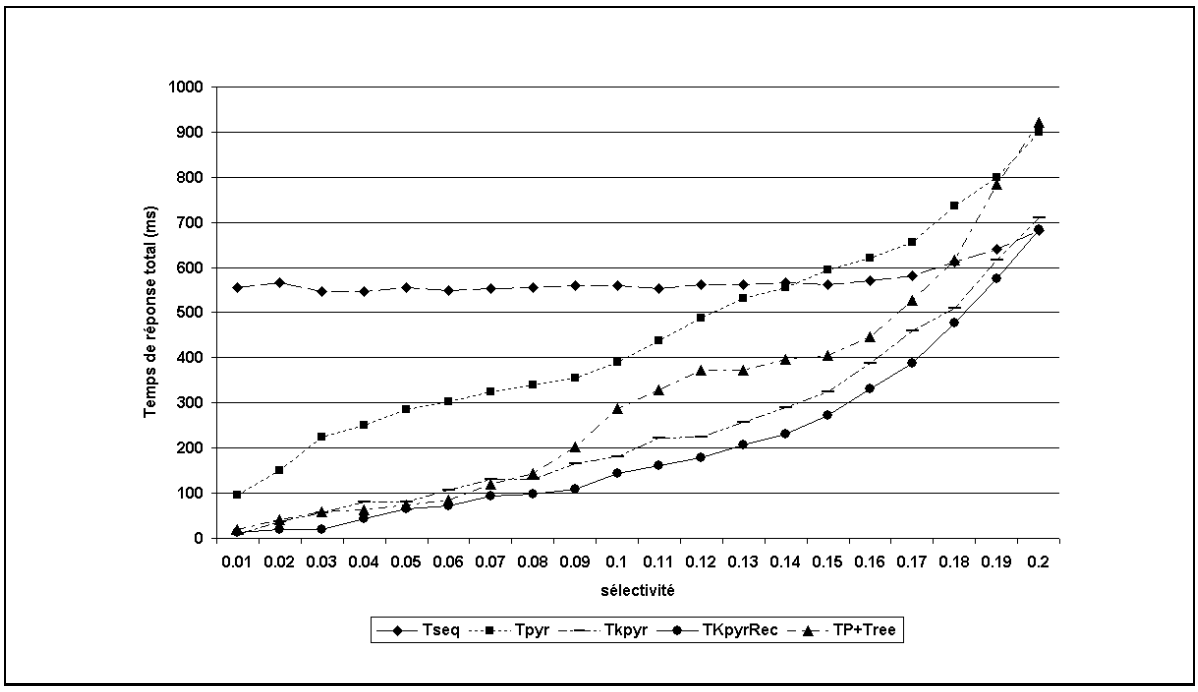

FIG. 3.15 – Effet de la sélectivité avec 1 000 000 données

Le même gain en temps de réponse a été constaté pour différents paramètres de la base de données : nombre de classes, volume de données, dimension de l'espace, etc. Grâce à l'algorithme PyrRec nous avons amélioré le temps de réponse de Kpyr car nous accédons à un nombre de données très réduit. De plus, notre méthode propose des index récursifs de petite taille qui permet d'avoir un gain mémoire non négligeable. Notons que les performances observées sur une requête par fenêtrage ont aussi été observées dans les résultats des expérimentations de l'algorithme des plus proches voisins.

3.4 Conclusion

Ce chapitre propose un aperçu général de nos structures d'indexation multidimensionnelle, Kpyr et KpyrRec. Ces deux structures d'indexation sont optimisées pour des bases de données volumineuses et de grandes dimensions. Kpyr se base sur l'optimisation de l'utilisation de la

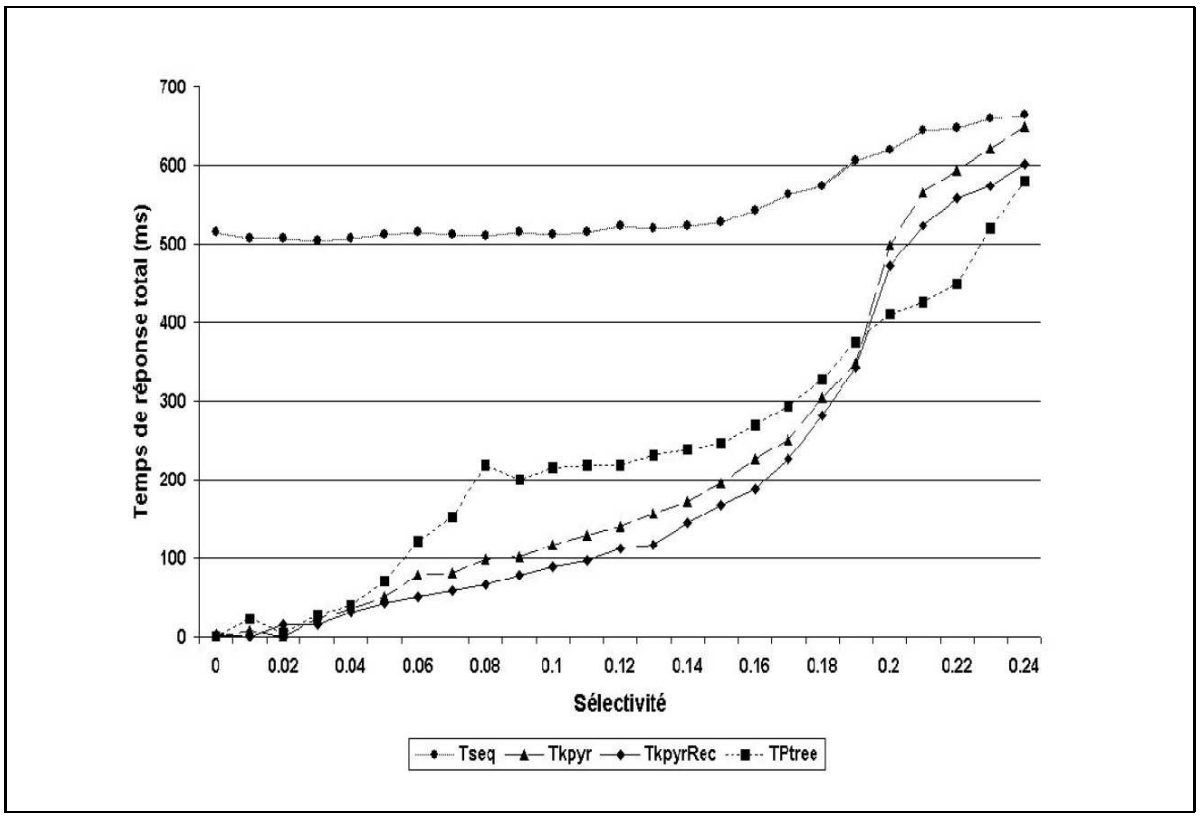

FIG. 3.16 – Problème remarqué pour une sélectivité élevée

technique de la pyramide en combinant avec la technique de classification k-means. La classification de la base de données permet d'avoir une meilleure répartition des points dans l'espace de chaque classe pour une application correcte de la technique de la pyramide. Nous avons proposé quelques unes des expérimentations réalisées pour Kpyr qui démontrent les performances de cette dernière en la comparant à des méthodes récentes de l'état de l'art ainsi qu'à la recherche séquentielle. Cette association de méthodes limite aussi le nombre d'accès inutiles aux données, néanmoins, nous avons constaté que ce point pouvait être amélioré pour accélérer la recherche. Pour atteindre ce but, nous avons proposé PyrRec, une structure d'indexation basée sur un découpage récursif de l'espace selon la répartition des données. L'association de PyrRec et k-means a permis de proposer KpyrRec qui présente des performances supérieures à celle de Kpyr. Le chapitre 4 présente une amélioration importante de ces structures d'indexa-

tion multidimensionnelle. Le chapitre 6 montre l'intégration de l'ensemble de notre approche dans un moteur de recherche en ligne.

Bibliographie

[BBK98] S. Berchtold, C. Bohm, and H. P. Kriegel. The pyramid technique : Towards breaking the curse of dimensionality. In *Proceedings of ACM SIGMOD Int. Conf on Management of Data*, pages 142–153, Seatle,Washington,USA, 1998.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, 1967. University of California Press.

[Mpe] Mpeg7. http ://www.chiariglione.org/mpeg/.

[Sal89] Gerard Salton. *Automatic Text Processing : The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[SM84] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, 1984.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11) :613–620, 1975.

[vR79] C.J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

[W3C] W3C. http ://www.w3.org/.

Chapitre 4

Optimisation de l'indexation avec une classification par projections aléatoires

Sommaire

Ce chapitre détaille dans une première partie les inconvénients majeurs de l'utilisation de l'algorithme k-means et présente quelques notions de classification et de projections aléatoires. Les inconvénients constatés théoriquement et expérimentalement de k-means ont orienté nos recherches vers l'élaboration d'un algorithme de classification adapté à notre base de données. Nous proposons un algorithme de classification par projections aléatoires qui présente de nombreux avantages : possède une complexité en $O(N \log N)$, est adapté à des données de grande dimension et n'est pas affecté par des données "bruit". Nous détaillons dans une deuxième partie cet algorithme accompagné des résultats expérimentaux et d'une application sur les images. Enfin, nous présentons RPyR, une structure d'indexation multidimensionnelle optimisée par la combinaison de notre classification par projections aléatoires avec notre structure d'indexation PyrRec.

4.1 Vers un nouvel algorithme de classification

4.1.1 Inconvénients d'une classification mal adaptée

L'un des résultats importants que nous avons constaté au cours de nos expérimentations du chapitre précédent était le lien existant entre le temps de réponse et le nombre de données accédées pour une recherche. Nous avons donc introduit une nouvelle structure d'indexation multidimensionnelle *PyrRec*. Comme nous l'avons montré dans la section 3.3.4, une amélioration du temps de réponse a été constatée pour différents paramètres de la base de données : le nombre de classes, le volume des données, la dimension de l'espace, etc. En effet, l'utilisation de PyrRec réduit le temps de réponse d'une recherche car celle ci accède à un nombre de données très réduit par rapport à une recherche utilisant simplement la technique de la Pyramide. Néanmoins quelques résultats aléatoires ont été observés pour la même base de données. Ces phénomènes aléatoires ne peuvent pas être dûs à la structure d'indexation PyrRec qui est exacte. C'est pourquoi nous avons focalisé nos recherches sur la classification pour améliorer les accès à notre base de données.

Les structures d'indexation que nous avons proposées utilisent un algorithme de classification des données avant d'appliquer une structure d'indexation basée sur un découpage géomé-

trique de l'espace. Nous avons choisi k-means [Mac67], un algorithme de partitionnement des données, pour sa rapidité et son efficacité pour un volume important de données. Comme nous n'avons émis aucune hypothèse sur la nature de la distribution hétérogène des données en plusieurs classes, la classification réalisée par k-means semblait affecter parfois la performance des résultats lors d'une recherche.

En effet, l'algorithme k-means fonctionne très bien pour une répartition des données en classes de forme approximativement sphérique (ou globulaire). Si la répartition des données devient plus complexe, ou si la base de données contient des données "bruit", k-means donnera des résultats très aléatoires, voire incorrects. Nous définissons les données "bruit" comme des données réparties aléatoirement dans l'espace sans appartenance à une classe déjà existante. De plus, k-means présente le problème du choix de K, paramètre représentant le nombre de classes à fixer par l'utilisateur. Ce choix a une grande importance pour l'efficacité de la classification de k-means et peut poser problème selon la base de données utilisée, même pour un utilisateur expérimenté. Enfin, la haute dimensionalité des données n'est pas non plus un avantage pour k-means.

Pour une structure d'indexation multidimensionnelle, la classification préalable des données n'affecte pas la qualité des résultats de la recherche, mais elle peut, en revanche, influer sur le temps de réponse. En effet, quel que soit la répartition et la nature des données, l'algorithme de classification k-means fournit un ensemble de K classes à l'utilisateur. Ensuite, l'ensemble de ces classes, contenant l'intégralité des données, est indexé avec l'une de nos méthodes dans des arbres B+. Les erreurs de classification de k-means peuvent être dues, entre autres, à des mauvais choix de paramètres initiaux, une mauvaise initialisation, à la présence de données "bruit" ou encore à une base de données répartie en classes de formes non adaptée pour k-means. Ces erreurs de classification peuvent provoquer des zones importantes de recouvrement. Deux exemples de recouvrement entre classes sont présentés sur la figure 4.1 : à droite, un recouvrement dû au choix d'un K inadapté, et à gauche, dû à la présence de données "bruit".

Si une recherche q, représentée sur la figure par le carré noir, se situe dans une région de l'espace où deux classes se recouvrent, il est nécessaire de traiter ces classes (chargement en mémoire de l'arbre B+ correspondant, recherche dans l'arbre, etc.), ce qui implique une aug-

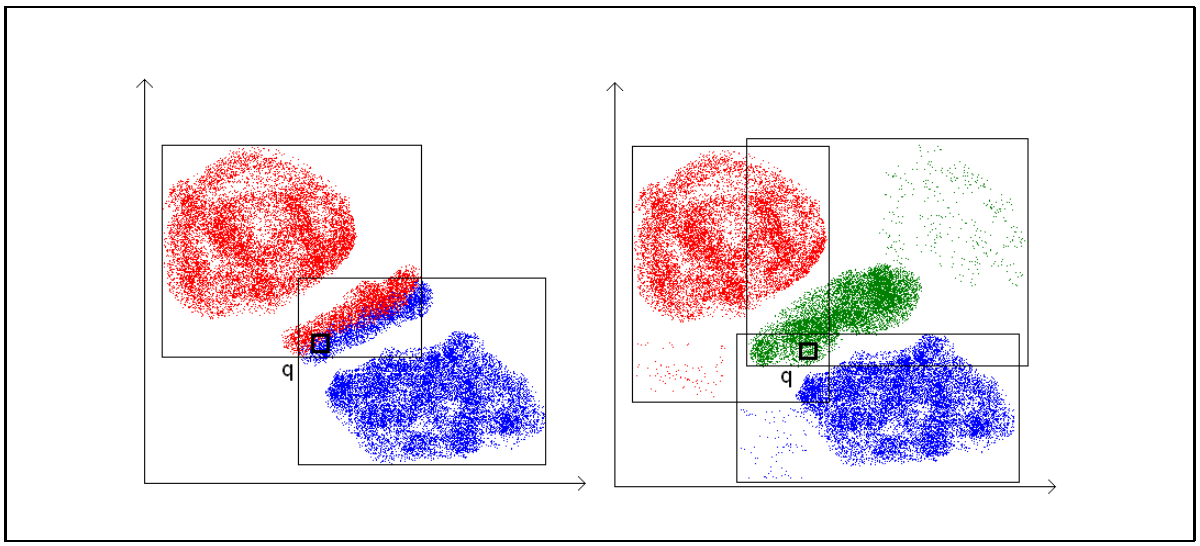

FIG. 4.1 – Exemples de recouvrement entre classes

mentation significative du temps de réponse. Or, la plupart de ces accès aux données s'avèrent inutiles.

Pour ces différentes raisons, nous proposons *RPyR*, que nous détaillons dans la section 4.3, une nouvelle structure d'indexation multidimensionnelle alliant une méthode de classification des données par projections aléatoires à la technique PyrRec. Notre classification est plus robuste à différentes répartitions des données de grande dimension et gère dans le même temps la possibilité d'avoir des données "bruit" ;

Dans la section suivante, nous proposons un aperçu de l'état des méthodes de classification, avant de détailler notre méthode de classification par projections aléatoires accompagnée de résultats expérimentaux.

4.1.2 Classification et Projection aléatoires

Soit S un sous ensemble fini de $\mathbb{R}^n$ et δ, γ deux nombres réels tels que $\delta > 0$ et $\gamma \geq 0$. Si C est un sous-ensemble de Borel de $\mathbb{R}^n$, alors on note $\text{vol}(C)$ la valeur de sa mesure de Lebesgue que nous considérons comme son volume.

Une *classification* $-(\delta, \gamma)$ *de* S est une famille $\kappa = \{C_1, \ldots, C_p\}$ de sous ensembles non

vides de $\mathbb{R}^n$ (Ce sont les éléments constituants la classification) qui satisfont les conditions suivantes :

1. Les ensembles éléments de κ sont deux à deux disjoints ;
2. Quelque soit i, $1 \leq i \leq p$, la densité des points de S dans chaque ensemble C_i est supérieure à δ, nous avons donc :
$$\frac{|S \cap C_i|}{\mathsf{vol}(C_i)} \geq \delta;$$
3. La proportion des points situés à l'extérieur des ensembles C_i est inférieure à γ, soit,
$$\frac{|\mathsf{UNC}(\kappa)|}{|S|} \leq \gamma,$$
où $\mathsf{UNC}(\kappa) = S - \bigcup_{i=1}^{p} C_i$ est l'ensemble *des points non classés de S*.

Les *classes* de la classification κ sont les ensembles $S \cap C_i$ pour $1 \leq i \leq p$.

Une *classification de S* est une famille $\kappa = \{C_1, \ldots, C_p\}$ qui est une classification-(δ, γ) de S avec $\delta, \gamma \in \mathbb{R}$.

La seconde condition de la définition ci dessous assure que la densité des points de chaque ensemble C_i est suffisamment importante ; la troisième limite le pourcentage des points non classés.

Soit $\mathbf{H}$ une matrice-$n \times n$ orthogonale aléatoire. Une telle matrice peut être obtenue par exemple, en choisissant aléatoirement les valeurs d'une matrice-$n \times n$ en utilisant une distribution uniforme sur un intervalle et ensuite, en appliquant la technique de Gram-Schmidt pour produire une matrice orthogonale.

Une projection aléatoire de $\mathbb{R}^n$ est une transformation linéaire $\mathbf{\Phi_H} : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ définie par $\mathbf{\Phi_H}(\mathbf{x}) = \mathbf{Hx}$ pour $\mathbf{x} \in \mathbb{R}^n$. L'ensemble des lignes $\mathbf{u}_1, \ldots, \mathbf{u}_n$ de $\mathbf{H}$ représente notre *repère aléatoire n-dimensionnel*.

Identifier des classes dans un espace uni-dimensionnel est un processus de complexité approximativement linéaire utilisant un algorithme décrit dans la section 4.2.1) qui construit l'histogramme de densité de l'ensemble des points projeté sur une droite aléatoire.

Si la projection d'un sous ensemble K de $\mathbb{R}^n$ dans un espace de dimension inférieure est une classe alors il est impossible de conclure que K est une classe dans l'espace n-dimensionnel.

Un autre problème est posé par le fait que deux classes disjointes dans l'espace *n*-dimensionnel peuvent avoir des projections dans un espace de dimension inférieure qui ont une intersection non vide, ce que nous définissons comme le phénomène d'*occultation*.

Soient C, D deux classes dans $\mathbb{R}^n$ et $\mathbf{u}$ un vecteur unitaire du même espace. Par souci de clarté dans notre présentation, les classes C et D sont approximées par des sphères de rayon r_1 et r_2 centrées sur les points $\mathbf{c}$ et $\mathbf{d}$ respectivement. La projection orthogonale d'un ensemble K sur un vecteur $\mathbf{u}$ est l'ensemble :

$$\mathsf{proj}_u(K) = \{\mathbf{u} \cdot \mathbf{x} | \mathbf{x} \in K\}.$$

Une *occultation-**u*** entre les deux classes C et D apparaît si

$$\mathsf{proj}_u(C) \cap \mathsf{proj}_u(D) \neq \emptyset.$$

Cette situation est représentée dans la figure 4.2.

Ce phénomène d'occultation est un inconvénient majeur de notre point de vue puisqu'il fusionne la projection de C et de D sur le vecteur $\mathbf{u}$.

Nous devons alors évaluer la probabilité qu'une occultation-$\mathbf{u}$ puisse avoir lieu entre deux classes car notre algorithme se base sur une combinaison de classifications sur des projections unidimensionnelles pour identifier les classes finales dans $\mathbb{R}^n$. Soit $\mathbf{u}$, un vecteur unitaire aléatoire, l'angle α entre $\mathbf{u}$ et le vecteur $\mathbf{d} - \mathbf{c}$ est uniformément distribué dans l'intervalle $[0, 2\pi]$. Les arguments sont essentiellement les mêmes pour des sous espaces de dimension arbitraire. Une occultation-$\mathbf{u}$ entre les deux classes C et D a lieu lorsque la longueur de la projection du segment joignant les points $\mathbf{c}$ et $\mathbf{d}$ est inférieure à $r_1 + r_2$; En d'autres termes, si $|\mathbf{u} \cdot (\mathbf{d} - \mathbf{c})| = \| \mathbf{d} - \mathbf{c} \| \, |\cos\alpha| \leq r_1 + r_2$.

Par conséquence, la probabilité d'avoir une occultation-$\mathbf{u}$ entre les deux classes est :

$$P\left(-\frac{r_1 + r_2}{\| \mathbf{d} - \mathbf{c} \|} \leq \cos\alpha \leq \frac{r_1 + r_2}{\| \mathbf{d} - \mathbf{c} \|} \right),$$

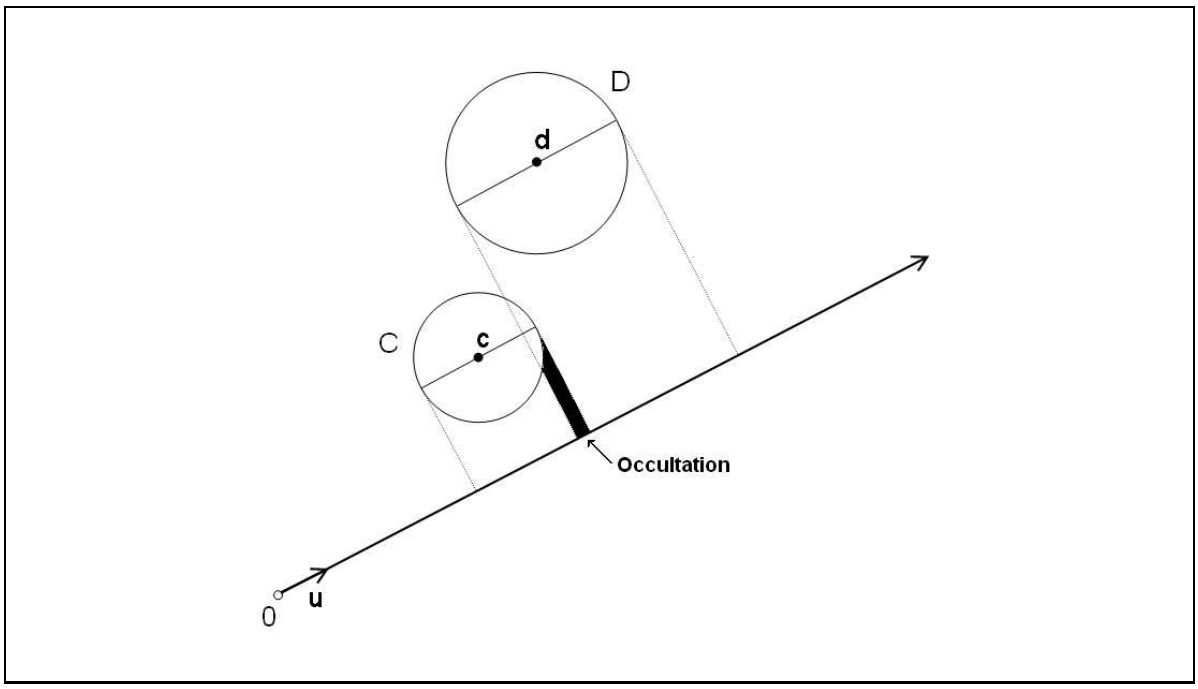

FIG. 4.2 – Phénomène d'*occultation* entre deux classes sur une projection aléatoire

ce qui est facilement égale à :

$$1-\frac{2}{\pi}\arccos\left(\frac{r_1+r_2}{\|\,\mathbf{d}-\mathbf{c}\,\|}\right),$$

avec $\|\,\mathbf{d}-\mathbf{c}\,\|\geq r_1+r_2$, ce qui est le cas lorsque les classes sont suffisamment proches l'une de l'autre. En utilisant le développement limité des séries de MacLaurin de $\arccos z$,

$$\arccos z=\frac{\pi}{2}-\left(z+\frac{z^3}{6}+\frac{3}{40}\frac{z^5}{5}+\cdots\right),$$

nous pouvons arrondir cette valeur par $z=\frac{2(r_1+r_2)}{\pi\|\mathbf{d}-\mathbf{c}\|}$.

Soit O_i le phénomène d'occultation observé sur l'axe i de notre repère aléatoire, pour $1\leq i\leq n$, nous devons évaluer la probabilité qu'il y ait au moins une projection sur laquelle il n'y aura pas de phénomène d'occultation entre les deux classes. Nous avons donc $P(\overline{O_1}\cup$

$\cdots \cup \overline{O_n}) = 1 - P(O_1 \cap \cdots \cap O_n)$. En prenant l'hypothèse que les phénomènes d'occultation $O_1, \ldots, O_n$ sont indépendants, nous pouvons en déduire que

$$P(\overline{O_1} \cup \cdots \cup \overline{O_n}) = 1 - \left(\frac{2(r_1 + r_2)}{\pi \parallel \mathbf{d} - \mathbf{c} \parallel} \right)^n .$$

Il est évident que l'hypothèse d'indépendance n'est pas réelle. Nous l'avons formulée ici pour obtenir une estimation possible, soutenue par nos résultats expérimentaux.

Nous pouvons aussi déduire de la formule de probabilité ci dessus que la probabilité qu'il existe une dimension qui n'ait pas ce phénomène d'occultation entre deux classes augmente avec le nombre de dimensions de notre espace. Cette observation montre l'utilité du choix d'un repère aléatoire séparant, au moins partiellement, avec une certaine incertitude, deux classes qui n'auraient pas été séparées dans leur repère original.

4.2 Algorithme de Classification par projections aléatoires

4.2.1 Notre Algorithme de Classification par Projections Aléatoires

Nous proposons ici un algorithme de classification qui combine des idées provenant des projections aléatoires et de la classification par densité. D'un certain point de vue, notre méthode se rapproche de celle proposée par [AGGR98]. Nous construisons des classes dans des espaces de plus petites dimensions et nous sélectionnons les dimensions qui peuvent identifier au mieux les classes dans notre base de données originale. Notre contribution principale consiste à choisir un repère orthonormal aléatoire dans $\mathbb{R}^n$. Nous projetons ensuite l'ensemble des points de notre base de données sur chaque axe de ce repère. Nous avons montré dans la section précédente que choisir un repère aléatoire à la place du repère original diminue la probabilité d'avoir un phénomène d'*occultation* entre les classes.

Les histogrammes de densité des projections unidimensionnelles, qui contiennent le plus d'informations sur les classes, sont ensuite combinés pour retrouver la localisation des classes dans l'espace de données original. Une deuxième partie de notre algorithme affine la classification trouvée par projection aléatoire, à l'aide de deux processus distincts : la *bi-modulation*

et *l'expansion des classes*.

A partir d'une base de données de dimension n, la phase de projections aléatoires suit cinq étapes :

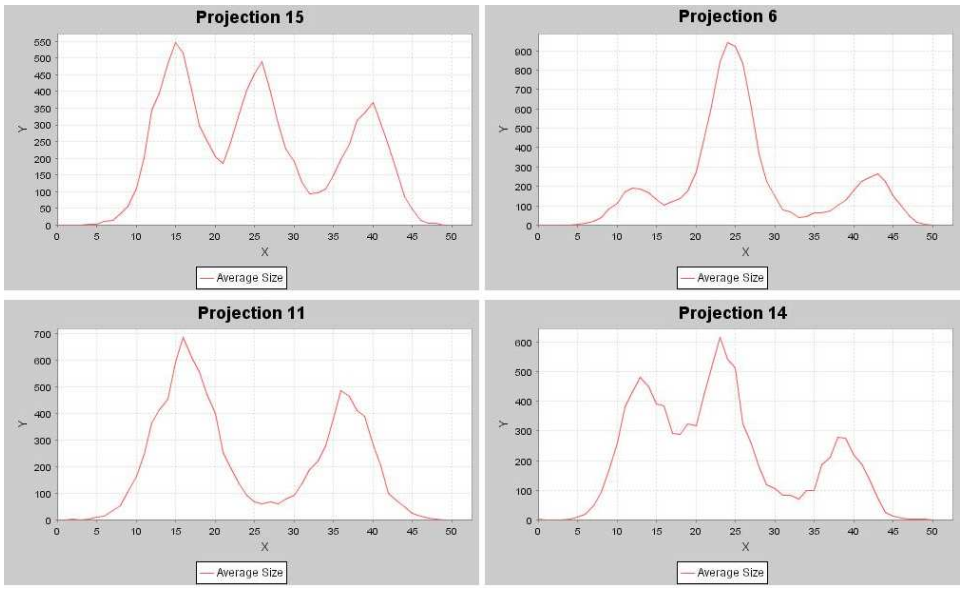

FIG. 4.3 – Exemples d'histogrammes de densité sur certaines projections

1. l'algorithme choisit une nouvelle base orthonormale aléatoire $n * n$. Pour cela, nous construisons une matrice $n * n$ avec des valeurs créées à partir d'une distribution uniforme sur un intervalle. Nous appliquons ensuite la technique de Gram-Schmidt sur notre matrice aléatoire pour obtenir une matrice orthonormale ;
2. chaque point est ensuite projeté sur chaque dimension (axe du repère) de la nouvelle base choisie, ce qui permet d'obtenir n histogrammes de densité, voir figure 4.3 un exemple de quatre histogrammes de densité. Chaque i ème histogramme contient un nombre k d'intervalles de largeur ℓ_i, le choix de k dépend de la taille de D. Par exemple, pour $|D| = 10^4$, nous avons utilisé $k = 50$;
3. les σ "meilleurs" histogrammes contenant le plus d'informations sur les classes sont

choisis parmi les n histogrammes. La qualité d'une projection est évaluée par le produit de la moyenne des hauteurs des pics et du nombre de pics d'un histogramme. Dans l'exemple de la figure 4.3, les projections 14 et 15 ont visiblement trois hauts pics, et sont ainsi jugées de meilleure qualité que les autres. Un histogramme de densité qui ne contient qu'un seul sommet n'apporte aucune information sur la répartition des classes dans l'espace. Nos différentes expériences ont permis d'évaluer et de fixer le choix du paramêtre σ à une valeur égale à 10 % de n, le nombre total d'histogrammes.

4. nous obtenons, à partir des "meilleurs" histogrammes sélectionnés, une liste d'intervalles dans lesquels la densité du nombre de points est élevée. Ces intervalles pourraient contenir une ou plusieurs classes. Dans l'exemple de la figure 4.4, les trois intervalles obtenus pour la projection 14 sont compris entre 9 et 16, 21 et 28 et enfin 35 et 43. Ces intervalles permettent de savoir que notre base de données est au moins répartie en 3 classes distinctes.

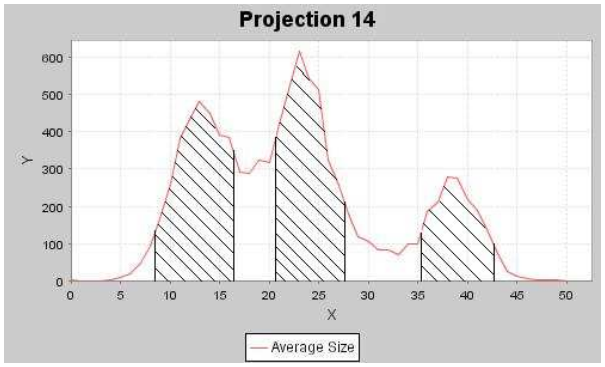

FIG. 4.4 – Exemple d'intervalles choisis pour la projection 14

5. nous combinons l'ensemble de ces intervalles pour former des régions de densité élevée que nous définissons ensuite comme les classes obtenues par notre phase de projections aléatoires. Cet algorithme de combinaison est détaillé plus loin dans la section. La figure 4.5 montre la combinaison de la projection 14 et de la projection 15. Cette combinaison nous permet de connaître 9 régions (en gris foncé sur la figure) qui pourraient contenir une densité élevée de données, il suffit ensuite de calculer pour chaque région

sa densité et de déterminer si cette région correspond à une classe ou non.

A la fin de cette phase de projections aléatoires, notre algorithme nous procure un ensemble de K classes déterminées par des régions dans l'espace où la densité de points est importante. A ce moment précis, le nombre K de classes est fixé par notre algorithme automatiquement.

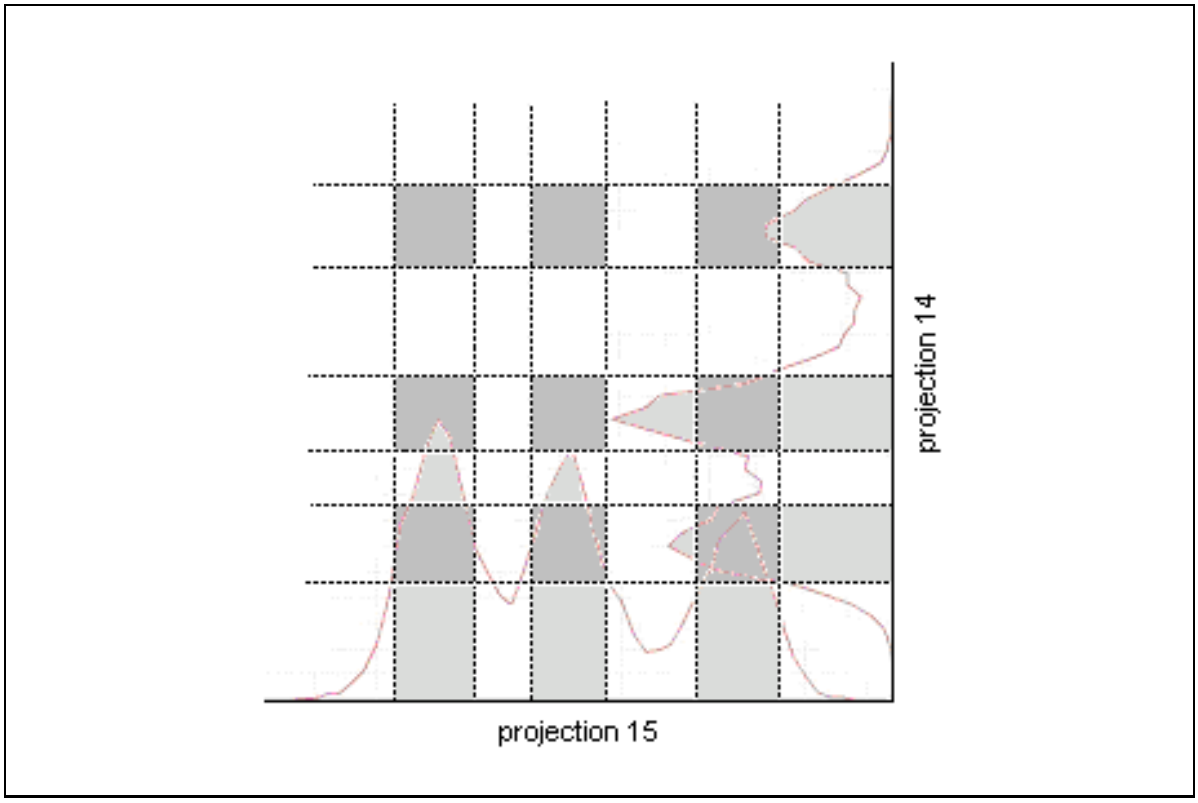

FIG. 4.5 – Exemple de combinaison d'intervalles pour deux projections

Une partie importante de notre algorithme de projections aléatoires est la combinaison des σ "meilleurs" histogrammes. Supposons que les pics de la i ème projection aléatoire correspondent aux intervalles $C^i_1,\ldots,C^i_{p_i}$. Soit σ, le nombre de meilleurs histogrammes d'une projection, nous utilisons alors une structure de données $\mathscr{F}$, où chaque donnée contient $1+n+t$ caractéristiques. Tous les points $\mathbf{x}=(x_1,\ldots,x_n)$ à classer sont représentés par un identifiant unique du point, les coordonnées originales du point $x_1,\ldots,x_n$ et pour chaque projection i, un

nombre $B(\mathbf{x},i)$ défini par :

$$B(\mathbf{x},i) = \begin{cases} j & \text{si la } i^{\text{ème}} \text{ projection de} \\ & \mathbf{x} \text{ appartient à } C_j^i, \\ 0 & \text{sinon.} \end{cases}$$

$\mathscr{F}$

Id du point	x_1	$\cdots$	x_n	$B(1,\mathbf{x})$	$\cdots$	$B(t,\mathbf{x})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
h	a_1	$\cdots$	a_n	b_1	$\cdots$	b_t
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Les données contenant au moins un 0 sont retirées de notre structure. A ce stade de l'algorithme, ces données correspondent à des points non classés. Ensuite, la structure de données est triée suivant les champs de valeurs $B(1,\mathbf{x}),\cdots,B(t,\mathbf{x})$. Chaque ensemble de points correspondant à un vecteur $(b_1,\ldots,b_t)$ est un ensemble

$$C_{b_1\cdots b_t}^{i_1\cdots i_t} = C_{b_1}^{i_1} \times \cdots \times C_{b_t}^{i_t}$$

qui peut être considéré comme un élément possible de notre classification. La condition

$$\frac{|\text{proj}_{i_1\cdots i_t}(S) \cap C_{b_1\cdots b_t}^{i_1\cdots i_t}|}{m(C_{b_1\cdots b_t}^{i_1\cdots i_t})} \geq \delta$$

assure que les classes

$$\text{proj}_{i_1\cdots i_t}(S) \cap C_{b_1\cdots b_t}^{i_1\cdots i_t},$$

où $\text{proj}_{i_1\cdots i_t}(S)$ est la projection de S sur les dimensions $i_1\cdots i_t$ du repère aléatoire, dépassent la valeur minimale de densité δ fixée par l'utilisateur. Dans nos expériences, nous avons utilisé $\delta = 0.01$, ce qui reflète notre décision de ne pas considérer les régions contenant moins de 1% de points comme des classes, la densité de la région étant trop faible pour être intéressante. La

valeur de δ doit être choisie par l'utilisateur et peut être différente selon les applications.

Notons que

$$|\mathrm{proj}_{i_1\cdots i_t}(S) \cap C_{b_1\cdots b_t}^{i_1\cdots i_t}| \leq \min_r |\mathrm{proj}_{i_r}(S) \cap C_{b_r}^{i_r}|.$$

C'est pourquoi la densité minimale imposée aux classes obtenues implique que la densité unidimensionnelle δ_1 est au moins égale à $\delta\ell^{t-1}$ où ℓ est la largeur des intervalles des histogrammes définies précédemment.

Le coût asymptotique de notre algorithme de classification par projection aléatoire est en $O(N\log N)$, où N est le nombre de données de notre base. L'opération la plus onéreuse de notre algorithme est le tri de la structure de données $\mathscr{F}$. Cependant, il est important de noter que si le nombre de dimensions n est relativement élevé par rapport au nombre de données, nous aurons alors un algorithme en $O(n^2N + N\log N)$.

4.2.2 Algorithmes d'amélioration de la classification

Une deuxième phase de notre algorithme de classification consiste en deux post-processus qui permettent d'améliorer la classification obtenue par projections aléatoires. Cette deuxième partie n'affecte pas le coût asymptotique de notre algorithme. Nous avons constaté, lors de la première partie de notre algorithme, que notre base de données était répartie en K classes déterminées automatiquement. La combinaison des différents intervalles des projections aléatoires, figure 4.5, nous permet d'obtenir des zones de très forte densité. Les points à l'extérieur de ces régions de forte densité ne sont pas classés à cette étape de notre algorithme. Or, il est possible que certains d'entre eux forment une classe qui n'a pas été découverte par notre algorithme de projections aléatoires ou simplement appartiennent à une des classes trouvées mais se situent à la limite des intervalles fixés par notre premier algorithme. Ces deux possibilités nous ont poussés à développer deux algorithmes post-processus qui permettent d'améliorer la qualité de notre classification finale.

Le premier post processus appelé *bi-modulation* consiste en une fusion de deux classifications. Par la nature aléatoire de notre premier algorithme, les résultats obtenus par deux exécutions de celui-ci sont différents. Chaque exécution i fournit K_i classes et un ensemble assez important de points non classés. Une grande partie des deux résultats obtenus est simi-

laire, cependant des différences peuvent apparaître, notamment sur l'ensemble des points non classés. L'idée principale est de combiner les résultats de deux classifications pour en obtenir une meilleure et diminuer l'ensemble des points non classés. Dans de très rares cas, l'une des exécutions peut avoir manqué une classe, occultée par les autres sur l'ensemble de "meilleures" projections, l'ensemble des points de cette classe "manquée" fait alors partie des points non classés pour cette exécution. L'algorithme de *bi-modultation* va atténuer ce phénomène et re-classer ces points dans une nouvelle classe grâce au résultat de la deuxième exécution. Suivant cette même idée d'amélioration des classes, il est possible de classer une partie des points non classés par une exécution grâce à l'autre exécution. Les résultats expérimentaux obtenus pour cette phase de *bi-modultation* nous montre que souvent, nous avons une augmentation de la taille des classes, et très rarement une classe "manquée" retrouvée. A la suite de cette phase de bi-modulation, nous obtenons K' classes, avec $K' \geq K$ et un ensemble de points non classés plus petit.

Le deuxième post-processus est une ε-*expansion* des régions des classes obtenues. Cet algorithme augmente le rectangle minimum englobant d'un pourcentage ε et détermine si l'ensemble des points non classés contenu dans cette nouvelle région de la classe appartiennent ou non à la classe. Si de nouveaux points sont ajoutés à une classe, son rectangle minimum englobant est recalculé et de nouveau étendu. Ce post-processus permet d'inclure les points non classés se trouvant à proximité d'une classe dans cette dernière. Les points restants non classés après ces deux post-processus sont considérés comme "bruit" par notre algorithme.

4.2.3 Expérimentations

Nous avons réalisé nos expérimentations sur trois types de données : des données purement synthétiques générées aléatoirement dans $\mathbb{R}^n$ avec une répartition en plusieurs classes, des images "synthétiques" constituées de rectangles colorés aléatoirement répartis, et enfin une application sur des images réelles. Nous présentons un ensemble de résultats avec en particulier une comparaison avec les algorithmes de classification k-means et DB-scan [EKSX96].

Classes identifiées	K_1	K_2	K_3	K_4
C_1	1520	0	0	0
C_2	0	0	2297	0
C_3	0	0	0	283
C_4	0	780	0	0
Données non classées	1231	1214	1677	998

TAB. 4.1 – Intersections entre les classes originales et celles trouvées par notre algorithme

4.2.3.1 Expérimentations sur des données synthétiques

Nous avons testé notre algorithme de classification par projections aléatoires sur une base de données contenant 10000 points dans $\mathbb{R}^{30}$ répartis en quatre classes distinctes : K_1, K_2, K_3, K_4. Notre première phase de classification par projections aléatoires classe correctement la majorité des données dans quatre classes C_1, C_2, C_3, C_4, cependant nous avons environ 50% de points non classés, comme nous le montrons dans le tableau 4.1.

Le tableau 4.1 représente l'intersection entre les classes originales $K_1, \ldots, K_4$ (correspondant aux colonnes dans le tableau) et les classes $C_1, \ldots, C_4$ trouvées par notre algorithme. La dernière ligne correspond à l'ensemble des points non classés par notre algorithme. Nous observons que cette première phase de l'algorithme nous donne automatiquement le bon nombre de classes et très souvent les régions les plus denses des classes d'origine. Ce pourcentage élevé (environ 50%) montre l'utilité des deux algorithmes post-processing que nous avons développés. Pour ce jeu de données "simple", les deux algorithmes de post-processing remplissent correctement leur rôle et permettent d'obtenir un résultat quasi parfait que nous présentons pas ici.

Nous préférons présenter les résultats obtenus avec la même base de données sur laquelle nous avons rajouté 10% de données "bruit". Nous obtenons ainsi un jeu de données contenant 11000 vecteurs dans $\mathbb{R}^{30}$. Les résultats obtenus sont présentés dans le tableau 4.2, qui montre que le bruit influe très peu sur la qualité de notre classification.

Classes identifiées	K_1	K_2	K_3	K_4	Noise
C_1	0	2885	3	0	8
C_2	0	2	803	0	3
C_3	0	0	0	1929	10
C_4	1056	0	0	0	14
Données non classées	328	1472	591	931	965

TAB. 4.2 – Intersections entre les classes originales et celles trouvées par notre algorithme avec introduction de données "bruit"

4.2.3.2 Expériences sur des images synthétiques

Une seconde série d'expériences utilisent des images que nous pourrions comparer à des peintures de Mondrian [Mon]. Ces images sont construites avec des "rectangles" de tailles, couleurs et répartition aléatoires. Les résultats présentés sont des moyennes réalisées sur plusieurs images contenant entre 10 et 40 rectangles de couleurs. Nous présentons un exemple d'une image utilisée figure 4.6.

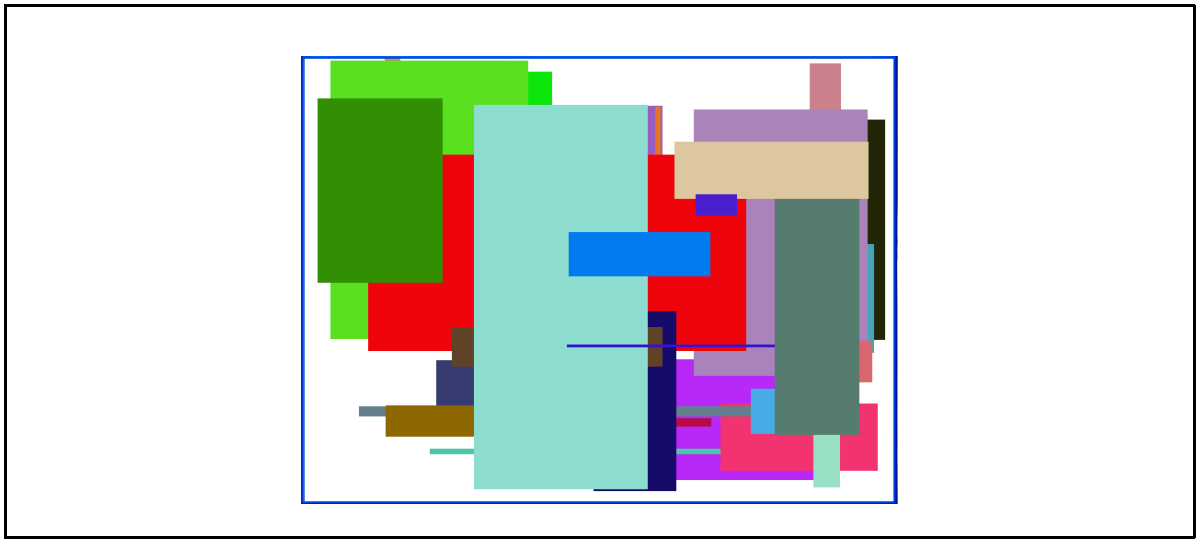

FIG. 4.6 – Exemple d'une image synthétique

Les objectifs principaux des séries d'expériences réalisées dans cette section sont de démontrer que notre algorithme de classification est de bonne qualité et ce, pour un nombre de

Vrai positif VP	Points colorés bien classés	63.29%
Faux positif FP	Points blancs mal classés	0.09%
Vrai négatif VN	Points blancs bien classés	28.9%
Faux négatif FN	Points colorés mal classés	7.79%

TAB. 4.3 – Définition et résultats de notre tableau de contingence

dimensions pouvant atteindre 320. Nous augmentons le nombre de dimensions en prenant des carrés de pixels. Par exemple, un carré de deux pixels de largeur nous donne 20 dimensions, les coordonnées x,y et les valeurs R, G et B de chacun de ces pixels. Nous pouvons noter que plus on augmente le nombre de dimensions, plus le nombre de données pour une même image diminue. Nous avons testé notre algorithme en se basant sur une image de $240 \times 320 = 76,800$ pixels représentant un ensemble de vecteurs dans $\mathbb{R}^5$. Nous avons ensuite groupé ces pixels dans des carrés de $4 \times 4 = 16$ pixels. Ce qui nous donne une base de données de 9600 vecteurs dans $\mathbb{R}^{80}$.

Dans une première partie de nos expériences, nous avons testé la capacité de notre algorithme à différencier les régions colorées, que nous traiterons par la suite comme nos classes, et les régions de pixels blancs. Nous définissons les termes de notre tableau de contingence dans 4.3.

La précision et le rappel pour notre évaluation sont donnés par les formules

$$\text{Précision} = \frac{\text{VP}}{\text{VP} + \text{FP}} = 0.99$$

et

$$\text{Rappel} = \frac{\text{VP}}{\text{VP} + \text{FN}} = 0.89.$$

La mesure F_1 est la moyenne harmonique de la précision et du rappel, sa valeur est 0.94. Ces résultats montrent les capacités de notre algorithme à retrouver des points classés.

Les évolutions de VP, VN, FP, et FN en fonction du paramètre d'extension ε du deuxième post-processus sont présentées dans les figures 4.7 à 4.10, respectivement.

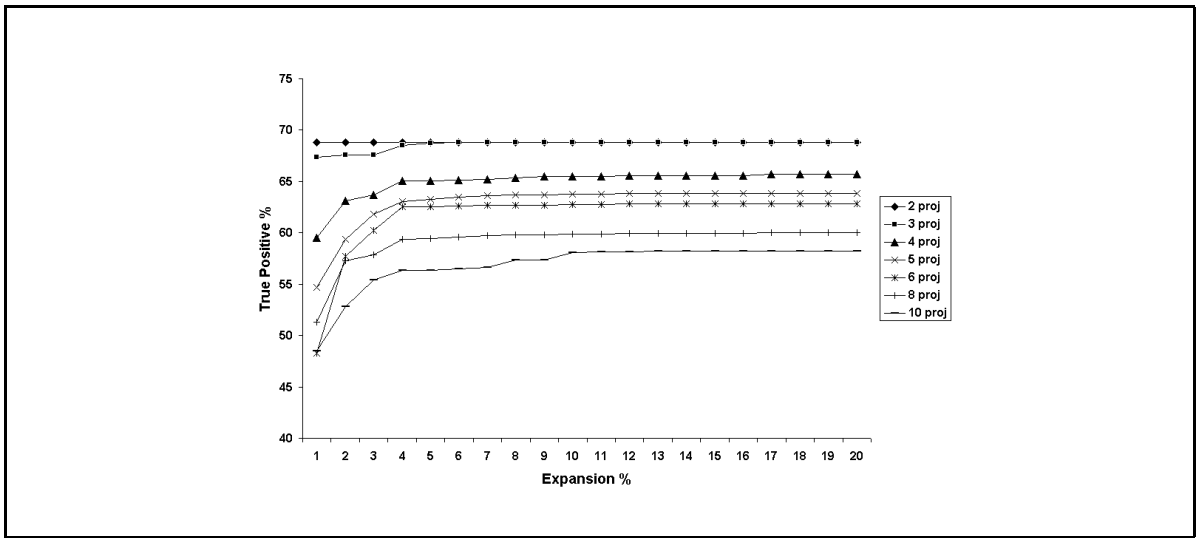

FIG. 4.7 – Évolution de VP en fonction d'ε

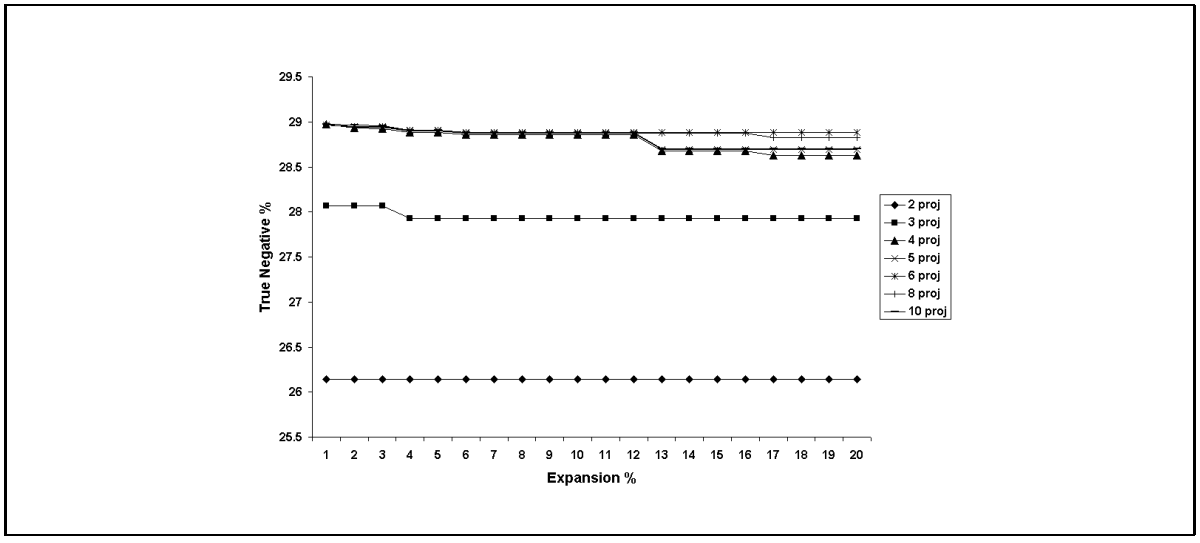

FIG. 4.8 – Évolution de VN en fonction d'ε

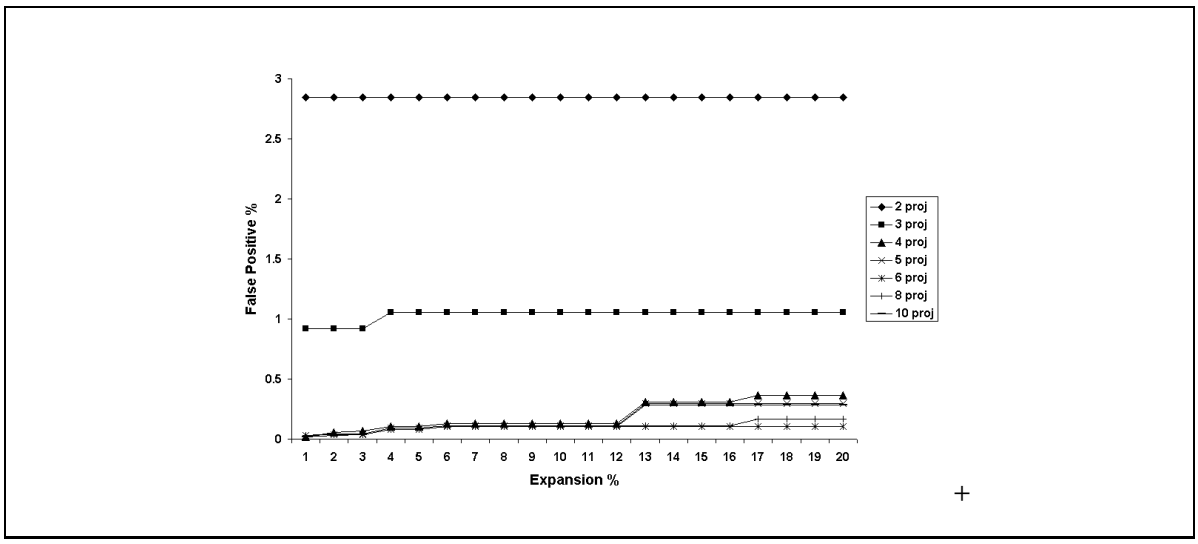

FIG. 4.9 – Évolution de FP en fonction d'ε

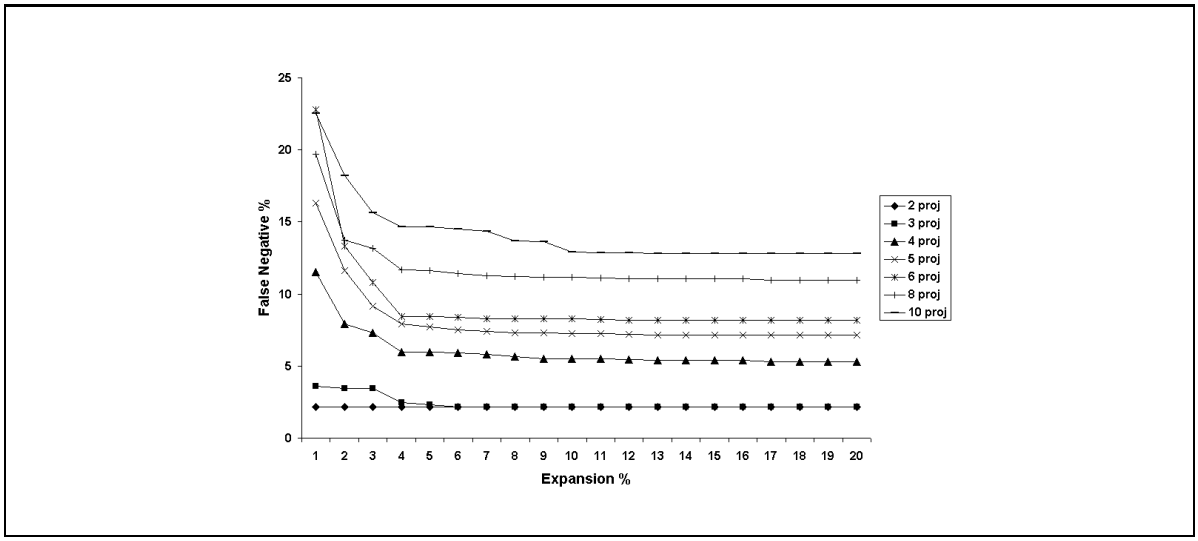

FIG. 4.10 – Évolution de FN en fonction d'ε

Les évolution de la précision, du rappel et de la mesure F_1 en fonction d'ε sont présentées dans les figures 4.11 à 4.13, respectivement.

Nous observons sur ces figures qu'il n'y a plus d'amélioration conséquente du rappel ou de la mesure F_1 lorsque la valeur d'ε dépasse 6%. Il en est de même lorsque le nombre de

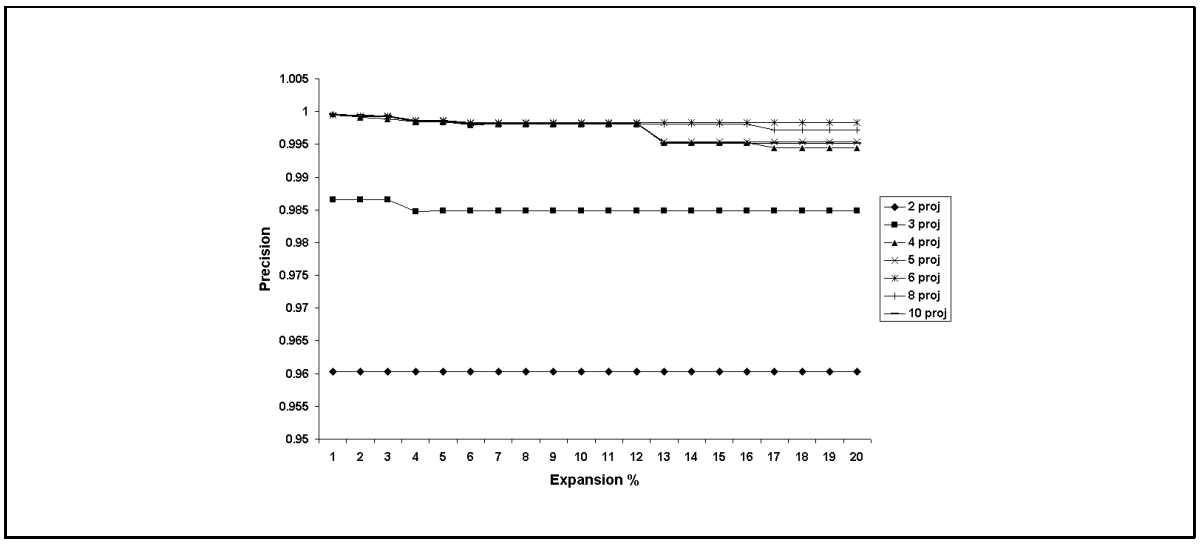

FIG. 4.11 – Évolution de la précision en fonction d'ε

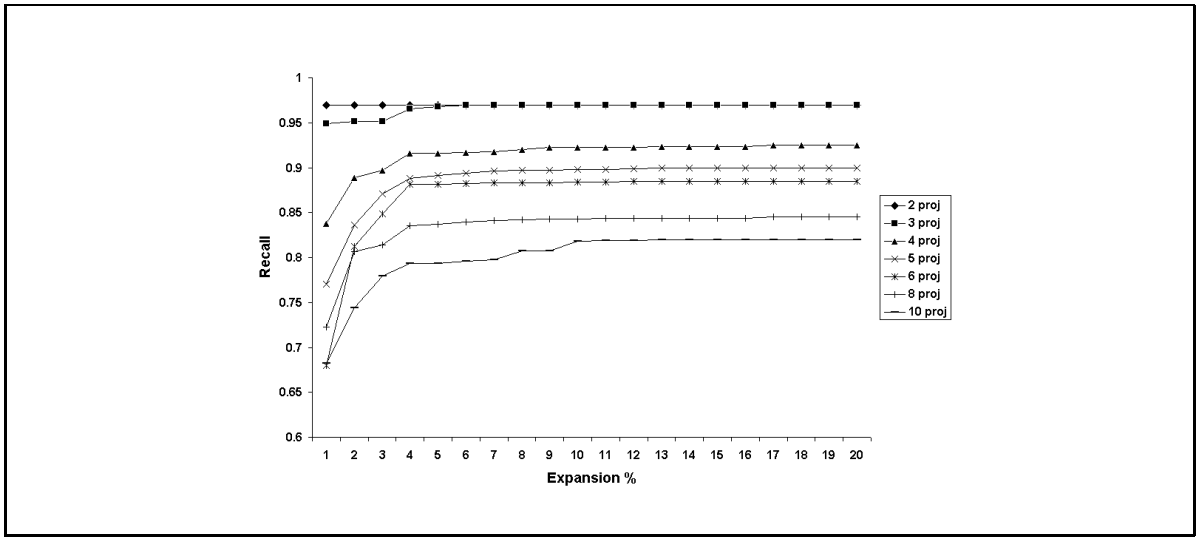

FIG. 4.12 – Évolution du rappel en fonction d'ε

projections dépasse 6. Nous utiliserons ces résultats dans la suite de nos expériences sur nos images réelles.

Pour évaluer la qualité de notre algorithme de classification à retrouver les classes origi-

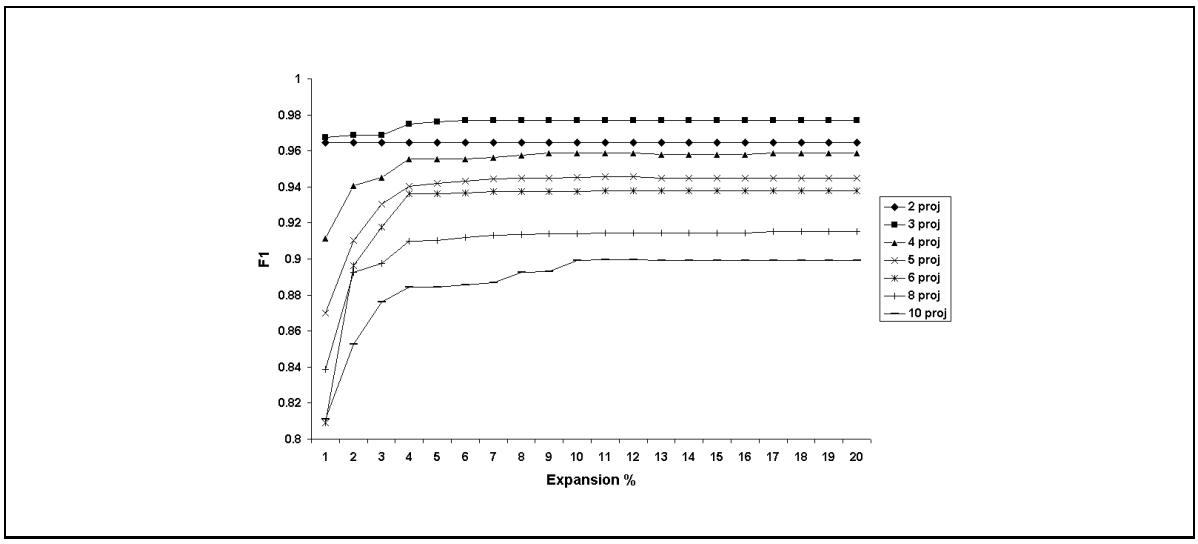

FIG. 4.13 – Évolution de la mesure F_1 en fonction d'ε

nales, nous avons calculé plusieurs mesures basées sur la validité de la classe (voir [TSK06], p. 549). Soit $C_1,\ldots,C_r$, nos r classes originales, et $D_1,\ldots,D_q$, les q classes trouvées par notre algorithme, la probabilité qu'une donnée d'une classe D_j, $1 \leq j \leq q$, appartienne à la classe C_k, $1 \leq k \leq r$ est égale à $p_{jk} = \frac{|D_j \cap C_k|}{|D_j|}$, ce qui est aussi connu comme la précision de D_j relative à C_k.

Nous définissons la *pureté d'une classe* D_j par le nombre $p_j = \max\{p_{jk} | 1 \leq k \leq r\}$, et la *pureté d'une classification* $\lambda = \{D_1,\ldots,D_q\}$ par

$$\mathsf{pur}(\lambda) = \sum_{j=1}^{r} \frac{|D_j|}{|D|} p_j,$$

où $D = \bigcup_{j=1}^{q} D_j$. La pureté d'une classification est une moyenne pondérée des puretés de chaque classe.

Nous présentons dans la figure 4.14 une moyenne du temps d'exécution de l'ensemble de notre algorithme en fonction du nombre de données. La complexité asymptotique de notre algorithme en $O(N \log N)$ est validée par nos expériences faites sur des données dans $\mathbb{R}^{20}$ et dans $\mathbb{R}^{80}$.

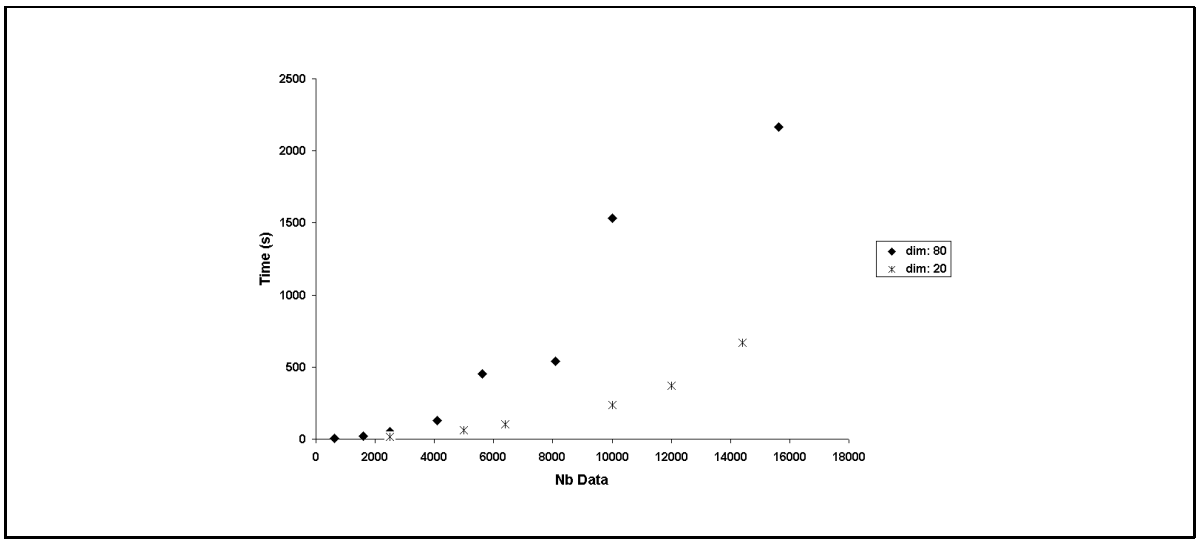

FIG. 4.14 – Temps (sec) en fonction du nombre de données

Nous présentons enfin quelques résultats obtenus sous forme d'un tableau de contingence (voir Tableau 4.4) pour notre algorithme de classification comparé aux algorithmes k-means et Db-scan [SEKX98], sur une base de données à répartition hétérogène de 10000 points de 80 dimensions et sur la même base de données contenant 10% de bruit. Le tableau de contingence est un tableau de mesures statistiques calculées à partir des classes obtenues par un algorithme de classification et des classes réelles.

	Algorithmes		
	k-means	Db-scan	Notre algorithme
VP (%)	62	58	63.3
FP (%)	28	3	0.1
VN (%)	10	34	28.9
FN (%)	1	5	7.8
Précision	0.69	0.95	0.99
Recall	0.98	0.92	0.89
F_1	0.81	0.94	0.94

TAB. 4.4 – Tableau de contingence entre notre algorithme, k-means et Db-scan

L'ensemble des expérimentations sur des images synthétiques montre les bonnes performances de notre méthode de classification par projections aléatoires par rapport à k-means et Db-scan. Rappelons que notre algorithme en $O(N \log N)$ a une complexité moins élevée que les deux autres algorithmes. Nous présentons dans la section suivante une application particulière de notre algorithme.

4.2.3.3 Application avec des images réelles

De multiples applications pourraient servir à tester notre algorithme de classification par projections aléatoires. Nous avons choisi de présenter ici une application sur des images réelles, car la visualisation des résultats est plus simple. Nous montrons un exemple d'image réelle figure 4.15 à gauche.

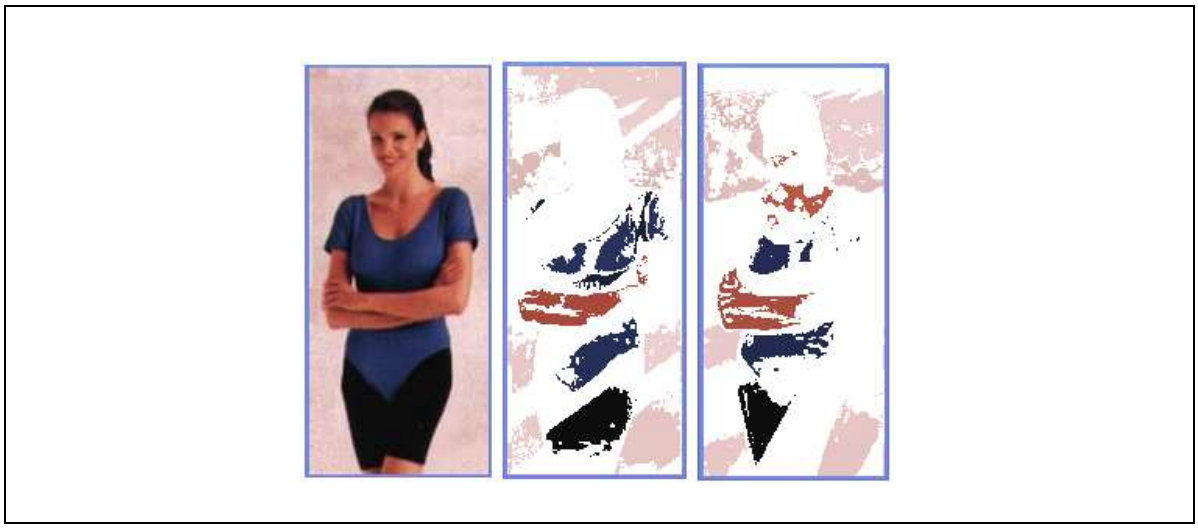

FIG. 4.15 – Images obtenues après la première phase de l'algorithme

Cette image contient 32,000 vecteurs dans $\mathbb{R}^5$. Les dimensions correspondent aux deux coordonnées spatiales x et y ainsi qu'aux trois composants de couleurs du pixel R, G et B (red, green and blue). Les deux images de droite de la même figure correspondent aux deux classifications obtenues seulement avec l'algorithme de projections aléatoires (sans les post-processus). Nous avons obtenu respectivement 10 et 12 classes.

Les deux images de droite de la figure 4.16 présentent les résultats obtenus après avoir appliqué le post-processus de *bi-modulation*. La bi-modulation a permis de rajouter une classe à chaque classification, ainsi qu'un grossissement des classes déjà existantes. L'apport de ce post-processing peut être visuellement constaté pour certaines parties des images. Néanmoins, une grande partie des points, correspondant aux régions blanches sur les images, reste non classée.

FIG. 4.16 – Images obtenues avec le post-processing de bi-modulation

Pour terminer l'exécution de notre algorithme, nous avons utilisé notre post-processing d'ε-*expansion* sur les rectangles minimum englobant. Les classes finalement obtenues sont visibles sur les deux dernières images de la figure 4.17.

La qualité visuelle des deux dernières images montre une nette amélioration par rapport aux images de la figure 4.15. Cette observation montre l'importance des algorithmes de post-processus lorsque les centres des classes de densité élevée sont trouvées par projections aléatoires. Ces résultats montrent aussi que notre algorithme pourrait être utile dans le domaine de la segmentation d'image.

FIG. 4.17 – Images obtenues avec le post-processus d'ε-expansion

4.2.4 Conclusions sur notre classification par projections aléatoires

Nous avons proposé dans cette section une nouvelle méthode de classification utilisant les projections aléatoires. Notre algorithme est divisé en deux grandes parties :

- la première partie utilise les projections aléatoires pour créer des histogrammes unidimensionnels et les combiner pour trouver des régions de forte densité dans l'espace et ainsi obtenir nos classes initiales ;
- la seconde partie concerne l'application de deux algorithmes post-processus, l'un appelé *bi-modulation* combinant les informations de deux exécutions de la première partie et l'autre appelé ε*-expansion* qui étend les classes obtenues.

Nous avons montré théoriquement et expérimentalement que notre algorithme avait une complexité asymptotique de $O(N \log N)$, où N est le nombre de données sujettes à la classification. Nous étudierons aussi plus amplement différentes méthodes de post-processing qui pourraient s'avérer plus intéressantes selon les domaines d'application. Cependant, la première des applications intéressantes que nous avons testée est l'intégration de notre classification par projections aléatoires dans RPyR, une structure d'indexation multidimensionnelle optimisée, à la place de k-means.

4.3 RPyR : indexation multidimensionnelle optimisée

4.3.1 Algorithmes

Notre méthode de classification par projection propose une solution pour répondre aux problèmes de la méthode k-means : le choix du nombre K de classes, la possibilité de gérer des bases de données bruitées et enfin trouver des classes de forme autre que sphérique (ou globulaire).

Nous proposons une nouvelle technique d'indexation RPyR qui tire profit des avantages de notre méthode de classification par projection aléatoire et de PyrRec présentée dans la section 3.3.

L'**algorithme d'indexation** se divise en plusieurs étapes :

- K classes et un ensemble de points non classés que nous considérons comme du bruit sont obtenus avec notre méthode de classification par projection aléatoire.
- PyrRec indexe chaque classe trouvée dans un arbre B+.
- Les données "bruit" sont indexées de la même manière qu'une classe dans un arbre B+.
- Les K+1 arbre B+ sont stockés en mémoire avec leur rectangle minimum englobant et leur "médoïde", le point le plus proche du centre de gravité du cluster, pour un accès rapide aux bonnes classes lors de la recherche.

L'**algorithme de recherche** procède comme suit :

- A partir d'un point requête q, une "fenêtre" requête est formée en fonction de la sélectivité choisie.
- Tous les arbres B+ concernés par la recherche sont trouvés grâce à la position de la requête et les informations conservées par chaque arbre B+ (représentant une classe).
- Pour chaque classe atteinte, la requête est transformée et détermine les tranches des pyramides qu'il faut scanner.
- Enfin, avec un accès aux feuilles de l'arbre B+, les données "résultats" sont récupérées. Dans le cas où un point aurait des index multiples, un simple test de comparaison par index suffit pour savoir si le point est à l'intérieur ou à l'extérieur de la fenêtre requête.

Nous remarquons que les opérations supplémentaires et nécessaires lors d'une recherche dans notre arbre sont deux tests sur des nombres entiers pour les points ayant plusieurs index.

Ce coût est négligeable par rapport au coût du traitement inutile des données gagné grâce à la récurrence.

4.3.2 Expérimentations

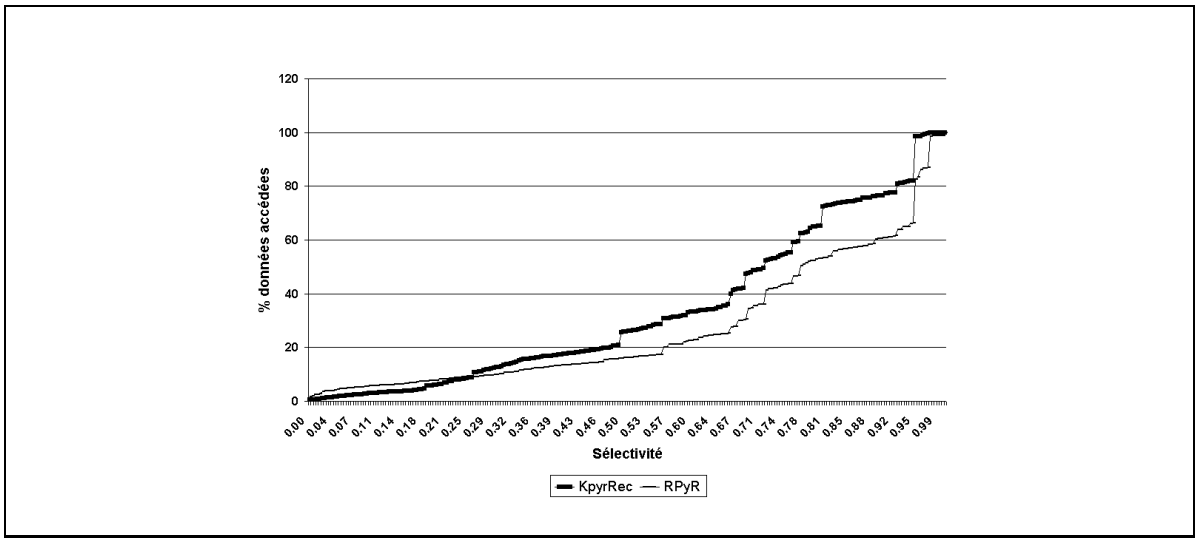

FIG. 4.18 – % données accédées en fonction de la sélectivité pour une base de données de dimension 20

Par ailleurs, nous avons présenté dans la section 3.3.4 de nombreux résultats comparant notre méthode d'indexation multidimensionnelle KpyrRec à quelques méthodes de l'état de l'art ainsi qu'à la recherche séquentielle. Certains de ces résultats sont présentés section 3. L'une des conclusions les plus importantes que nous avons pu faire est que le temps de réponse total d'une requête par fenêtrage est proportionnel au nombre de données accédées. C'est pourquoi nous présentons dans cette section des courbes du pourcentage de données accédées par rapport au nombre total de points dans notre base de données.

Les figures 4.18 à 4.21 montrent une étude comparative entre le % de données accédées par KpyrRec et notre nouvelle méthode RPyR sur une base de données synthétique contenant 100000 points répartis en 20 classes de formes aléatoires. Les figures 4.18 et 4.19 montrent les expériences effectuées sur une base de données de 20 dimensions alors que les figures 4.20

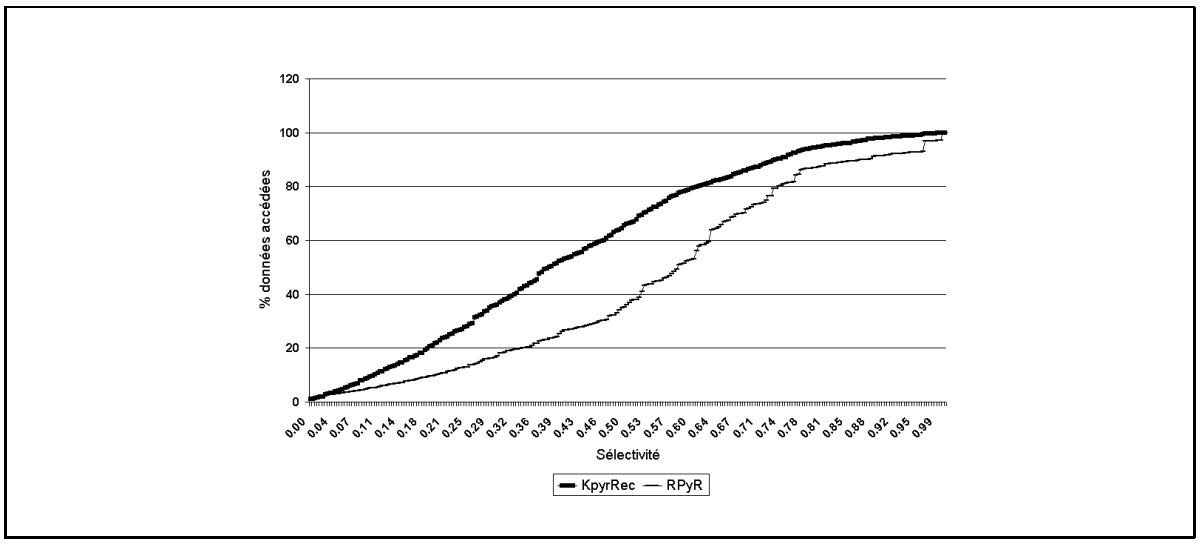

FIG. 4.19 – % données accédées en fonction de la sélectivité pour une base de données de dimension 20 avec 5% de bruit

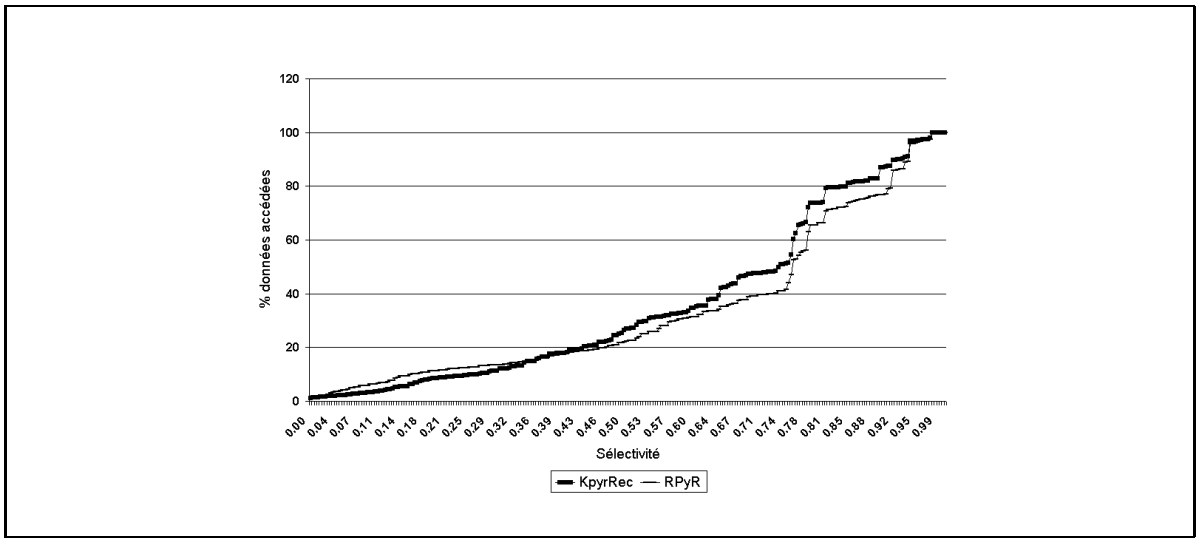

FIG. 4.20 – % données accédées en fonction de la sélectivité pour une base de données de dimension 80

et 4.21 montrent celles effectuées sur une base de données de 80 dimensions. Pour les figures 4.19 et 4.21, nous avons rajouté 5% de données "bruit" réparties aléatoirement dans l'ensemble

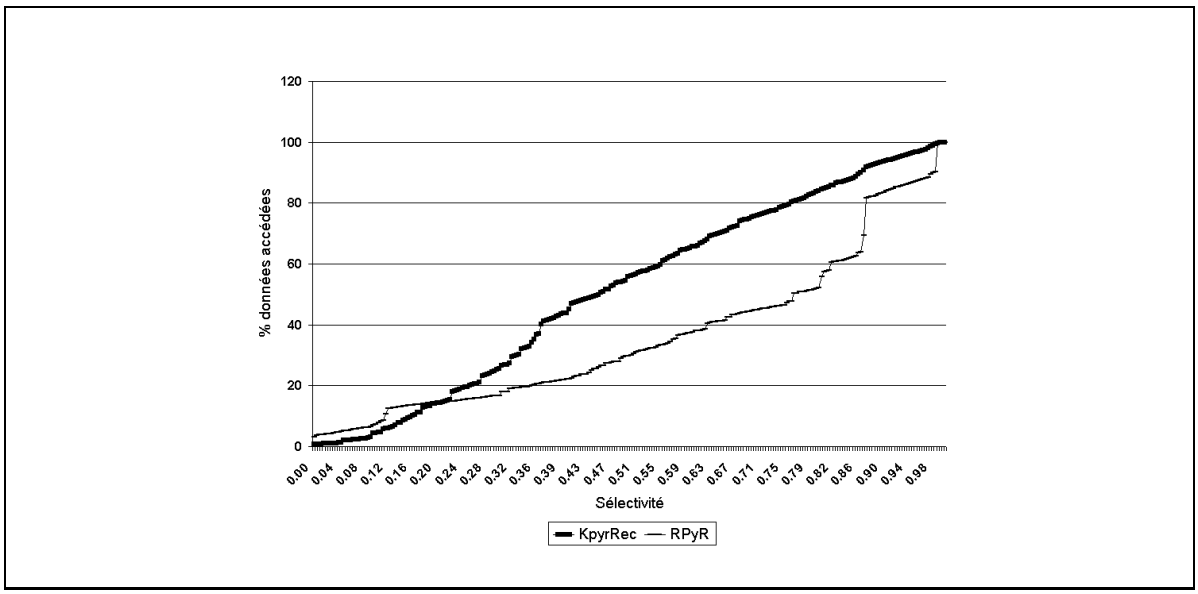

FIG. 4.21 – % données accédées en fonction de la sélectivité pour une base de données de dimension 80 avec 5% de bruit

de l'espace de données.

L'ensemble de ces expérimentations montre l'avantage considérable apporté par l'utilisation de notre algorithme de projection aléatoire par rapport à k-means, nous obtenons une baisse du volume de données accédées comprise en moyenne entre 10 et 50% entre KpyrRec et RPyR. L'algorithme de recherche étant le même pour les deux méthodes, le temps de recherche est proportionnel à ces résultats. Une meilleure classification des données et une indexation adaptée améliore la rapidité d'une requête par une diminution d'accès à des données inutiles.

4.4 Conclusion

Nous avons présenté, dans ce chapitre, une nouvelle approche de classification des données combinant des idées de classification par densité et des projections aléatoires. Nous avons ensuite justifié par une série d'expérimentations les bonnes performances en terme de qualité et rapidité de classification de notre approche. Enfin nous avons associé notre classification par

projections aléatoires avec la structure d'indexation PyrRec pour obtenir RPyR, une nouvelle méthode d'indexation multidimensionnelle. Cette association rend notre méthode plus robuste à des répartitions des données classables mais de forme pas nécessairement sphérique et gérant dans le même temps la possibilité d'avoir des données "bruit". Nos expérimentations ont mis en valeur les performances de RPyR par rapport à KpyrRec. En effet le nombre de données accédées lors une requête a nettement diminué ce qui prouve une meilleure classification par notre méthode.

Dans un futur proche, nous continuerons nos recherches et expérimentations sur RPyR en l'appliquant sur différentes bases de données réelles afin de tester et valider la robustesse de notre algorithme sur des répartitions de données aléatoires.

Bibliographie

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM-SIGMOD Int. Conf. Management of Data,*, pages 94–105, 1998.

[AM04] P. Agarwal and N. H. Mustafa. k-means projective clustering. In *Proceedings of PODS*, pages 155–165, 2004.

[APW+99] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 61–72, 1999.

[CUDW02] A. B. Chaudri, R. Unland, C. Djeraba, and W.Lindner, editors. *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops*, volume LNCS 2490 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2002.

[DG99] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. Technical Report TR-99-006, International Computer Science Institute, 1999.

[Dje03] C. Djeraba, editor. *Multimedia Mining - A Highway to Intelligent Multimedia Documents*. Kluwer, Boston, 2003.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, 1996.

[FM88] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *J. Comb. Theory B*, 44 :355–362, 1988.

[JD88] A. K. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[JL84] W. B. Johnson and J. Lindenstrauss. Extensions of lipshitz mappings into hilbert spaces. *Contemporary Mathematics*, 26 :189–206, 1984.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : A review. *ACM Computing Surveys*, 31 :264–323, 1999.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, 1967. University of California Press.

[Mon] P. Mondrian. http ://artchive.com/artchive/M/mondrian.html.

[SEKX98] J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Density-based clustering in spatial databases : The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery, an International Journal*, 2 :169–194, 1998.

[TSK06] P. N ; Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson/Addison-Wesley, Boston, 2006.

[UBD06] T. Urruty, F. Belkouch, and C. Djeraba. Indexation multidimensionnelle : Kpyrrec, une amélioration de kpyr. In *22ème journées Informatique des Organisation et Systèmes d'Information et de Décision (Inforsid06)*, pages 831–846, Hammamet, Tunisie, 2006.

[ZSD02] O. R. Zaïane, S. Simoff, and C. Djeraba, editors. *Mining Multimedia and Complex Data*, volume LNAI 2797 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 2002.

Chapitre 5

Distance Structurelle pour documents XML

Sommaire

La recherche d'information dans une collection de documents structurés (documents XML) est un domaine très vaste. La collection peut être une union de collections de documents ayant chacun sa propre DTD, ou n'ayant pas de DTD du tout. Dans ce cas, la collection de documents XML est dite hétérogène. De nombreuses méthodes pour la recherche d'information structurée ont été proposées récemment selon l'hétérogénéité des données. La thèse de Karen Sauvagnat [Sau05] propose un état de l'art complet de la recherche d'information dans des documents structurés. Il présente ses travaux de recherche sur les problématiques de recherche d'information dans des collections XML, en particulier, le modèle XFIRM [SBC04], un modèle flexible pour la recherche dans des documents semi-structurés, et son langage de requête.

Travailler sur des collections de données hétérogènes est très complexe, mais notre problématique de recherche se limite à une collection de documents XML homogène. En effet, notre collection XML est issue de la norme Mpeg-7. De plus l'ensemble des descripteurs utilisés a été choisi par notre partenaire métier pour annoter toutes les séquences vidéo de notre base de données. Notre objectif est d'accéder le plus rapidement possible aux informations contenues dans ces documents XML pour répondre aux requêtes des utilisateurs. Nos recherches dans ce domaine n'en sont qu'à leur commencement. La première idée que nous développons dans ce chapitre concerne la classification structurelle de documents XML. Cette idée n'est qu'une première étape pour accélérer l'accès à l'information lors des recherches. Les étapes suivantes, incluant la recherche d'information et l'optimisation de l'utilisation de langages de requêtes adaptés tels que XQuery ou XQL, font parties de nos perspectives de recherche.

Nous proposons une distance structurelle entre documents XML basée sur les multi-ensembles de chemins étiquetés. Nous démontrons que notre distance structurelle vérifie les trois propriétés d'une distance mathématique : symétrie, séparation et inégalité triangulaire. Ces propriétés justifient l'utilisation de notre proposition de cette distance structurelle dans une méthode de classification.

Dans ce chapitre, nous introduisons tout d'abord des notions sur les multi-ensembles avant de proposer une nouvelle mesure de distance entre deux structures de documents XML. Nous verrons que cette distance respecte les principes mathématiques de base, exploite la structure des documents et reste d'une complexité praticable. Enfin, nous présentons quelques résultats expérimentaux d'une classification utilisant notre distance sur une base de données synthé-

tiques de documents XML.

5.1 Notre proposition de distance structurelle

Dans cette section, nous introduisons tout d'abord quelques définitions et propriétés liées aux multi-ensembles que nous appliquons ensuite aux multi-ensembles de chemins d'arbre étiqueté. Enfin nous donnons notre proposition de distance structurelle.

5.1.1 Introduction aux multi-ensembles

Definition 5.1.1. *Un multi-ensemble d'un ensemble X est une application $M : X \longrightarrow \mathbb{N}$. Le nombre $M(x)$ est la multiplicité de x dans M. Si $M(x) > 0$ alors x est un élément de M.*

Si l'ensemble $\{x \in X | M(x) > 0\}$ est fini, alors le multi-ensemble M est fini. La cardinalité de l'ensemble $\{x \in X | M(x) > 0\}$ est la taille du multi-ensemble M, noté taille(M).

On note un multi-ensemble fini M dans X comme la somme formelle

$$M = m_1 x_1 + \cdots + m_k x_k,$$

où $x_1, \ldots, x_k$ sont les éléments distincts de l'ensemble $\{x \in X | M(x) > 0\}$.

Si M est un multi-ensemble de l'ensemble X et $X \subset Y$, on définit l'extension naturelle M_Y de M pour Y

$$M_Y(y) = \begin{cases} M(y) & \text{si } y \in X, \\ 0 & \text{sinon.} \end{cases}$$

On note M_Y comme l' *extension vide* de M pour Y.

L'union, l'intersection et la différence symétrique de deux multi-ensembles sont définies de telle sorte qu'elles généralisent les opérations habituelles de la théorie des ensembles. Soient M, P deux multi-ensembles de l'ensemble X :

- l'*union de M et P* est le multi-ensemble $M \cup P$ tel que $(M \cup P)(x) = \max\{M(x), P(x)\}$;
- l'*intersection de M et P* est le multi-ensemble $M \cap P$ tel que $(M \cap P)(x) = \min\{M(x), P(x)\}$ pour $x \in X$;

– la *différence symétrique* $M \oplus P$ est définie par $(M \oplus P)(x) = |M(x) - P(x)|$ pour $x \in X$. On remarque que ce n'est pas une opération associative car en général $||a-b|-c| \neq |a-|b-c||$ (par exemple, $||7-5|-3| = 1$ et $|7-|5-3|| = 5$).

Dans le cas de $\mathscr{P}(S)$, l'ensemble des sous-ensembles d'un ensemble donné S, la cardinalité de la différence symétrique de deux sous-ensembles est une métrique dans $\mathscr{P}(S)$.

Soient M, P deux multi-ensembles finis dans X, on définit les nombres

$$\delta_k(M,P) = \left(\sum_{x \in X} |M(x) - P(x)|^k \right)^{\frac{1}{k}}$$

Ensuite, à partir de l'inégalité de Minkowski, nous déduisons que pour tout k, δ_k est une métrique sur un ensemble de multi-ensembles. En particulier, pour $k = 1$ nous avons la métrique :

$$\delta_1(M,P) = \sum_{x \in X} |M(x) - P(x)| = \sum_{x \in X} (M \oplus P)(x).$$

Nous avons besoin d'une généralisation de cette métrique. Supposons que $w : X \longrightarrow \mathbb{R}_{\geq 0}$ est une fonction de pondération définie dans X. Nous définissons d_w comme

$$d_w(M,P) = \sum_{x \in X} w(x)|M(x) - P(x)| = \sum_{x \in X} w(x)(M \oplus P)(x)$$

pour $x \in X$. d_w est une métrique sur l'ensemble des multi-ensembles définis dans X, quelle que soit la fonction de pondération choisie w.

5.1.2 Multi-ensemble de chemins d'un arbre étiqueté

Un *arbre* est un graphe connexe ne contenant aucun cycle $\mathscr{A} = (V,E)$, où V est l'ensemble des sommets de l'arbre et E l'ensemble des étiquettes de l'arbre ; un *arbre-racine* est la paire $(\mathscr{A}, v_0)$, où v_0 est le sommet appelé aussi la *racine* de l'arbre. Un *arbre-racine étiqueté* $\mathscr{A}$ est un 4-tuple $(\mathscr{A}, v_0, \ell, E)$, où $(\mathscr{A}, v_0)$ est un arbre-racine, $\ell : V \longrightarrow E$ est une application, et E est un ensemble dont les éléments sont des étiquettes ; $\ell(v)$ est l'*étiquette* du sommet v.

L'ensemble des séquences finies d'éléments d'un ensemble E est noté $\mathbf{seq}(E)$. Un *chemin-*

racine étiqueté dans un arbre-racine étiqueté $(\mathscr{A}, v_0, \ell, E)$ est une séquence d'étiquettes $\mathbf{l} = (e_0, e_1, \ldots, e_n) \in \mathbf{seq}(E)$ telle qu'il existe un chemin $(v_0, v_1, \ldots, v_n)$ dans $\mathscr{A}$ et $\ell(v_i) = e_i$ pour $0 \leq i \leq n$. Pour chaque sommet de l'arbre-racine, il existe un chemin unique commençant par v_0 et terminant par v et pour chacun de ces sommets il existe un chemin-racine étiqueté qui se termine par $\ell(v)$.

Contrairement à la pratique standard de la théorie des graphes, nous définissons la longueur du chemin-racine $\mathbf{l} = (e_0, e_1, \ldots, e_n)$ simplement par la longueur $n+1$ de la séquence.

Pour un multi-ensemble $M = m_1\mathbf{p}_1 + \cdots + m_k\mathbf{p}_k$ de séquences d'éléments de E et une séquence r, nous définissons le multi-ensemble rM par

$$rM = m_1 r\mathbf{p}_1 + \cdots + m_k r\mathbf{p}_k,$$

où $r\mathbf{p}_i$ est la séquence obtenue en concaténant r et $\mathbf{p}$.

Le multi-ensemble de chemins-racine étiquetés, noté $\mathsf{CRE}(\mathscr{A}, v_0, \ell, E)$, d'un arbre-racine étiqueté $(\mathscr{A}, v_0, \ell, E)$ est un multi-ensemble de séquences d'étiquettes. Cet ensemble peut être défini récursivement comme suit :

1. Si $\mathscr{A} = (\{v_0\}, \emptyset)$ et $\ell(v_0) = a$, alors $\mathsf{CRE}(\mathscr{A}, v_0, \ell) = 1(a)$.

2. Supposons que les descendants immédiats de v_0 dans $\mathscr{A}$ soient $v_1, \ldots, v_m$ avec $\ell(v_i) = a_i$, et les sous-arbres de $\mathscr{A}$ ayant les racines dans $v_1, \ldots, v_m$ soient $\mathscr{A}_1, \ldots, \mathscr{A}_m$, respectivement. Alors,
$$\mathsf{CRE}(\mathscr{A}, v_0, \ell, E) = 1(a) + \sum_{i=1}^{m} a\mathsf{CRE}(\mathscr{A}_i, v_i, \ell_i, E).$$

Example 5.1.2. Prenons pour exemple les arbres-racine étiquetés de la figure 5.1. Leurs multi-ensembles respectifs sont donnés par :

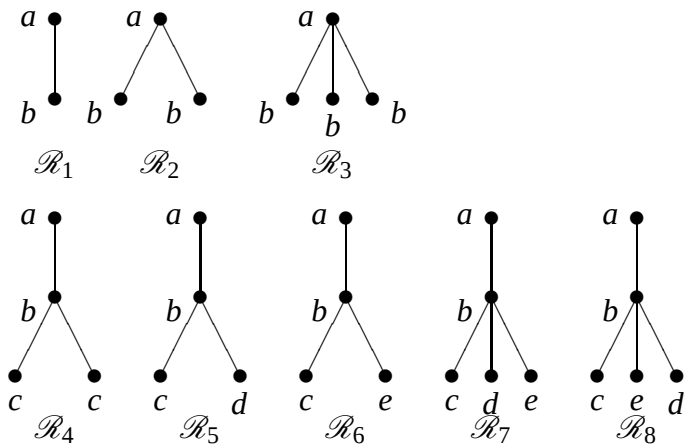

FIG. 5.1 – Exemples d'arbres-racine étiquetés

Arbre	Multi-ensembles de chemins-racine étiquetés
$\mathscr{R}_1$	1(a)+ 1(a,b)
$\mathscr{R}_2$	1(a)+ 2(a,b)
$\mathscr{R}_3$	1(a)+ 3(a,b)
$\mathscr{R}_4$	1(a)+ 1(a,b)+2(a,b,c)
$\mathscr{R}_5$	1(a)+ 1(a,b)+1(a,b,c)+1(a,b,d)
$\mathscr{R}_6$	1(a)+ 1(a,b)+1(a,b,c)+1(a,b,e)
$\mathscr{R}_7$	1(a)+1(a,b)+1(a,b,c)+1(a,b,d)+1(a,b,e)
$\mathscr{R}_8$	1(a)+1(a,b)+1(a,b,c)+1(a,b,e)+1(a,b,d)

□

Une caractérisation des multi-ensembles de chemins d'arbre-racine est donné ci dessous. Rappelons qu'une séquence **u** est un *préfixe* d'une séquence **v** si il existe une séquence **w** telle que l'égalité $\mathbf{v} = \mathbf{uw}$ soit vraie. De plus, **u** est un *préfixe strict* de **v** si **u** est un préfixe de **v** et $\mathbf{u} \neq \mathbf{v}$.

Theorem 5.1.3. *Soit l'ensemble E, un multi-ensemble fini de séquences L sur $\boldsymbol{seq}(E)$ est un multi-ensemble de chemins d'un arbre-racine étiqueté si et seulement si les conditions suivantes sont satisfaites :*

(i) *il existe une séquence* (a) *avec* $L((a)) = 1$ *qui est un préfixe strict de chaque séquence* $\mathbf{p}$ *avec* $L(\mathbf{p}) > 0$ *;*

(ii) *pour chaque préfixe* $\mathbf{r}$ *d'une séquence* $\mathbf{p}$ *avec* $L(\mathbf{p}) > 0$ *nous avons* $L(\mathbf{r}) > 0$.

preuve Supposons que les conditions du théorème soient remplies. Nous devons prouver l'existence d'un arbre-racine étiqueté $\mathscr{R}_L = (\mathscr{A}_L, v_0, \ell, E)$ tel que $\mathsf{CRE}(\mathscr{R}_L) = L$.

Les sommets de $\mathscr{A}$ sont indexés par des séquences de $\mathbf{seq}(E)$ telles que $L(\mathbf{p}) > 0$ et la racine de l'arbre est la séquence $v_{(a)}$. Notons que la première condition implique que la séquence (a) est l'unique séquence avec cette propriété.

Supposons que $\mathbf{p}, \mathbf{q}$ sont deux séquences distinctes dans $\mathbf{seq}(E)$ telles que $\mathbf{p}$ soit un préfixe de $\mathbf{q}$ et $L(\mathbf{q}) > 0$. Si $\mathbf{p}$ est une séquence de longueur maximale avec ces propriétés, alors il existe au moins un élément $e \in E$ tel que l'égalité $\mathbf{q} = \mathbf{p}(e)$ soit vraie. Dans le cas contraire, il existe alors une séquence $\mathbf{r}$ telle que $\mathbf{r}$ soit un préfixe de $\mathbf{q}$ et $\mathbf{p}$ soit un préfixe de $\mathbf{r}$ et nous avons $\mathbf{r} \neq \mathbf{q}$ et $\mathbf{r} \neq \mathbf{p}$. La seconde condition du théorème nous donne $L(\mathbf{r}) > 0$; Or $|\mathbf{r}| > |\mathbf{p}|$, ce qui contredit le fait que $\mathbf{p}$ soit de longueur maximale. Soit une paire $(v_{\mathbf{p}}, v_{\mathbf{q}})$ formant une arête (edge) dans $\mathscr{A}_L$, toutes les arêtes du graphe auront cette forme. Nous montrons alors que cet argument implique que pour chaque sommet $v_{\mathbf{q}}$ il y a un unique chemin qui commence par $v_{(a)}$ et se termine par $v_{\mathbf{q}}$. Donc $\mathscr{A}_L$ est bien un arbre. La fonction ℓ est donnée par $\ell(v_{\mathbf{q}}) = e$, où e est le dernier élément de la séquence $\mathbf{q}$. Ceci complète la définition de $\mathscr{R}_L$.

Nous devons ensuite vérifier que $\mathsf{CRE}(\mathscr{R}_L) = L$, ce qui est induit directement par le nombre n de sommets de l'arbre sous-jacent $\mathscr{R}_L$.

Pour le cas $n = 1$, la vérification est immédiate. Supposons alors que l'égalité soit vérifiée pour des arbres contenant moins de n sommets et que $\mathscr{A}_L$ soit l'arbre sous-jacent de $\mathscr{R}_L$. Soient $\mathscr{A}_1, \ldots, \mathscr{A}_m$ les sous-arbres immédiats de $\mathscr{R}_L$ et $\mathscr{R}_1, \ldots, \mathscr{R}_m$ les arbres-racine étiquetés correspondant, alors $\mathscr{R}_i = (\mathscr{A}_i, v_{(b_i)}, \ell_i, E)$. Soit K_i le multi-ensemble de chemins-racine étiquetés de $\mathscr{R}_i$, si nous construisons l'arbre-racine étiqueté de K_i comme nous l'avons fait pour L, alors $\mathscr{R}_{K_i}$ coïncide avec $\mathscr{R}_i$. Par l'hypothèse formulée, nous avons $\mathsf{CRE}(\mathscr{R}_i) = \mathsf{CRE}(\mathscr{R}_{K_i}) = K_i$. Or $L = (a) + \sum_{i=1}^{m} (a) K_i = (a) + (a) \sum_{i=1}^{m} \mathsf{CRE}(\mathscr{R}_i)$, nous obtenons alors $L = \mathsf{CRE}(\mathscr{R})$.

5.1.3 L'espace métrique des arbres étiquetés

Nous introduisons dans cette section une mesure de dissimilarité entre des arbres-racine étiquetés ayant un ensemble d'étiquettes E utilisant les multi-ensembles de chemins-racine étiquetés et une mesure définie par la classe de ces multi-ensembles qui utilise une fonction de pondération.

L'utilisation des multi-ensembles de chemins étiquetés dans notre proposition de mesure de dissimilarité permet de prendre en compte la multiplicité des chemins similaires dans un arbre et de donner de l'importance aux noeuds proches de la racine par rapport à ceux proches des feuilles. L'utilisation des chemins à partir de la racine permet aussi de différencier deux noeuds ayant une même étiquette en fonction de leurs noeuds père.

Definition 5.1.4. *Soit $\mathscr{R} = (\mathscr{A}, v_0, \ell, E)$ et $\mathscr{R}' = (\mathscr{A}', v_0', \ell', E)$ deux arbres-racine étiquetés, nous définissons la dissimilarité entre $\mathscr{R}$ et $\mathscr{R}'$:*

$$d(\mathscr{R},\mathscr{R}') = \sum_{\boldsymbol{p}\in \boldsymbol{seq}(E)} 2^{-length(\boldsymbol{p})}(CRE(\mathscr{R}) \oplus CRE(\mathscr{R}'))(\boldsymbol{p}).$$

d est une semi distance basée sur la classe des arbres-racine étiquetés. C'est à dire,

- $d(\mathscr{R},\mathscr{R}) = 0$,
- $d(\mathscr{R},\mathscr{R}') = d(\mathscr{R}',\mathscr{R})$ et
- $d(\mathscr{R},\mathscr{R}'') \leq d(\mathscr{R},\mathscr{R}') + d(\mathscr{R}',\mathscr{R}'')$ pour chaque $\mathscr{R},\mathscr{R}',\mathscr{R}''$.

Cependant, si $d(\mathscr{R},\mathscr{R}') = 0$ alors $\mathscr{R},\mathscr{R}'$ peuvent être relativement différents par l'ordre des descendants d'un sommet. Si nous identifions deux arbres-racine étiquetés qui ne sont différents que de ce point de vue, alors d est une distance.

Par exemple, la mesure de dissimilarité entre $\mathscr{R}_1$ et $\mathscr{R}_2$ est égale à
$d(\mathscr{R}_1,\mathscr{R}_2) = \frac{1}{2} * (1(a) - 1(a)) + \frac{1}{2^2} * (1(a,b) - 2(a,b)) = \frac{1}{2} * (0) + \frac{1}{4} * (1(a,b)) = \frac{1}{4}$.

Les autres mesures de dissimilarité entre les arbres-racine étiquetés $\mathscr{R}_1,\ldots,\mathscr{R}_8$ de la figure 5.1 sont présentées dans le tableau 5.1.

	$\mathscr{R}_1$	$\mathscr{R}_2$	$\mathscr{R}_3$	$\mathscr{R}_4$	$\mathscr{R}_5$	$\mathscr{R}_6$	$\mathscr{R}_7$	$\mathscr{R}_8$
$\mathscr{R}_1$	0	0.25	0.5	0.25	0.25	0.25	0.375	0.375
$\mathscr{R}_2$	0.25	0	0.25	0.5	0.5	0.5	0.625	0.625
$\mathscr{R}_3$	0.5	0.25	0	0.75	0.75	0.75	0.875	0.875
$\mathscr{R}_4$	0.25	0.5	0.75	0	0.25	0.25	0.375	0.375
$\mathscr{R}_5$	0.25	0.5	0.75	0.25	0	0.25	0.125	0.125
$\mathscr{R}_6$	0.25	0.5	0.75	0.25	0.25	0	0.125	0.125
$\mathscr{R}_7$	0.375	0.625	0.875	0.375	0.125	0.125	0	0
$\mathscr{R}_8$	0.375	0.625	0.875	0.375	0.125	0.125	0	0

TAB. 5.1 – Matrice de dissimilarité entre les arbres-racine étiquetés

5.2 Expérimentations

Afin de tester l'efficacité de notre mesure d, nous avons réalisé une classification de documents XML extraits d'une même DTD. Nous utilisons la DTD présentée figure 2.2 ainsi que l'exemple de document XML de la figure 2.1. Cette DTD est une DTD simplifiée d'une base de données de documents vidéo. Chaque document XML respectant cette DTD représente une vidéo, découpée en chapitres (tag *Chapters*) et contenant des informations relatives au propriétaire (tag *Companies*), aux acteurs (tag *Actors* et *Actor*), etc.

Pour construire un ensemble de documents XML répartis en classes différenciées selon la structure des documents, nous utilisons les éléments « multiple » de notre DTD, c'est à dire les éléments qui peuvent avoir une multiplicité différente représentée par le $*$ dans la ligne $<!ELEMENT\,Images(Image)*>$. Cette ligne de la DTD signifie que le noeud *Images* de chaque document XML que nous générons peut contenir entre 0 et une infinité de noeuds fils *Image*. Nous partons de l'hypothèse qu'une classe de documents XML est un ensemble de documents XML ayant à peu près la même multiplicité pour chaque noeud. Nous utilisons, pour générer notre base de documents XML en différentes classes, un tableau de différenciation contenant les valeurs inférieures et supérieures des multiplicités de chaque élément « multiple ».

Le tableau 5.2 présente les propriétés des quatre classes de documents XML de structure différente. Pour chacun des attributs considérés, nous définissons une multiplicité minimale (c_{x_m}) et une multiplicité maximale (c_{x_M}). Tous les documents appartenant à une classe res-

intervalles des classes	$c1_m$	$c1_M$	$c2_m$	$c2_M$	$c3_m$	$c3_M$	$c4_m$	$c4_M$
sstitre	1	1	3	5	3	5	1	2
langue	3	5	3	7	1	2	3	4
acteur	10	50	50	100	10	20	1	10
sponsors	1	5	2	3	4	4	1	1
chap	10	20	2	5	2	8	10	15
image	3	5	1	2	3	3	4	4
mot	1	1	2	2	3	3	4	10

TAB. 5.2 – Valeurs des bornes des intervalles pour chaque classe

pectent ces valeurs et ont pour chacun de leurs attributs, une multiplicité incluse entre ces valeurs. Notons qu'il est possible d'avoir des recouvrements des valeurs des éléments multiples. Par exemple, les documents XML de la classe 1 contiennent entre 10 et 50 noeuds *Actor* et ceux de la classe 3, 10 à 20. Nous avons réalisé des tests de classification de documents XML avec des classes de documents sans recouvrement des valeurs des éléments multiples. Les résultats de la classification hiérarchique ascendant sont parfaits dans ces conditions et ne sont pas détaillés dans cette section. Pour effectuer nos tests, dont les résultats sont présentés ci dessous, nous utilisons une base de données de 100 documents XML répartis en quatre classes de 20, 30, 40 et 10 documents XML respectivement. Pour simplifier la reconnaissance des documents et leur appartenance à une classe, nous notons que les doc_101 à doc_120 appartiennent à la classe 1, les doc_221 à doc_250 à la classe 2, les doc_351 à doc_390 à la classe 3 et enfin les doc_491 à doc_4100 à la classe 4.

Nous utilisons un algorithme de classification hiérarchique ascendant (CAH) à partir de la matrice des distances des documents XML deux à deux. L'algorithme de CAH nous donne ses résultats sous forme d'un dendogramme, i.e. un arbre de classification (figure 5.2). Si nous prenons les quatre premières classes de cet arbre, l'algorithme CAH basé sur notre distance structurelle répartit les documents XML dans les classes du tableau 5.3. Nous observons que les quatre premières classes trouvées par l'algorithme CAH ne sont pas celles que nous avons créées. Les classes 3 et 4 sont groupées tandis que la classe 2 a été séparée en deux classes.

On observe que 3 documents de la classe 1 ($doc_1 04$, $doc_1 05$ et $doc_1 10$) ont été groupés à la classe 2. Si nous prenons les cinq premières classes de l'algorithme CAH, les classes 3 et 4 se séparent correctement.

Si nous prenons les huit premières classes de l'arbre de classification, l'algorithme CAH répartit les documents XML selon les classes représentées dans le tableau 5.4. Nous observons que les classes 1, 2 et 3 sont uniquement composées par des documents de la classe 1 originelle. Les classes 4, 5 et 6 proviennent des documents de la classe 2. Les classes 7 et 8 correspondent respectivement aux classes 3 et 4 d'origine. Nous retrouvons finalement nos classes d'origine décomposées en plusieurs classes et aucune erreur de classification n'est observée.

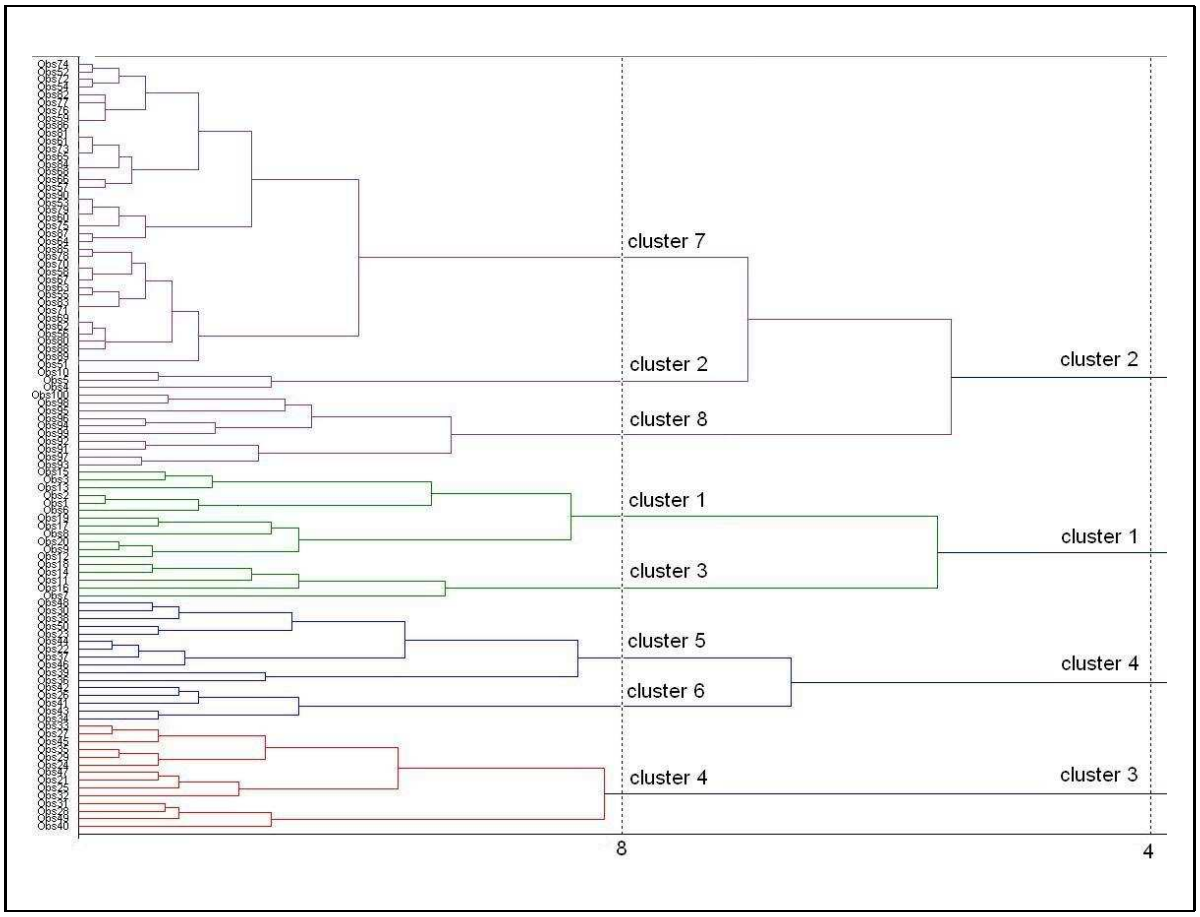

FIG. 5.2 – Dendogramme résultat de l'algorithme CAH pour 4 et 8 classes

classe 1	classe 2	classe 3	classe 4
$doc_1 01 \rightarrow doc_1 03$	$doc_1 04 \rightarrow doc_1 05$	$doc_2 21$; $doc_2 24$	$doc_2 22$; $doc_2 23$
$doc_1 06 \rightarrow doc_1 09$	$doc_1 10$	$doc_2 27 \rightarrow doc_2 29$	$doc_2 26$; $doc_2 30$
$doc_1 11 \rightarrow doc_1 20$	$doc 51 \rightarrow doc 100$	$doc_2 31 \rightarrow doc_2 33$	$doc_2 34$; $doc_2 46$
		$doc_2 25$; $doc_2 35$	$doc_2 36 \rightarrow doc_2 39$
		$doc_2 40$; $doc_2 45$	$doc_2 41 \rightarrow doc_2 44$
		$doc_2 47$; $doc_2 49$	; $doc_2 48$; $doc_2 50$

TAB. 5.3 – Répartition en 4 classes des documents XML par l'algorithme CAH

classe 1	classe 2	classe 3	classe 4
$doc_1 01 \rightarrow doc_1 03$	$doc_1 04 \rightarrow doc_1 05$	$doc_1 07$; $doc_1 11$	$doc_2 21$; $doc_2 24$
$doc_1 06$; $doc_1 08$	$doc_1 10$	$doc_1 14$; $doc_1 16$	$doc_2 25$; $doc_2 35$
$doc_1 09$; $doc_1 12$		$doc_1 18$	$doc_2 27 \rightarrow doc_2 29$
$doc_1 13$; $doc_1 15$			$doc_2 31 \rightarrow doc_2 33$
$doc_1 17$; $doc_1 19$			$doc_2 40$; $doc_2 45$
$doc_1 20$			$doc_2 47$; $doc_2 49$
classe 5	classe 6	classe 7	classe 8
$doc_2 22$; $doc_2 23$	$doc_2 26$; $doc_2 34$	$doc_3 51 \rightarrow doc_3 90$	$doc_4 91 \rightarrow doc_4 100$
$doc_2 30$	$doc_2 41 \rightarrow doc_2 43$		
$doc_2 36 \rightarrow doc_2 39$			
$doc_2 44$; $doc_2 46$			
$doc_2 48$; $doc_2 50$			

TAB. 5.4 – Répartition en 8 classes des documents XML par l'algorithme CAH

Nous définissons maintenant la *pureté* de l'algorithme de classification comme le pourcentage de documents XML correctement classés sur l'ensemble des documents de la classe. Un document mal classé est un document appartenant à une classe d'origine C_i et classé par l'algorithme de CAH dans une classe contenant une majorité de documents XML C_j, avec $i \neq j$. Par exemple, pour le tableau 5.3, nous avons 3 documents de la classe 1 et 10 documents de la classe 4 avec une majorité de documents de la classe 3 d'où la $purete = \frac{10}{10+3} = 87\%$. Le tableau 5.5 donne les valeurs de la pureté de l'algorithme de CAH en fonction du nombre de classes considérées.

	4 classes	5 classes	6 classes	7 classes	8 classes
purete	87 %	97%	97%	97%	100%

TAB. 5.5 – Pureté en fonction du nombre de classes

Nous observons que pour quatre ou huit classes, l'algorithme CAH basé sur la matrice des distances calculées avec notre distance structurelle propose une très bonne classification avec, quel que soit l'ensemble des documents XML générés, une pureté supérieure à 95% et de plus de 99% pour huit classes.

Les résultats obtenus valident notre proposition de distance entre la structure de deux documents XML. L'utilisation de cette distance est pertinente sur une base de données de documents XML créés à partir de la même DTD. En effet, si deux documents ont une racine différente, aucun chemin ne sera similaire aux deux documents et notre définition n'a plus réellement de signification.

5.3 Conclusion

Nous avons abordé, dans ce chapitre, une nouvelle distance basée sur les différences de structures entre documents XML. Cette distance se base sur l'utilisation de multi-ensembles de chemins pour comparer les documents selon leur structure hiérarchique. Contrairement à nombre d'autres techniques, notre proposition de distance respecte les propriétés de base des distances, notamment l'inégalité triangulaire, permettant une utilisation justifiée dans une application de classification. Une expérimentation sur des données XML synthétiques présente de très bons résultats de classification en produisant des classes pures dans lesquelles les classes de documents sont clairement différenciées. Des futurs travaux devraient nous permettre tout d'abord de valider notre approche sur de plus grands ensembles de données et d'enrichir notre mesure de similarité en prenant en compte les valeurs définies pour chacune des entrées des documents XML. Nous orienterons nos perspectives de recherches vers une méthode de recherche d'information globale incluant la classification de documents XML et la recherche d'information dans le but d'appliquer nos propositions à notre base de données de séquences

vidéo et notre moteur de recherche. Il pourrait être très intéressant d'utiliser une plate-forme de visualisation, d'analyse et de classification de données semi-structurées comme la plate-forme Tétralogie [MCD+06] pour mieux appréhender notre collection de données Mpeg-7.

Bibliographie

[MCD+06] Josiane Mothe, Claude Chrisment, Taoufiq Dkaki, Bernard Dousset, and Saïd Karouach. Combining mining and visualization tools to discover the geographic structure of a domain. *Computers, Environment and Urban Systems*, 30(4) :460–484, 2006.

[Sau05] Karen Sauvagnat. Modèle flexible pour la recherche d'information dans des corpus de documents semi-structurés., 2005. Doctorat de l'Université Paul Sabatier de Toulouse.

[SBC04] Karen Sauvagnat, Mohand Boughanem, and Claude Chrisment. Searching xml documents using relevance propagation. In *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004*, volume 3246 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004.

Chapitre 6

Amélioration de l'indexation dans un moteur de recherche vidéo

Sommaire

Ce chapitre présente l'utilisation de nos méthodes de recherche présentées dans les chapitres précédents avec une application réelle pour un moteur de recherche de séquences vidéo d'une base de films d'entreprise. Nous détaillons tout d'abord le modèle de représentation vectorielle, simple mais adapté à notre domaine applicatif. Puis nous présentons l'interface de notre moteur de recherche. Enfin nous nous intéressons à la détection d'erreurs d'indexation par l'analyse des usages des utilisateurs.

6.1 Modèle de représentation vectorielle

Les recherches sur les représentations vectorielles ont commencé depuis un certain nombre d'années sur les contenus textuels par les travaux de Salton ([SWY75], [SM84] et [Sal89]) et Van Rijsbergen ([vR79]). De nombreuses recherches d'information se sont basées sur ces modèles ou les ont transformés selon l'application souhaitée. Notre but dans cette section n'est pas de faire un état de l'art exhaustif ou de proposer un modèle de représentation vectorielle plus performant.Nous proposons ici notre modèle de représentation simple mais bien adapté au contexte de notre application : indexer une base de films d'entreprise. Notre modèle de représentation vectorielle transforme les descriptions des séquences vidéo des films contenues dans les documents Mpeg-7, en vecteurs $P_i(x_0, x_1, \ldots, x_d)$ dans un espace de dimension d. Le nombre de dimensions est fixé par l'égalité suivante $d = d_1 + d_2$, où d_1 représente le nombre de dimensions relatives aux descripteurs numériques tels que la date, le temps, la couleur dominante de l'image clé de la séquence, etc. et d_2 est le nombre de dimensions issues des descripteurs sémantiques. Le nombre d_2 provient du nombre de catégories existantes dans le thésaurus pour classer les mots clés des descriptions vidéo. Le thésaurus a été créé par les experts métiers, ce qui implique que le nombre de dimensions d est donc lié à notre base de films d'entreprise. Le modèle de représentation vectorielle que nous utilisons pour transformer nos séquences vidéo en points multidimensionnels suit les étapes suivantes :

- il récupère tous les mots clés des documents Mpeg-7 décrivant les séquences vidéo ;
- tous les noms propres qui apparaissent dans le document sont classés par catégorie : producteur, acteur, etc et sont stockés dans des tables de hachage qui seront utilisées lors d'une recherche booléenne sur un ou plusieurs noms ;

– en utilisant notre thésaurus, tous les mots-clés restants sont classés sémantiquement par catégories. Par exemple : les mots clés "montagne" et "forêt" appartiennent à la catégorie "milieu naturel".

La transformation d'une séquence vidéo en un vecteur $P_i(x_0, x_1, \ldots, x_d)$ dans l'espace de dimension d respecte la règle suivante : toutes les coordonnées x_i sont normalisées, $x_i \in [0,1]$. Si la dimension correspond à un descripteur numérique alors x_i est égale à la valeur normalisée du descripteur.

$$x_i = \frac{ValeurDescripteur}{ValeurMaximaleDescripteur} \tag{6.1}$$

Si la dimension correspond à une catégorie issue de descripteurs sémantiques, alors la valeur de x_i est une pondération mesurant l'importance de cette catégorie dans l'ensemble du document.

$$x_i = \frac{NombreMotsClesCategorie}{NombreMotsClesDocument} \tag{6.2}$$

La figure 6.1 présente un exemple de notre modèle de représentation vectorielle pour une séquence vidéo. Elle transforme une partie la description donnée dans le fichier XML en vecteur normalisé.

Après la transformation de chaque séquence vidéo de notre base de données en un point multidimensionnel, l'indexation se fait uniquement avec les coordonnées de ces points multidimensionnels de l'espace vectoriel créé. L'index représentant un point P_i correspond à la description normalisée Mpeg-7 d'une séquence vidéo.

Nous décrivons par la suite l'utilisation de ce modèle de représentation vectorielle dans notre moteur de recherche de films d'entreprise.

6.2 Moteur de recherche

L'un des principaux objectifs du projet est la mise en place d'un outil de recherche de séquences de vidéo. Cet outil a été réalisé dans l'objectif d'une utilisation par des experts métier. En effet, un film d'entreprise de 30 minutes requiert souvent plus de 5h de tournage sur le ter-

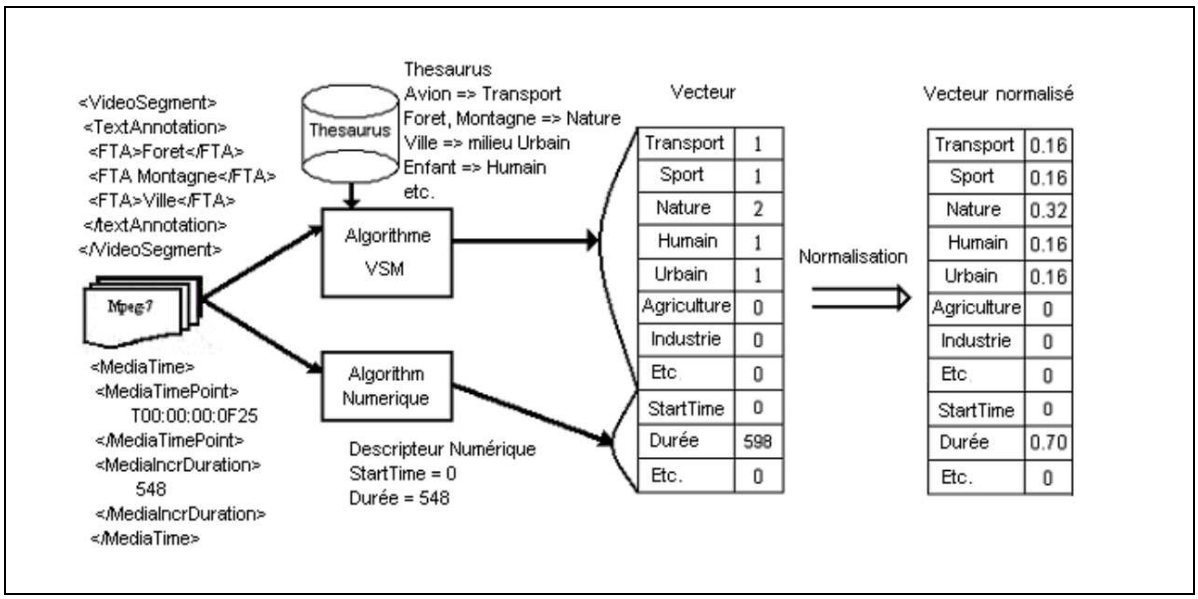

FIG. 6.1 – Modèle de représentation vectorielle

rain. Les séquences vidéo non utilisées par l'entreprise sont ensuite stockées dans une énorme base de données. Ces milliers d'heures de vidéo pourraient être utiles pour de prochaines créations de films. Une bonne gestion d'une telle base de séquences vidéo apporterait un gain de temps et d'argent. Il serait inutile d'envoyer une équipe de tournage sur le terrain filmé une séquence vidéo qui existe déjà dans la base de données.

Nous proposons donc un moteur de recherche de séquences vidéo basé sur la description du contenu. Seules les structures d'indexation Kpyr et KpyrRec ont été intégrées à ce moteur pour un accès rapide et efficace à l'information contenue dans les séquences vidéo. La partie "structure XML" n'a pas encore été implémentée et nécessiterait une mise à jour importante du moteur de recherche. L'interface de notre moteur de recherche se décompose en trois parties : une interface de requête pour l'utilisateur, l'affichage des résultats et l'affichage du compte rendu des différentes recherches pour la même requête.

L'utilisation de notre moteur de recherche s'effectue donc en trois étapes. La première étape concerne l'interface de requête. Cette interface de recherche est pour le moment limitée à une recherche par mots clés et catégories. L'utilisateur construit sa requête en choisissant

FIG. 6.2 – Interface de requêtes

une des catégories proposées par l'interface. Une liste de mot appartenant à cette catégorie est ensuite disponible, l'utilisateur peut alors choisir l'un de ces mots prédéfinis ou rentrer lui même un autre mot. Les catégories que propose l'interface de recherche sont les catégories souhaitées par les experts métier de notre base de données. Nous avons, par exemple dans la figure 6.2, la catégorie "Milieu naturel" dans laquelle la liste de mots prédéfinis contient des mots tels que "Lac", "Montagne", ou "Forêt". La requête de l'utilisateur peut être composée de plusieurs éléments de recherche comme on le voit figure 16. Il est aussi possible à l'utilisateur de rentrer des mots clés.

FIG. 6.3 – Interface de résultats

La deuxième étape de la recherche concerne l'accès à l'information et l'affichage des résultats. La requête est analysée comme nous l'avons détaillé dans la section 3.1 : une partie de la requête contient les noms propres et accède à des tables de hachage pour une recherche booléenne et l'autre partie de la requête contient les mots clés restants. Ces derniers permettent de transformer la requête en un vecteur multidimensionnel en utilisant le modèle de représentation vectorielle décrit dans la section 6.1. Le vecteur requête est alors "élargi" en une requête par

fenêtrage qui accède à la structure d'indexation pour retourner les résultats les plus similaires.

Comme le montre la figure 6.3, les résultats retournés par notre algorithme de recherche sont affichés avec les informations nécessaires pour identifier chaque séquence vidéo dans la base. Nous avons par exemple le titre de la cassette, l'emplacement (Timecode) du début de la séquence dans la cassette, sa durée, une image clé qui est aussi un lien vers un extrait de la séquence vidéo numérisée et le document Mpeg-7 décrivant la séquence video. Nous proposons aussi un indice de confiance basé sur la pertinence du résultats par rapport à la requête, ce qui nous permet d'ordonner les résultats présentés. Cet indice de pertinence est calculé en fonction de

- la distance entre la requête transformée en vecteur, et l'ensemble des vecteurs résultats représentant les séquences vidéo retourné par la requête par fenêtrage ;
- l'intersection possible avec les résultats retournés par la recherche booléenne sur l'ensemble des noms propres stockés dans des tables de hachage.

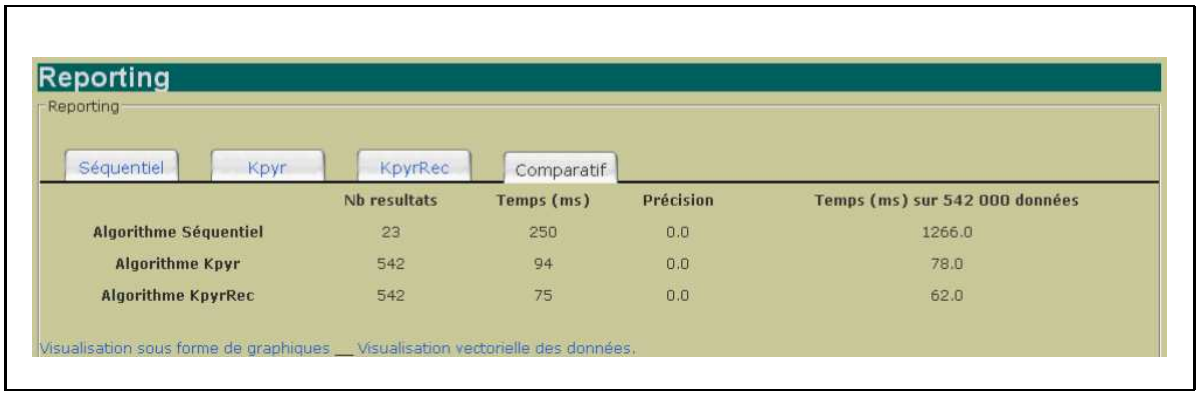

FIG. 6.4 – Comparaison des différentes méthodes

Enfin, un compte rendu des différentes méthodes utilisées pour répondre à la requête est présenté (voir figure 6.4). Il contient le nombre de séquences vidéo retournées par la requête ainsi que le temps de réponse pour chacune des méthodes utilisées. Ce compte rendu permet d'observer que nos structures d'indexation sont de même qualité que la recherche séquentielle en un temps de réponse meilleur. Cette partie du moteur de recherche présente un lien vers un module de visualisation décrit dans la section suivante.

6.3 Module de visualisation

Parallèlement à la recherche, nous avons développé des outils de visualisation. Ces outils nous permettent de visualiser les vecteurs représentant les séquences vidéo de notre base de données. La requête transformée en vecteur est elle aussi affichée. Ces outils de visualisation sont très utiles pour comprendre la répartition des données dans l'espace. La figure 6.5 montre la visualisation des coordonnées parallèles de l'ensemble des données, la figure 6.6 nous propose seulement la visualisation des données appartenant à une classe précise. Tandis que la figure 6.7 visualise l'ensemble des résultats retournés pour une requête par fenêtrage donnée, cette dernière est représentée sur les trois figures par les 2 traits noirs.

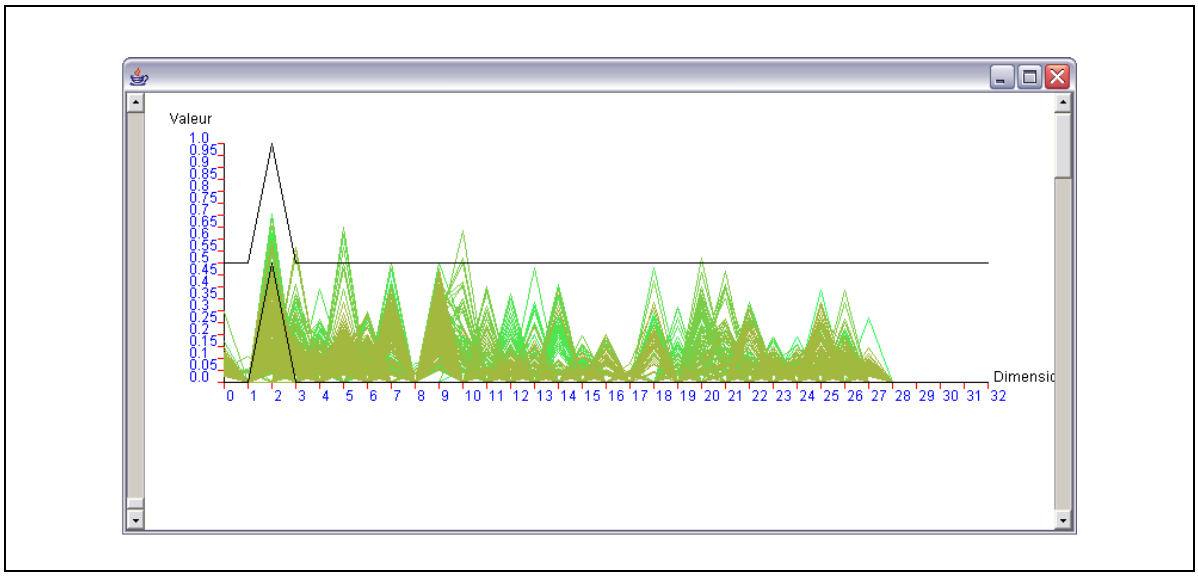

FIG. 6.5 – Visualisation des coordonnées parallèles

Une autre fonctionnalité intéressante de notre outil est la visualisation selon deux dimensions choisies. La figure 6.8 montre l'ensemble des données selon les catégories "Humain" et "Nature". On visualise alors facilement les données à l'intérieur de la requête. L'ensemble des points contenus dans le carré noir ne sont pas forcément des points solutions car ils n'appartiennent probablement pas à la requête selon 2 autres dimensions, voir figure 6.9.

L'ensemble des outils de visualisation permet d'évaluer la pertinence des réponses retour-

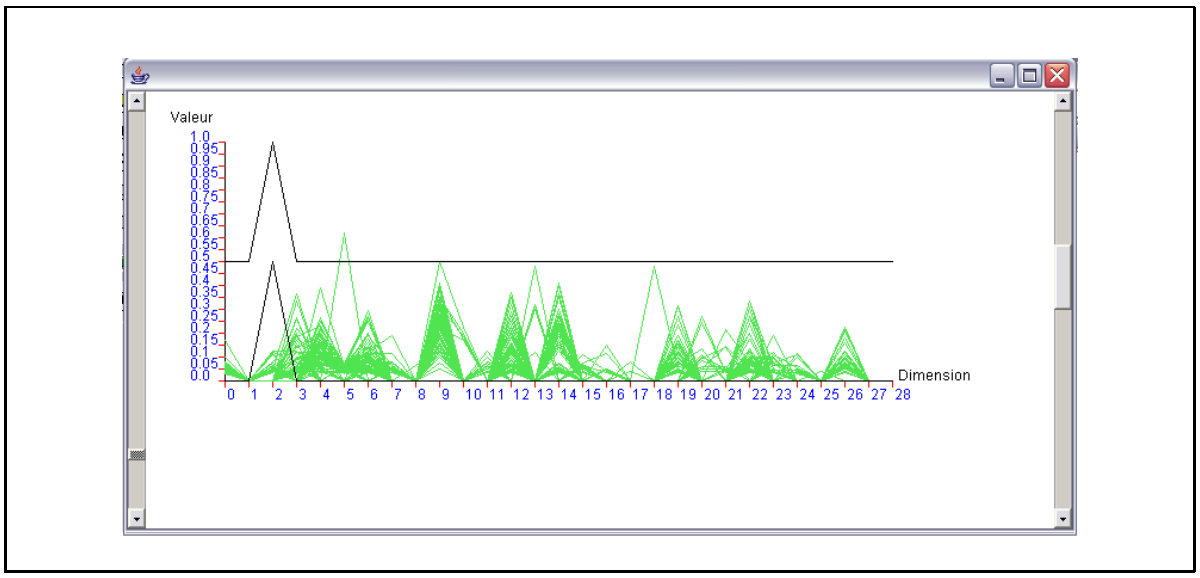

FIG. 6.6 – Visualisation des coordonnées parallèles d'une classe

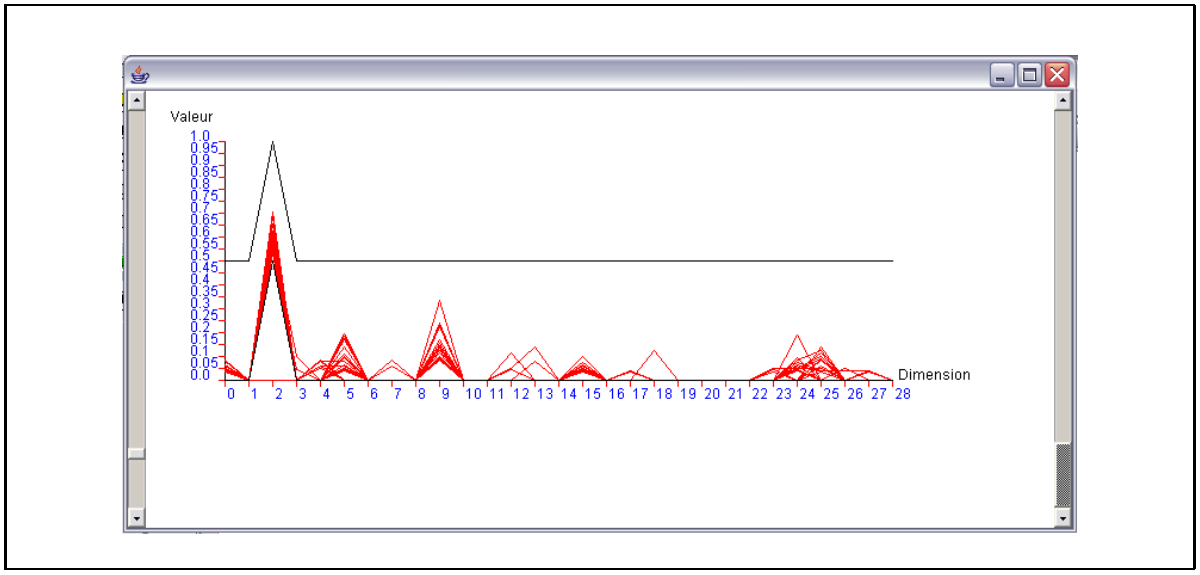

FIG. 6.7 – Visualisation des coordonnées parallèles des résultats

nées :

– la visualisation de l'ensemble des données nous permet de connaître la répartition de

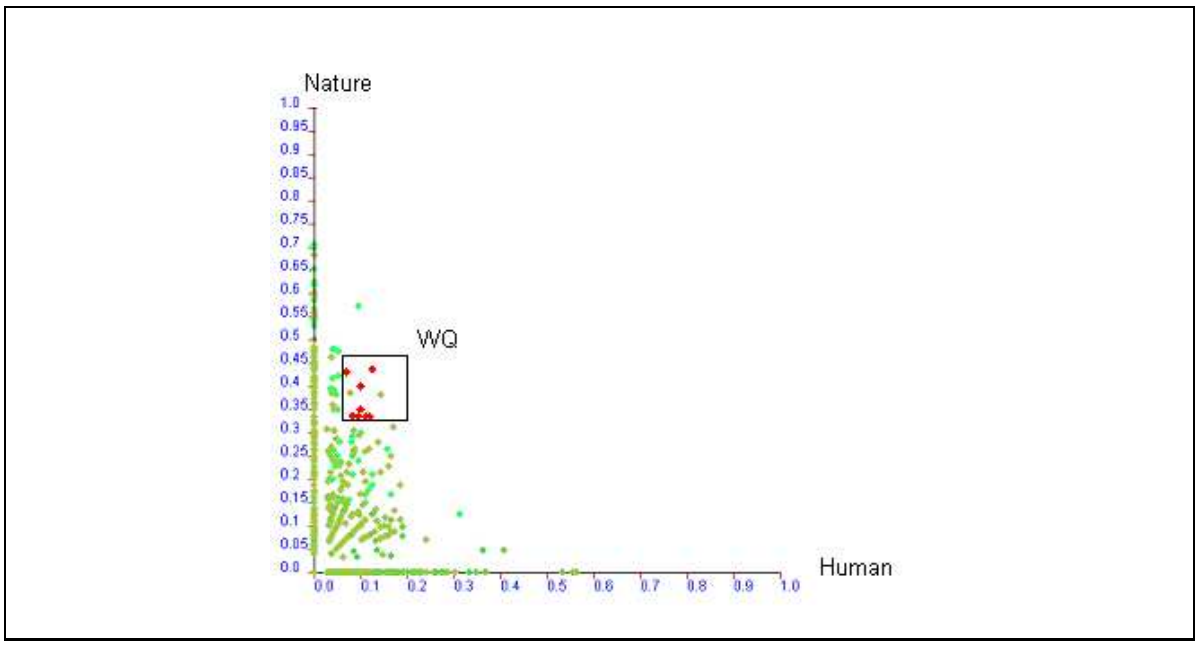

FIG. 6.8 – Visualisation des données selon les dimensions "Nature" et "Humain"

toutes les données, ou simplement d'une classe, dans tout l'espace et par dimension. Cela nous permet de revoir soit le nombre de classes, soit notre modèle de représentation vectorielle ;

– la visualisation du positionnement des résultats par rapport à la requête par fenêtrage donne un aperçu à priori des résultats. Cela nous permet de mieux choisir la sélectivité des requêtes pour de futures requêtes.

Les outils de visualisation mis en place ont bien contribué à la validation des résultats, et ce, en permettant de vérifier visuellement la proximité des vecteurs de la base de données aux différentes requêtes utilisées.

6.4 Amélioration de l'indexation par retours utilisateur

Nous avons présenté et détaillé dans les chapitres précédents les différents points de notre approche pour la gestion d'une base de données de films d'entreprise. Cette gestion se divise en

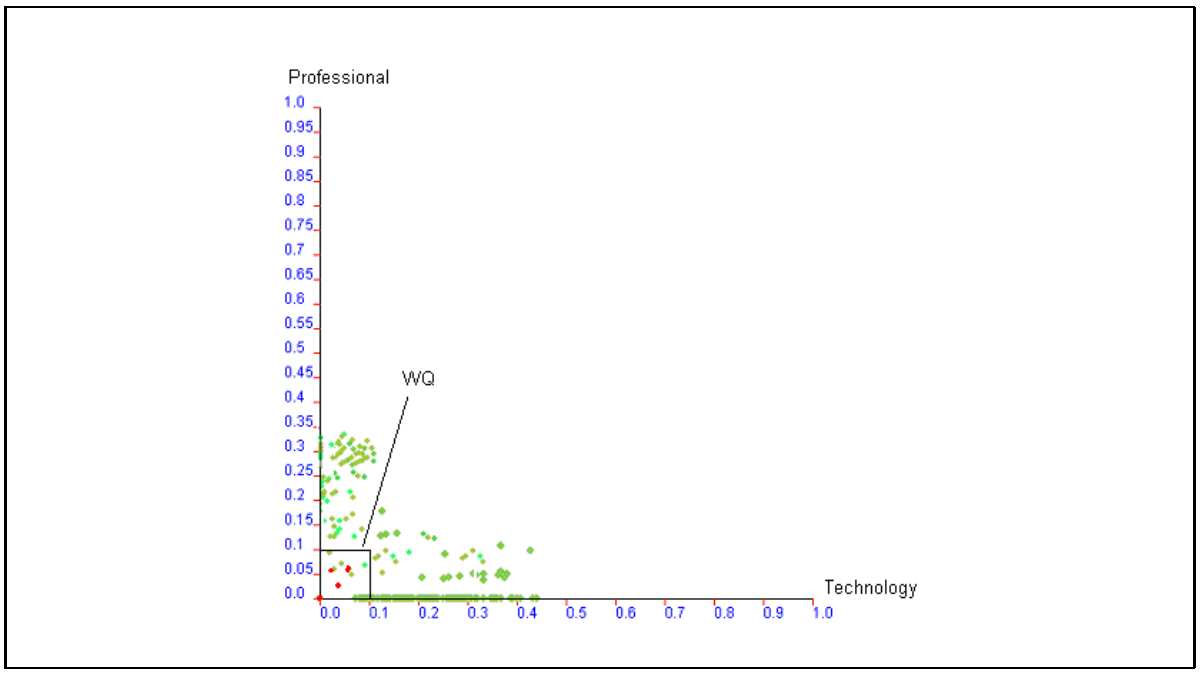

FIG. 6.9 – Visualisation des données selon les dimensions "Professionnel" et "Technologie"

trois grandes parties : l'annotation manuelle par un expert métier de l'ensemble des séquences vidéo, l'indexation de ces informations dans une structure optimisée et le module de recherche permettant un accès efficace aux contenus des séquences vidéo.

Cependant, des erreurs d'indexation peuvent exister que ce soit par une erreur humaine lors de l'annotation d'une séquence vidéo ou dans la création du thésaurus. Par exemple, en attribuant un mot clé non pertinent à une séquence vidéo ou encore par l'utilisation d'un thésaurus erroné pour le modèle de représentation vectorielle, des séquences vidéo sans intérêt peuvent être retournées en tant que résultat pour une recherche précise.

Pour vérifier la qualité des descriptions et de l'indexation, nous proposons d'utiliser une méthode de correction des erreurs d'indexation en étudiant le comportement des utilisateurs sur notre moteur de recherche vidéo. Une analyse parallèle nous permet de modifier les résultats présentés à l'utilisateur sous forme d'une pondération de l'indice de confiance ou encore d'un ajout de résultats à la recherche.

Par exemple, les requêtes utilisant les mots clés *rue* et *monument* présentent à l'utilisateur

10 séquences vidéo résultat. En moyenne, 9 séquences vidéo ont été visionnées en grande partie, et une se trouve fermée (très) rapidement. Notre algorithme propose donc une correction à l'expert métier : une erreur d'indexation se situe sur la séquence vidéo λ liée à l'ensemble de mots clés $\{rue, monument\}$. L'expert pourra alors manuellement vérifier l'annotation de cette séquence pour la corriger si nécessaire. Un autre exemple de requête retourne à l'utilisateur un ensemble de séquences vidéo résultat contenant 7 des 9 séquences vidéo "correctes" de la requête exemple précédente. Notre algorithme peut alors proposer les 2 séquences vidéo restantes car elles sont souvent regardées en même temps que les 7 autres, ainsi que pondérer les valeurs de confiance attribuées aux séquences par la recherche selon les comportements type des utilisateur enregistrés.

Nous proposons donc dans cette section une méthode de correction et d'amélioration de l'indexation par retours utilisateur. Nous expliquons dans un premier temps la mise en place d'une telle méthode dans notre approche globale. Puis, nous évoquons ensuite rapidement la méthode d'analyse des comportements utilisateur utilisée. Ensuite, nous présentons les deux méthodes mises mises en place. Enfin, nous présenterons quelques résultats expérimentaux de corrections et d'améliorations de l'indexation.

6.4.1 Intégration de l'analyse des comportements utilisateur

L'amélioration de l'indexation par retours utilisateur nécessite la mise en place d'un module de récupération des journaux utilisateur dans le moteur de recherche vidéo. Les journaux enregistrées par l'interface de requête décrite dans la section 6.2 ne sont pas assez nombreuses. En effet, cette interface permet seulement d'enregistrer les différents éléments de la requête formulée par l'utilisateur ainsi que les résultats qui lui ont été présentés. Ces journaux ne contiennent aucune donnée relative au comportement de l'utilisateur. Ils ne permettent pas de déterminer si les résultats sont pertinents ou non.

Nous proposons dans notre interface de résultats un lien vers les séquences vidéo retournées par la recherche. L'ensemble des comportements utilisateur sur les vidéos est conservé, ces actions (Play, Pause, Stop, Jump, Forward, Rewind) sont des informations complémentaires à celles contenues par la requête de l'utilisateur. Elles permettent une analyse appro-

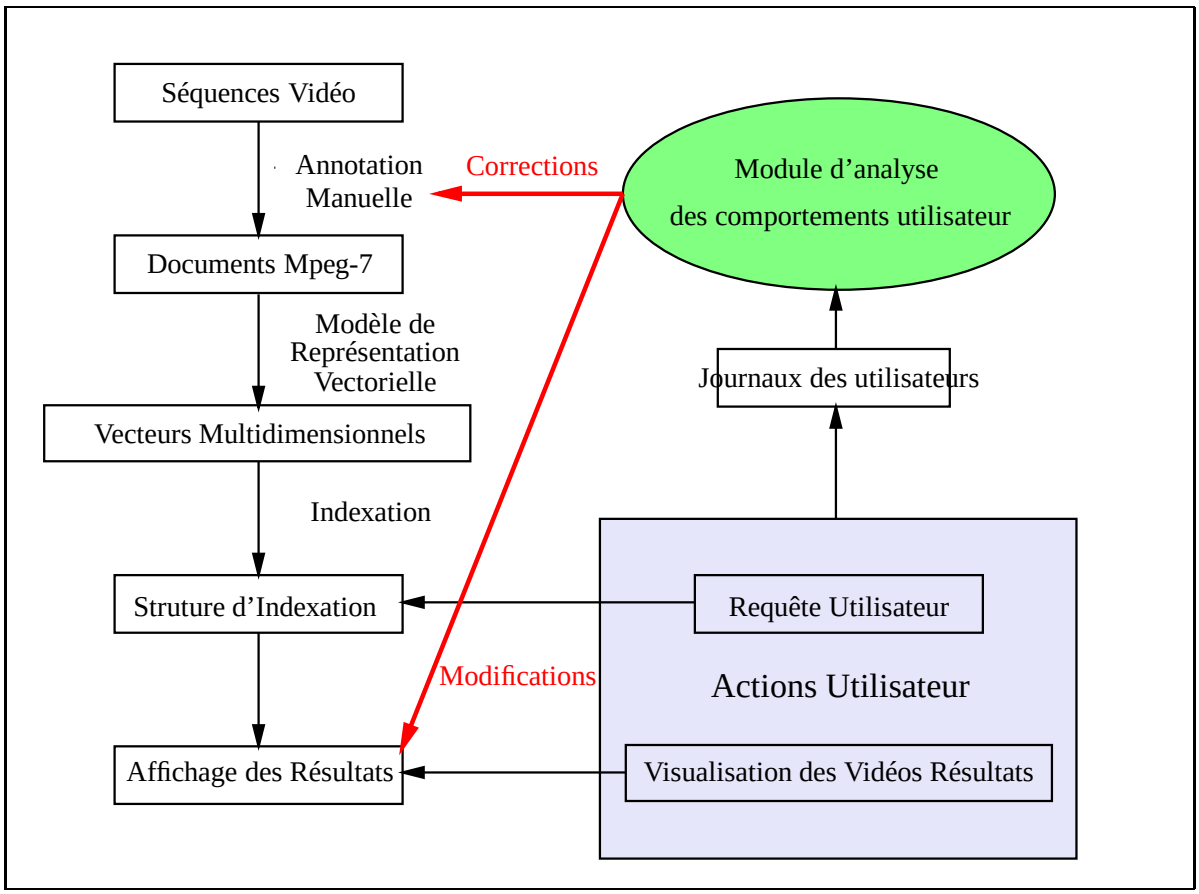

FIG. 6.10 – Récupération et analyse des journaux des utilisateurs pour la correction de l'indexation et la pondération des résultats

fondie pour déterminer l'existence possible d'erreurs d'indexation. La figure 6.10 montre les actions de l'utilisateur que nous conservons dans une base de journaux utilisateur. Ces journaux sont ensuite analysées par notre module d'analyse de comportements utilisateur. Un exemple des deux types de journaux est visible sur la figure 6.11, nous utilisons le langage XML et une DTD adaptée pour représenter nos journaux. La requête et les actions sur les vidéos sont conservées séparément. Notre méthode d'analyse des comportements utilisateur combine les informations contenues dans les différents journaux pour détecter les erreurs potentielles d'indexation à effectuer directement sur les descriptions des séquences vidéo ou dans le thésaurus

utilisé pour le modèle de représentation vectorielle. Une pondération dans l'indice de confiance des résultats proposés à l'utilisateur est également possible avec notre méthode.

```
<Journal>
   <Requête idreq="req01" Idsession="s101">
      <MotsClés>
         <Catégorie>Milieu Naturel</Catégorie>
         <MotClé>Lac</MotClé>
         <Rang>1</Rang>
      </MotsClés>
      <MotsClés>
         <Catégorie></Catégorie>
         <MotClé></MotClé>
         <Rang>2</Rang>
      </MotsClés>
      ...
      <Séquences>
         <Séquence id="seq103" />
         <Séquence id="seq95" />
         <Séquence id="seq53" />
          ...
      </Séquences>
   </Requête>
   <Requête idreq="req02" Idsession="s101">
   ...
</Journal>
```

```
<JournalActions>
   <Action idreq="req01" idseq="seq103">
      <Type>Play</Type>
      <Temps>1108948520</Temps>
      <Durée>5</Durée>
      <Début>0</Début>
      <Fin>5</Fin>
   </Action>
   <Action idreq="req01" idseq="seq103">
      <Type>Jump</Type>
      <Temps>1108948525</Temps>
      <Durée>1</Durée>
      <Début>5</Début>
      <Fin>9</Fin>
   </Action>
   ...
   <Action idreq="req01" idseq="seq53">
      <Type>Pause</Type>
      <Temps>1108948630</Temps>
      <Durée>10</Durée>
      <Début>7</Début>
      <Fin>7</Fin>
   </Action>
   ...
</JournalActions>
```

FIG. 6.11 – Exemple de journal des actions utilisateur

L'exemple présenté sur la figure 6.11 montre une requête contenant plusieurs éléments. Une requête a un identifiant unique et contient l'identifiant de session de l'utilisateur. Un élément d'une requête est composé d'une catégorie, du mot clé et du rang du mot clé. Cette requête est associée à un ensemble de séquences vidéo visionnées par l'utilisateur parmi les résultats proposés. Le deuxième fichier contient l'ensemble des actions réalisées pendant le visionnage des séquences vidéo. Une action se compose de son type (Play, Pause, Stop, Rewind, Forward, Jump), de l'heure d'exécution de l'action, de sa durée en secondes dans le temps, et des repères de début et fin de l'action dans la séquence vidéo.

6.4.2 Correction de l'indexation et pondération des résultats grâce à l'analyse des comportements utilisateur

Avant de présenter les grandes lignes de l'analyse des comportements utilisateur sur des vidéo et de proposer ensuite nos approches pour la correction de l'indexation et la pondération des résultats, nous posons quelques définitions.

6.4.2.1 Analyse des comportements utilisateur

Nous utilisons, dans nos algorithmes, les informations extraites du journal des requêtes décrit dans la section précédente. Nous définissons l'identifiant de session $Id_{session}$ pour l'ensemble des requêtes, identifiées par un Id_{req} unique, faites par un seul utilisateur. Cet identifiant de session est lié à l'ensemble MC des r mots clés m_i, avec $MC = \{m_1, \ldots, m_r\}$. Pour chaque requête Id_{req_j} d'une session, un ensemble de résultats R est proposé. Chaque résultat $r_i \in R$ représente une séquence vidéo s_i sur laquelle l'utilisateur a la possibilité de faire un ensemble d'actions enregistrées dans les journaux vidéo.

L'ensemble des informations contenues dans les journaux vidéo sert à une analyse intra vidéo des comportements utilisateur. Cet algorithme intra-vidéo, détaillé dans [MD07], permet de découvrir les comportements type des utilisateurs sur une séquence vidéo. Ces comportements type sont représentés par des modèles de Markov (voir fig 6.12). Les transitions entre chaque état représentent la probabilité de passer d'un état à un autre à chaque seconde.

Cette analyse du comportement de l'utilisateur nous permet d'obtenir comme résultats intéressants pour nos besoins. Nous proposons deux algorithmes d'amélioration du moteur de recherche : une correction de l'indexation et une modification des résultats présentés à l'utilisateur par une pondération de l'indice de confiance ou un ajout d'autres résultats.

6.4.2.2 Correction de l'indexation

Nous proposons ici un algorithme de correction de l'indexation. Le but est de proposer à l'expert métier une liste de corrections possibles détectées automatiquement par l'algorithme. La figure 6.13 présente schématiquement les étapes de l'algorithme avec un exemple précis. A

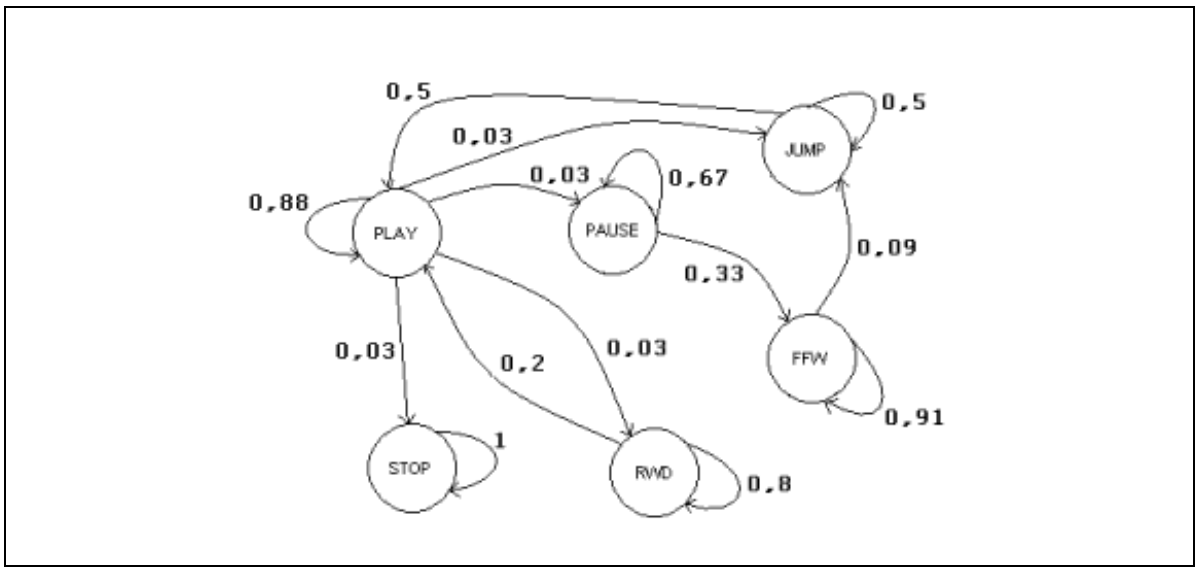

FIG. 6.12 – Exemple de comportement type

partir des résultats de l'analyse du comportement des utilisateurs, nous récupérons l'ensemble des comportements observés pour chaque mot clé. Trois niveaux de comportements sont définis :

- la vidéo a été visionnée en majorité par l'ensemble des utilisateurs, nous notons ce comportement $+$;
- la vidéo a été partiellement visionnée, ce qui correspond à des lectures partielles de la vidéo, nous notons ce comportement $=$;
- la vidéo n'a pas été visionnée (fermeture rapide) par les utilisateurs, nous notons ce comportement $-$.

A partir de ces ensembles $E_m = (id_video, comportement)$, nous détectons les séquences vidéo très peu visionnées. Par exemple, pour le mot $m23$, la séquence vidéo $v3$ comporte un comportement type $-$. Deux possibilités :

- soit ce problème détecté est généralisé à l'ensemble des requêtes, alors il est nécessaire de s'attarder à la séquence vidéo elle même. Il est possible que cette dernière soit de mauvaise qualité ou ne présente réellement aucun intérêt quel que soit la requête.
- soit ce problème est isolé, alors il est possible que le mot $m23$ ne soit pas adapté à la

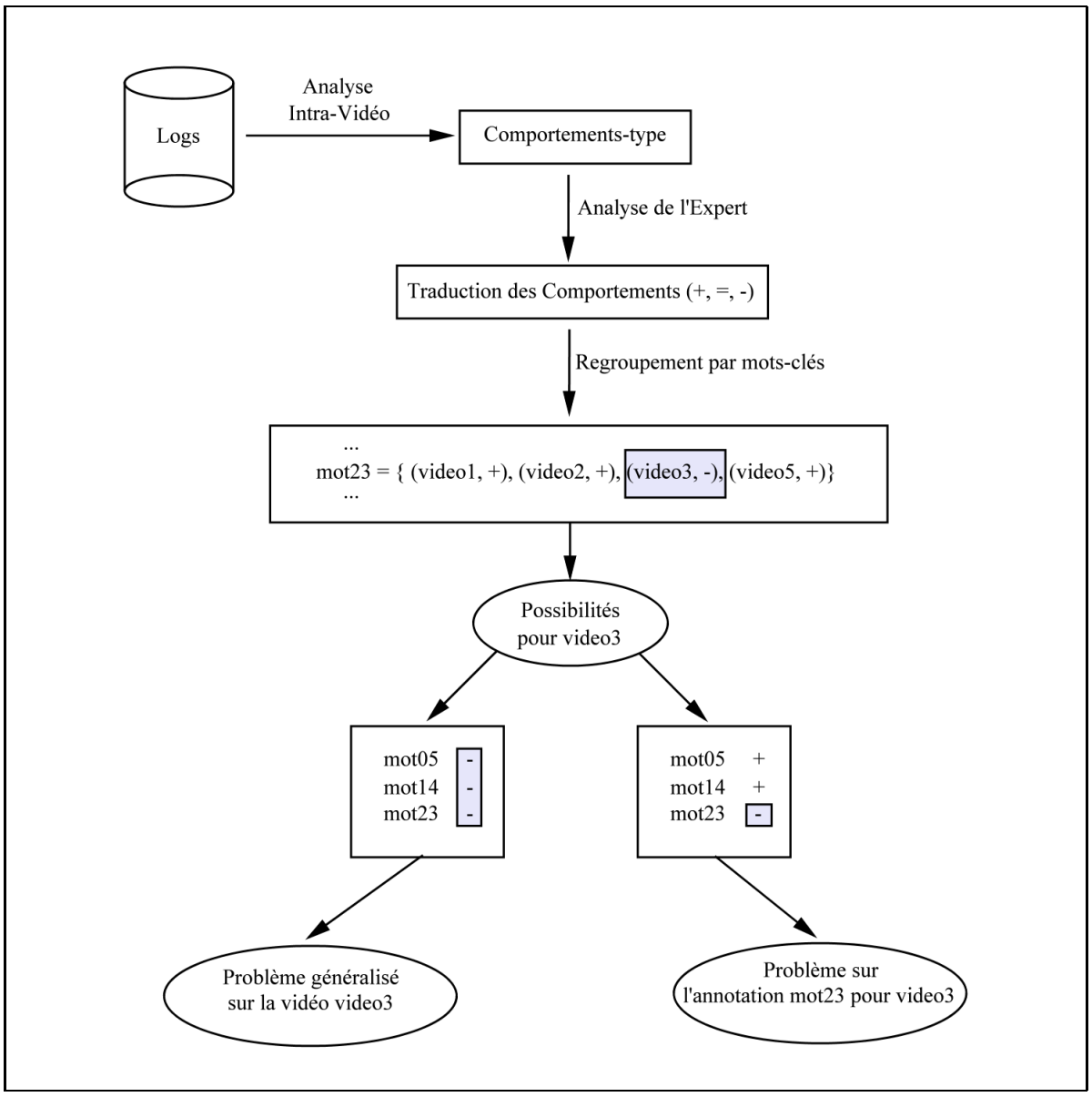

FIG. 6.13 – Exemple de propositions de corrections

description de la séquence vidéo.

Quelle que soit la possibilité choisie, notre algorithme propose son analyse à l'expert métier. Ce dernier pourra alors faire les corrections nécessaires s'il les considère pertinentes.

6.4.2.3 Modification des résultats

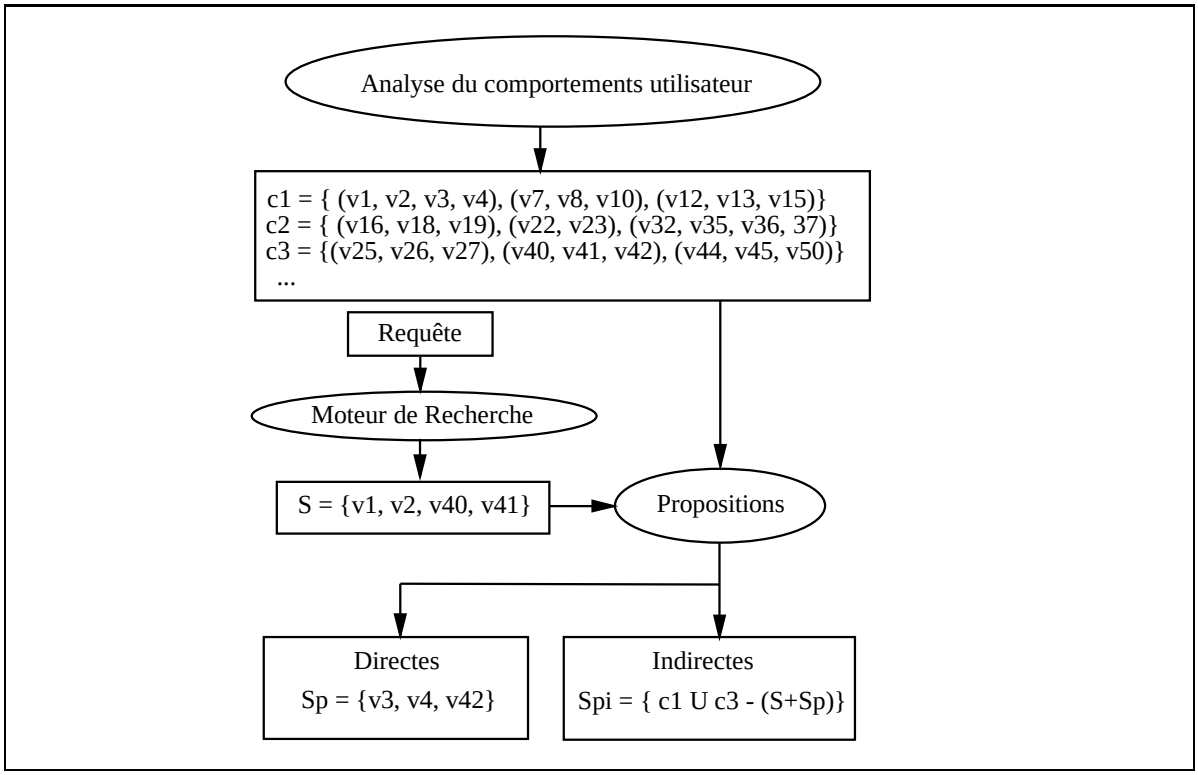

FIG. 6.14 – Exemple de modifications des résultats

Une analyse inter vidéo à partir des journaux d'actions et des comportements type des utilisateurs nous permet de proposer un module de modification des résultats. Comme le montre la figure 6.14, notre module peut ajouter des propositions directes et indirectes. A partir des suites de séquences vidéo fréquentes obtenues, nous complétons automatique l'ensemble des résultats S de la recherche avec l'ensemble Sp contenant les séquences vidéo appartenant à la même suite de séquences vidéo fréquentes. Par exemple, la recherche retourne comme solutions les séquences vidéo $v1$ et $v2$, ces vidéos appartiennent à une suite fréquente séquences vidéo $(v1, v2, v3, v4)$, il est alors naturel de faire une proposition directe à l'utilisateur en ajoutant les séquences $v3$ et $v4$ à la suite de l'ensemble S.

La proposition indirecte propose un ajout de solutions à l'ensemble S provenant de la clas-

sification des plus longues séquences obtenues par l'analyse des comportements. Il est possible de proposer à l'utilisateur avec un indice de pondération plus faible, un ensemble *Spi* contenant les suites de séquences vidéo appartenant aux mêmes classes que les séquences vidéo de *S* et *Sp*. Cet algorithme de modification des résultats permet de proposer un ensemble de séquences vidéo pertinentes en plus de celles obtenues par une recherche normale.

Une autre possibilité de modification des résultats, non représentée sur la figure, utilise les valeurs de pourcentage de visionnage obtenues par l'analyse des comportements utilisateur pour pondérer l'indice de confiance proposé par la recherche. Par exemple, deux séquences vidéo $v1$ et $v2$ sont classées comme solutions à une recherche avec des indices de confiance de 90% et 70% respectivement. Or, ces deux vidéos ont des pourcentages de visionnage très différents : $v1$, peu souvent regardée entièrement, a un comportement type $=$ et v2, très pertinente à la recherche, a un comportement type $+$. Il est alors possible de pondérer l'indice de confiance donné par la recherche par le pourcentage de visionnage. Par exemple, l'ordre des résultats peut être modifié en plaçant $v2$ avec 80% et $v1$ avec 70% pour indice de confiance. Cette modification des résultats permet d'obtenir un ensemble de résultats *S*, inchangé, mais classé de manière plus pertinente.

6.4.3 Résultats expérimentaux

6.4.3.1 Correction de mots clés non pertinents

Afin de tester la qualité de notre méthode, nous avons simulé une base de données de 50 séquences vidéo. Chaque séquence vidéo a été annotée par 5 mots clés. Sur ces 50 séquences vidéo :

- 34 séquences vidéo sont correctement indexées. Nous simulons alors des journaux d'actions correspondant à des comportements type ($+$ ou $=$) d'utilisateurs visionnant entièrement ou partiellement la séquence quel que soit le mot clé utilisé pour la requête.
- 11 séquences vidéo sont sans intérêt pour les utilisateurs. Les journaux simulées correspondent à des comportements type ($-$) pour tous les mots clés des séquences vidéo.
- 7 séquences vidéo ont un seul mot clé qui ne correspond pas à son contenu. Seules les actions enregistrées pour ce mot clé erroné contiennent des comportements type $-$, pour

les autres mots clés, les actions enregistrées sont de type$+$.

Une base de 10000 actions d'utilisateurs est alors créée à partir des 50 vidéos et des comportements type avec une répartition aléatoire pour chaque vidéo. A partir de cette base de données, l'algorithme d'analyse des comportements permet d'obtenir un ensemble de comportements type. Chaque comportement type est alors associé à un comportement type $+$, $=$ ou $-$. Nous obtenons alors un ensemble de couples $(motsCles,\ comportement)$. Une simple analyse de chaque vidéo et ses 5 mots clés permet de retourner à l'expert la liste des vidéos comportant des problèmes.

Nous pouvons alors construire le tableau de contingence présenté figure 6.1 avec les résultats obtenus.

Vrais positifs VP	bonnes vidéos détectées comme bonnes	34
Faux positifs FP	vidéos à problème détectées comme bonnes	4
Vrais négatifs VN	vidéos à problème détectées comme à problème	14
Faux négatifs FN	bonnes vidéos détectées comme à problème	0

TAB. 6.1 – Définition de notre tableau de contingence

Ce tableau permet d'évaluer la précision, le rappel et la mesure F_1 de notre algorithme.

$$\text{Précision} = \frac{\text{VP}}{\text{VP}+\text{FP}} = 0,895$$

,

$$\text{Rappel} = \frac{\text{VP}}{\text{VP}+\text{FN}} = 1$$

et

$$F_1 = \frac{2 \times \text{Rappel} \times \text{Précision}}{\text{Rappel}+\text{Précision}} = 0,945$$

Les expérimentations réalisées présentent de bons résultats sur une base de données simulées. Nous observons que certaines vidéos à problème ne sont pas détectées, cependant nous

obtenons un rappel parfait ce qui montre que notre algorithme ne détecte aucune bonne vidéo comme à problème. Ces résultats montrent donc un réel apport pour l'analyse de l'indexation d'une base de vidéos conservant les journaux des comportements des utilisateurs. Nos propositions de corrections permettent de réduire le travail de l'expert de la base en se concentrant seulement sur les vidéos à problème. La mise en place de cet algorithme sur notre moteur de recherche vidéo est une partie de nos futurs travaux afin de le tester sur une base de données réelles.

6.5 Conclusion

Ce chapitre présente les implémentations réalisées pour tester nos méthodes d'indexation sur un moteur de recherche de séquences vidéo. Nous avons présenté le modèle de représentation vectoriel adapté à notre base de données de documents Mpeg-7 afin de transformer les descriptions des séquences vidéo en vecteurs multidimensionnel. Nous avons détaillé les grandes parties de notre moteur de recherche permettant la comparaison sur des données réelles des structures d'indexation que nous avons développées. Enfin nous proposons une méthode de correction d'indexation à l'aide des comportements utilisateur sur notre moteur de recherche et sur la visualisation des séquences vidéo.

Bibliographie

[MD07] Sylvain Mongy and Chabane Djeraba. A study on video viewing behavior. application to movie trailer miner. In *Proceedings of the IS&T/SPIE 19th Annual Symposium Electronic Imaging Science and Technology (SPIE 2007*, 2007.

[Sal89] Gerard Salton. *Automatic Text Processing : The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[SM84] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, 1984.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11) :613–620, 1975.

[vR79] C.J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

Chapitre 7

Conclusion

Ce mémoire de thèse présente le travail réalisé sur l'indexation optimisée d'une base de descripteurs Mpeg-7 représentant la description d'une base de séquences vidéo de films d'entreprise. Pour résoudre cette problématique, nous avons axé nos travaux de recherche dans plusieurs domaines. La première solution que nous avons proposée est d'utiliser une structure d'indexation optimisée pour une base de données multidimensionnelles. La deuxième solution toujours en cours d'élaboration est de regrouper les documents XML en classes structurelles. Nous résumons dans la section suivante les résultats obtenus pour ces deux solutions. Nous présentons ensuite nos perspectives de recherche à la suite de ces années de thèse.

7.1 Résultats de nos travaux de recherche

Dans cette première section de la conclusion de ce mémoire de recherche, nous proposons de synthétiser l'ensemble des travaux présentés. Nous discutons tout d'abord de nos méthodes d'indexation multidimensionnelle ainsi que de leurs performances. Nous soulignons ensuite l'importance d'une technique de regroupement adaptée aux données multidimensionnelles. Enfin nous donnons les avantages et inconvénients de telles structures.

Les structures d'indexation multidimensionnelle que nous avons proposées sont basées sur une intégration d'une méthode de regroupement à une structure d'indexation. *Kpyr* présente l'association de *K-Means* à la *technique de la pyramide*. L'utilisation d'une technique de re-

groupement telle que K-Means permet d'obtenir des classes de données dont la répartition est plus homogène que l'espace originel. La technique de la pyramide peut alors être utilisée dans de meilleures conditions afin d'obtenir de bonnes performances, confirmées par nos expérimentations. Néanmoins, certaines de nos expérimentations ont soulevé le problème du nombre d'accès aux données trop élevé pour une simple requête. Le temps de réponse étant étroitement lié au nombre d'accès, nous avons focalisé nos recherches vers une diminution de ce problème. Nous avons alors proposé *KpyrRec*, une structure d'indexation basée sur Kpyr dans laquelle nous avons remplacé la technique de la pyramide par une technique d'indexation récursive. Dans KpyrRec, les index des données représentent une plus petite région de l'espace ce qui permet d'accéder plus précisément aux bonnes données lors d'une requête. Les résultats obtenus ont montré clairement une forte diminution des accès aux données, ce qui, par conséquent, a amélioré le temps de réponse par rapport à *Kpyr*.

Cependant, un deuxième problème a été constaté : pour certaines répartitions de données, les résultats étaient aléatoires. Pour cela, nous avons proposé une méthode de regroupement par projections aléatoires adaptée à des données de dimension élevée et dont la répartition des données dans l'espace n'affecte pas la qualité des classes obtenues. Nous avons prouvé que la complexité de notre méthode était au maximum de $O(N\log(N))$ et que le nombre de classes était déterminé automatiquement. Nous avons démontré avec une série de tests que l'association de notre technique de regroupement et de notre structure d'indexation multidimensionnelle récursive permet à *RPyR*, notre dernière structure d'indexation d'être adaptée à des bases de données multidimensionnelles et dont la répartition des données n'a pas d'influence sur les performances.

L'application de notre structure d'indexation à une base de données réelles de documents Mpeg-7 décrivant des séquences vidéo permet de comprendre les avantages et limites de ces structures d'indexation. Tout d'abord, l'utilisation d'un modèle de représentation vectoriel est nécessaire pour transformer les données sémantiques en vecteurs numériques multidimensionnels, mais cela engendre une perte de précision pour les futures requêtes. Ensuite, l'utilisation d'une structure d'indexation requiert un minimum de temps pour accéder à la structure d'indexation en dehors des accès aux données solutions de la requête. Ce temps devient négligeable pour une base de données volumineuse par rapport au temps de recherche. Cependant,

le volume de données et la dimension élevée des données sont des paramètres qui influent rapidement sur le temps de recherche d'où l'importance d'une structure d'indexation optimisée.

7.2 Perspectives

Nos travaux de recherche sur l'accès direct aux contenus des documents XML n'en sont qu'à leur commencement. La recherche d'information et l'utilisation d'un langage de requêtes adapté à notre application, font parties de nos perspectives de recherche. La comparaison entre l'utilisation d'un modèle de représentation vectoriel lié à une structure d'indexation sur des vecteurs et l'accès direct aux contenus de documents XML est l'une des perspectives majeures de ce travail de thèse. Nous nous intéresserons plus particulièrement au compromis entre le temps de réponse et qualité de recherche.

Nous dirigerons nos expérimentations sur de grandes bases de données multimédia pour notre proposition de structure d'indexation multidimensionnelle, en prenant en compte des descriptions automatiques du contenu vidéo en plus des descriptions manuelles. L'ajout des descripteurs de contenus automatiques de vidéo nous permettra d'augmenter nettement la dimension de nos données, nous pourrons alors valider les performances de nos structures sur des données réelles de très grande dimension.

Notre algorithme de regroupement de données par projections aléatoires a montré un fort potentiel applicatif dans des applications multimédia. Nous espérons pouvoir le tester dans d'autres domaines applicatifs pour valider son adaptabilité à des bases de données hétérogènes réelles. Une amélioration de l'algorithme est prévue au niveau de la complexité asymptotique de ce dernier. Nous espérons pouvoir nous rapprocher d'une complexité de $0(N)$.

Publications

Revue Internationale avec comité de programme et actes

[ULIS07] Thierry Urruty, Stanislas Lew, Nacim Ihadaddene and Dan A. Simovici. Detecting Eye Fixations by Projection Clustering. *To appear in ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*. ACM Press. 2007.

[Urr07] Thierry Urruty. KpyrRec : a Recursive Multidimensional Indexing Structure. *To appear in International Journal of Parallel, Emergent and Distributed Systems Journal*. Taylor-Francis. 2007.

Chapitre de livre avec comité de programme et actes

[UBDB06] Thierry Urruty, Fatima Belkouch, Chabane Djeraba, Bruno Bachimont, Edouard Gerard, Jean De Bissy, Olivier Lombard and Patrick Alléaume. Optimization of video content descriptions for retrieval. *Chapter of book, Encyclopedia of Multimedia*, Editor : Borko Furht, p673-681, Springer, ISBN 038724395X. February 2006.

Conférences Internationales avec comité de programme et actes

[UDS07] Thierry Urruty, Chabane Djeraba and Dan A. Simovici. Clustering by Random Projection. *7th Industrial Conference on Data Mining*, Lectures Notes in Artificial Intelligence (LNAI), p107-119, Springer, ISBN 3-540-73434-1, Leipzig, Germany, 14-18th July 2007.

[UBD06a] Thierry Urruty, Fatima Belkouch and Chabane Djeraba. Efficient Indexing for High Dimensional Data : Applications to a Video Search Tool. *12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM KDD 2006)*, ISBN 1-59593-339-5, Philadelphia, USA , 20-23 august 2006.

[UBD05a] Thierry Urruty, Fatima Belkouch and Chabane Djeraba. Kpyr : an efficient indexing method. *Proc. Of IEEE International Conference on Multimedia & Expo (IEEE ICME 2005)*, p1448 - 1451, ISBN 0-7803-9332-5, Amsterdam, Netherland, 6-8 July 2005.

Workshops Internationaux avec comité de programme et actes

[ULDS07] Thierry Urruty, Stanislas Lew, Chabane Djeraba and Dan Simovici. Detecting Eye Fixations by Projection Clustering. *Workshop on Video and Multimedia Digital Library (VMDL'07) in the 14th Int. Conf. on Image Analysis and Processing*, p45-50, ISBN 0-7695-2921-6, Modena, Italy, 10th september 2007.

[UDS06] Thierry Urruty, Chabane Djeraba and Dan Simovici. Clustering by Random Projections : Application to Image Segmentation. *7th ACM International Workshop on Multimedia Data Mining "Merging Multimedia and Data Mining Research" (ACM KDD/MDM 2006)*, p119-124, Philadelphia, USA, 20 august 2006.

Conférences Nationales avec comité de programme et actes

[UBDS07] Thierry Urruty, Fatima Belkouch, Chabane Djeraba and Dan A. Simovici. RPyR : Nouvelle Structure d'Indexation avec Classification par Projections Aléatoires. *23ème journées Informatique des Organisation et Systèmes d'Information et de Décision (INFORSID-2007)*, p242-258, Hermès Ed., ISBN 978-2-9527630-1, Perros Guirec, France, 22 au 25 Mai 2007.

[UBD06b] Thierry Urruty, Fatima Belkouch and Chabane Djeraba. Indexation Multidimensionnelle : KpyrRec, une amélioration de Kpyr. *22ème journées Informatique des Organisation et Systèmes d'Information et de Décision (INFORSID-2006)*, p831-846, Hermès Ed., ISBN 2-906855-22-7, Hammamet, Tunisie, 1er au 3 Juin 2006.

[UBD05b] Thierry Urruty, Fatima Belkouch and Chabane Djeraba. Kpyr, une structure efficace d'indexation de documents vidéo. *21ème journées Informatique des Organisation et Systèmes d'Information et de Décision (INFORSID-2005)*, p403-418, Hermès Ed., ISBN 2-906855-21-9, Grenoble, France, 24 au 27 Mai 2005.

Bibliographie

[Abd07] Hervé Abdi. Distance. In Neil J. Salkind, editor, *Encyclopedia of Measurement and Statistics*, pages 280–284. Sage, 2007.

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics : Ordering points to identify the clustering structure. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 49–60, Philadelphia, Pennsylvania,USA, 1999. ACM Press.

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM-SIGMOD Int. Conf. Management of Data,*, pages 94–105, 1998.

[AGQ06] S. Ayache, J. Gensel, and G. M. Quénot. Clips-lsr experiments at trecvid 2006. In *TRECVID'2006 Workshop*, Gaithersburg, MD, USA, 2006.

[AI] LSH Algorithm and Implementation. http ://web.mit.edu/andoni/www/LSH/index.html.

[AM04] P. Agarwal and N. H. Mustafa. k-means projective clustering. In *Proceedings of PODS*, pages 155–165, 2004.

[APW+99a] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 61–72, 1999.

[APW+99b] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 61–72, Philadelphia, Pennsylvania,USA, 1999. ACM Press.

[AV99] R. I. Arriaga and S. Vempala. An algorithmic theory of learning : Robust concepts and random projection. In *IEEE Symposium on Foundations of Computer Science*, pages 616–623, 1999.

[AY00a] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 70–81, Dallas, Texas, USA, 2000. ACM.

[AY00b] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of ACM-SIGMOD Intl. Conference on Management of Data*, pages 70–81, 2000.

[BBJ+00] S. Berchtold, C. Böhm, H. V. Jagadish, H. P. Kriegel, and J. Sander. Independant quantization : An index compression technique for high-dimensional data spaces. In *Proceedings of 16-th Int. Conf. on Data Engineering, IEEE ICDE*, pages 577–588, San diego, California, USA, 2000.

[BBK98] S. Berchtold, C. Bohm, and H. P. Kriegel. The pyramid technique : Towards breaking the curse of dimensionality. In *Proceedings of ACM SIGMOD Int. Conf on Management of Data*, pages 142–153, Seatle,Washington,USA, 1998.

[Ben79] J. L. Bentley. Multidimensional binary search trees in database applications. *Proceedings of IEEE Transactions on Software Engineering*, SE-5 :333–340, 1979.

[Ber02] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA, 2002.

[Ber04] Sid-Ahmed Berrani. Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu., 2004.

[BHR00] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *SPIRE*, pages 39–48, 2000.

[BK73] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4) :230–236, 1973.

[BKK96] S. Berchtold, D.A. Keim, and H.P. Kriegel. The x-tree : An index structure for high-dimensional data. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 322–331, Bombay, Inde, 1996. Morgan Kaufmann.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The r*-tree : An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD International conference on Management of Data*, pages 322–331, Atlantic City, NJ, USA, 1990. ACM Press.

[BL95] J. P. Barthélemy and B. Leclerc. The median procedure for partitions. In *Partitioning Data Sets*, pages 3–14, Providence, RI, 1995. American Mathematical Society.

[BM72] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1 :173–189, 1972.

[BYRN99] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[CC02] Guang-Ho Cha and Chin-Wan Chung. The gc-tree : a high-dimensional index structure for similarity search in image databases. *IEEE Transactions on Multimedia*, 4(2) :235–247, 2002.

[CCS04] Gerardo Canfora, Luigi Cerulo, and Rita Scognamiglio. Measuring xml document similarity : A case study for evaluating information extraction systems. In *10th IEEE International Software Metrics Symposium (METRICS 2004)*, pages 36–45, Chicago, IL, USA, 2004. IEEE Computer Society.

[CFZ99] C. H. Cheng, A. W.-C. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *5th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 84–93. ACM, 1999.

[CIEI+07] CLIPS-IMAG, EURECOM, INA, IRIT, LABRI, LIP6, NOVELTIS, and SFRS. Campagne d'évaluation d'outils de surveillance de contenus vidéo, 2007. http ://www.irit.fr/argos.

[CJ02] J.-W. Chang and D.-S. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC)*, pages 503–507, Madrid, Spain, 2002. ACM.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree : An efficient access method for similarity search in metric spaces. In *Proceedings of 23rd International Conference on Very Large Data Bases*, pages 426–435, Athenes, Grece, 1997. Morgan Kaufmann.

[CUDW02] A. B. Chaudri, R. Unland, C. Djeraba, and W.Lindner, editors. *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops*, volume LNCS 2490 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2002.

[CZ03] S.-C. S. Cheung and A. Zakhor. Efficient video similarity measurement with video signature. *IEEE Trans. Circuits Syst. Video Techn.*, 13(1) :59–74, 2003.

[CZPC02] G.-H. Cha, X. Zhu, P. Petkovic, and C.-W. Chung. An efficient indexing method for nearest neighbor searches in high-dimensional image databases. *IEEE Transactions on Multimedia*, 4(1) :76–87, 2002.

[Dai] Dailymotion. http ://www.dailymotion.com/fr/.

[DFG99] Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors. *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*. ACM Press, 1999.

[DG99] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. Technical Report TR-99-006, International Computer Science Institute, 1999.

[Did73] Edwin Diday. The dynamic clusters method and optimization in non-hierarchical clustering. In *5th Conference on Optimization Techniques*, volume 3 of *Lecture Notes in Computer Science*, pages 241–258, Rome, Italy, 1973. Springer.

[DIIM04] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In J. Snoeyink and J. Boissonnat, editors, *Proceedings of the 20th ACM Symposium on Computational Geometry*, pages 253–262, Brooklyn, New York, USA, 2004. ACM.

[Dje03] C. Djeraba, editor. *Multimedia Mining - A Highway to Intelligent Multimedia Documents*. Kluwer, Boston, 2003.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the *em* algorithm. *Journal of the Royal Statistical Society*, 39(1) :1–38, 1977.

[DPGM04] C. Domeniconi, D. Papadopoulos, D. Gunopulos, and S. Ma. Subspace clustering of high dimensional data. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, Lake Buena Vista, Florida, USA, 2004. SIAM.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, 1996.

[FM88] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *J. Comb. Theory B*, 44 :355–362, 1988.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2) :170–231, 1998.

[GHJ91] A. Guénoche, P. Hansen, and B. Jaumard. Efficient algorithms for divisive hierarchical clustering with the diameter criterion. *Journal of classification*, 8(1) :5–30, 1991.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, pages 518–529, Edinburgh, Scottland, UK, 1999. Morgan Kaufmann.

[GLC02] K. Goh, B. Li, and E. Chang. Dyndex : a dynamic and non-metric space indexer. In *MULTIMEDIA '02 : Proceedings of the tenth ACM international conference on Multimedia*, pages 466–475, Juan-les-Pins, France, 2002. ACM Press.

[GRS98] S. Guha, R. Rastogi, and K. Shim. Cure : An efficient clustering algorithm for large databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 73–84, Seattle, Washington, USA, 1998. ACM Press.

[GRS00] S. Guha, R. Rastogi, and K. Shim. Rock : A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5) :345–366, 2000.

[Gut84] A. Guttman. R-trees : a dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, USA, 1984. ACM Press.

[Hen98] A. Henrich. The lsd^h-tree : An access structure for feature vectors. In *Proceedings of the Fourteenth International Conference on Data Engineering (ICDE)*, pages 362–369, Orlando, Florida, USA, 1998. IEEE Computer Society.

[HK98] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 58–65, 1998.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer. The lsd tree : Spatial access to multidimensional point and nonpoint objects. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 45–53, Amsterdam, Pays-Bas, 1989. Morgan Kaufmann.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors : Towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[IMD] IMDB. http ://www.imdb.com/.

[JBPKQ07] Philippe Joly, Jenny Benois-Pineau, Ewa Kijak, and Georges Quénot. The argos campaign : Evaluation of video analysis and indexing tools. *Image Commun.*, 22(7-8) :705–717, 2007.

[JD88] A. K. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[JF96] A. K. Jain and P. J. Flynn. Image segmentation using clustering. In *Advances in Image Understanding : A Festschrift for Azriel Rosenfeld*, pages 65–83, Piscataway, NJ, 1996. IEEE Press.

[JL84] W. B. Johnson and J. Lindenstrauss. Extensions of lipshitz mappings into hilbert spaces. *Contemporary Mathematics*, 26 :189–206, 1984.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : A review. *ACM Computing Surveys*, 31 :264–323, 1999.

[Jol06] Alexis Joly. Recherche par similarité statistique dans une grande base de signatures locales pour l'identification rapide d'extraits vidéo., 2006. Doctorat de l'université de La Rochelle.

[KHK99] G. Karypis, E.-H. Han, and V. Kumar. Chameleon : Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8) :68–75, 1999.

[KKK04] Peer Kröger, Hans-Peter Kriegel, and Karin Kailing. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, Lake Buena Vista, Florida, USA, 2004. SIAM.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data : An Introduction to Cluster Analysis.* John Wiley, 1990.

[KS97] N. Katayama and S. Satoh. The sr-tree : An index structure for high-dimensional nearest neighbor queries. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 369–380, Tucson,Arizona,USA, 1997. ACM Press.

[KV05] C. Kim and B. Vasudev. Spatiotemporal sequence matching for efficient video copy detection. *IEEE Transaction on Circuits and Systems for Video Technology*, 15(1) :127–132, 2005.

[LJW+06] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. A time-space efficient locality sensitive hashing method for similarity search in high dimensions. Technical report, Princeton University, New Jersey, USA, 2006.

[LOH05] J. Lee, J.-H. Oh, and S. Hwang. Strg-index : Spatio-temporal region graph indexing for large video databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 718–729, Baltimore, Maryland, USA, 2005.

[LSG01] M.S. Lew, N. Sebe, and P. Gardner. Video indexing and understanding, 2001.

[LXY00] B. Liu, Y. Xia, and P. S. Yu. Clustering through decision tree construction. In *9th international conference on Information and knowledge management*, pages 20–29. ACM, 2000.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, 1967. University of California Press.

[Mar04] José M. Martínez. Mpeg-7 overview, 2004. http ://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

[MCD+06] Josiane Mothe, Claude Chrisment, Taoufiq Dkaki, Bernard Dousset, and Saïd Karouach. Combining mining and visualization tools to discover the geographic structure of a domain. *Computers, Environment and Urban Systems*, 30(4) :460–484, 2006.

[MD07] Sylvain Mongy and Chabane Djeraba. A study on video viewing behavior. application to movie trailer miner. In *Proceedings of the IS&T/SPIE 19th Annual Symposium Electronic Imaging Science and Technology (SPIE 2007*, 2007.

[MF02] Oge Marques and Borko Furht. *Content-Based Image and Video Retrieval*. Springer, 2002.

[MLBM05] N. Mo
ënne-Loccoz, E. Bruno, and S. Marchand Maillet. Interactive retrieval of video

sequences from local feature dynamics. In *Proceedings of the 3rd International Workshop on Adaptive Multimedia Retrieval, AMR
05, Glasgow, UK, July 2005.

[Moh98] Rakesh Mohan. Video sequence matching. In *Int. Conf. on Audio, Speech and Signal Processing (ICASSP)*, volume 6, pages 3697–3700, 1998.

[Mon] P. Mondrian. http ://artchive.com/artchive/M/mondrian.html.

[Mpe] Mpeg7. http ://www.chiariglione.org/mpeg/.

[NGC99] H. Nagesh, S. Goil, and A. Choudhary. Mafia : Efficient and scalable subspace clustering for very large data sets. Technical report, Northwestern University, 1999.

[NH94] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases*, pages 144–155, Santiago, Chile, 1994. Morgan Kaufmann Publishers.

[OT02] B. C. Ooi and K.-L. Tan. B-trees : Bearing fruits of all kinds. In Xiaofang Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, volume 5, Melbourne, Victoria, Australie, 2002. Australian Computer Society.

[OTYB00] B. C. Ooi, K. L. Tan, C. Yu, and S. Bressan. Indexing the edges - a simple and yet efficient approach to high-dimensional. In *Proceedings of 19-th ACM SIGMOD SIGACT SIGART Symposium on Principles of Database Systems*, pages 166–174, Dallas, Texas, USA, 2000.

[PHL04] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data : a review. *SIGKDD Explorations*, 6(1) :90–105, 2004.

[PJAM02] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 418–427, Madison, Wisconsin, USA, 2002. ACM.

[PM00] D. Pelleg and A. W. Moore. X-means : Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International*

Conference on Machine Learning (ICML 2000), pages 727–734, Stanford University, Standord,CA, USA, 2000. Morgan Kaufmann.

[PPC06] V. Parshin, A. Paradzinets, and L. Chen. Multimodal data fusion for video scene segmentation. In *Visual Information and Information Systems, 8th International Conference, VISUAL 2005*, volume 3736 of *Lecture Notes in Computer Science*, Amsterdam, The Netherlands, 2006. Springer.

[RHC99] Y. Rui, T. Huang, and S. Chang. Image retrieval : current techniques, promising directions and open issues. *Journal of Visual Communication and Image Representation*, 10(4) :39–62, april 1999.

[Rij79] C.J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.

[Rob81] J. T. Robinson. The k-d-b-tree : A search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, Michigan, USA, 1981. ACM Press.

[Sal89] Gerard Salton. *Automatic Text Processing : The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[Sau05] Karen Sauvagnat. Modèle flexible pour la recherche d'information dans des corpus de documents semi-structurés., 2005. Doctorat de l'Université Paul Sabatier de Toulouse.

[SBBG02] S. M. Savaresi, D. Boley, S. Bittanti, and G. Gazzaniga. Cluster selection in divisive clustering algorithms. In *Proceedings of the Second SIAM International Conference on Data Mining*, Arlington, VA, USA, 2002. SIAM.

[SBC04] Karen Sauvagnat, Mohand Boughanem, and Claude Chrisment. Searching xml documents using relevance propagation. In *String Processing and Information Retrieval, 11th International Conference, SPIRE 2004*, volume 3246 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004.

[SEKX98] J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Density-based clustering in spatial databases : The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery, an International Journal*, 2 :169–194, 1998.

[SM84] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.

[SOZ05] H. T. Shen, B. C. Ooi, and X. Zhou. Towards effective indexing for very large video sequence database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 730–741, Baltimore, Maryland, USA, 2005. ACM.

[SPK07] I. Song, J. Paik, and U. Kim. Semantic-based similarity computation for xml document. *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, 0 :796–803, 2007.

[SS01] Philippe Salembier and John R. Smith. Mpeg-7 multimedia description schemes. *IEEE Trans. Circuits Syst. Video Techn.*, 11(6) :748–759, 2001.

[SWS+00] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12) :1349–1380, 2000.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11) :613–620, 1975.

[SYUK00] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The a-tree : An index structure for high-dimensional spaces using relative approximation. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 516–526. Morgan Kaufmann, 2000.

[Tek] Kardi Teknomo. http ://www.people.revoledu.com/kardi/tutorial/Similarity/.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[TSK06] P. N ; Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson/Addison-Wesley, Boston, 2006.

[Tub] You Tube. http ://www.youtube.com/.

[TZKO06] A. K. H. Tung, R. Zhang, N. Koudas, and B.C. Ooi. Similarity search : A matching based approach. In *Proceedings of the 32nd International Conference on*

Very Large Data Bases, Seoul, Korea, September 12-15, 2006, pages 631–642. ACM press, 2006.

[UAH76] J. D. Ullman, A. V. Aho, and D. S. Hirschberg. Bounds on the complexity of the longest common subsequence problem. *J. ACM*, 23(1) :1–12, 1976.

[UBD05] T. Urruty, F. Belkouch, and C. Djeraba. Kpyr, une structure efficace d'indexation de documents vidéo. In *21ème journées Informatique des Organisation et Systèmes d'Information et de Décision (Inforsid05)*, pages 403–418, Grenoble, France, 2005.

[UBD06] T. Urruty, F. Belkouch, and C. Djeraba. Indexation multidimensionnelle : Kpyrrec, une amélioration de kpyr. In *22ème journées Informatique des Organisation et Systèmes d'Information et de Décision (Inforsid06)*, pages 831–846, Hammamet, Tunisie, 2006.

[UDS06] T. Urruty, C. Djeraba, and D. Simovici. Clustering by random projections : Application to image segmentation. In *Proceedings of 7-th Int. Workshop on Multimedia Data Mining "Merging Multimedia and Data Mining Research" (ACM KDD/MDM 2006*, pages 119–124, Philadelphia, USA, 2006.

[Vem04] S. S. Vempala. *The Random Projection Method*. American Mathematical Society, Providence, Rhode Island, 2004.

[W3C] W3C. http ://www.w3.org/.

[WD94] S. WAllace and D. L. Dowe. Intrinsic classification by mml - the snob program. In *the 7 Australian Joint Conference on Artificial Intelligence*, pages 37–44, Singapour, 1994.

[WF74] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1) :168–173, 1974.

[WF05] I. H. Witten and E. Frank. *Data Mining - Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, second edition, 2005.

[WJ96] D. A. White and R. Jain. Similarity indexing with the ss-tree. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, pages 516–523, New Orleans, Louisiana, USA, 1996. IEEE Computer Society.

[WLKL04] K.-G. Woo, J.-H. Lee, M.-H. Kim, and Y.-J. Lee. Findit : a fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology*, 46(4) :255–271, March 2004.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of 24rd International Conference on Very Large Data Bases*, pages 194–205, New York city, New York, USA, 1998. Morgan Kaufmann.

[YCN04] K. Y. Yip, D. W. Cheung, and M. K. Ng. Harp : A practical projected clustering algorithm. *IEEE Transactions on Knowledge Data Engineering*, 16(11) :1387–1397, 2004.

[YL99] Y. Yang and X. Liu. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49, Berkley,USA, August 1999.

[YOTJ01] C. Yu, B. C. Ooi, K. L. Tan, and H. V. Jagadish. Indexing the distance : an efficient method to knn processing. In *Proceedings of 27-th Int. Conf. on Very Large Data Bases*, pages 421–430, Rome,Italy, 2001.

[ZAF+03] X. Zhu, W. G. Aref, J. Fan, A. C. Catlin, and A. K. Elmagarmid. Medical video mining for efficient database indexing, management and access. In *Proceedings of the 19th International Conference on Data Engineering*, pages 569–580. IEEE Computer Society, 2003.

[ZDC06] B. Zhang, W. Dou, and L. Chen. Combining short and long term audio features for tv sports highlight detection. In *Advances in Information Retrieval, 28th European Conference on IR Research, ECIR 2006*, Lecture Notes in Computer Science, pages 472–475, London, UK, 2006. Springer.

[ZEX+05] X. Zhu, A. K. Elmagarmid, X. Xue, L. Wu, and A. C. Catlin. Insightvideo : toward hierarchical video content organization for efficient browsing, summarization and retrieval. *IEEE Transactions on Multimedia*, 7(4) :648–666, 2005.

[ZLCZ02] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity metric for xml documents. In *Workshop on Knowledge and Experience Management, FGWM 2003*, Karlsruhe, Germany, 2002.

[ZOT04] R. Zhang, B. C. Ooi, and K. L. Tan. Making the pyramid technique robust to query types and workloads. In *Proceedings of 20-th Int. Conf. on Data Engineering, IEEE ICDE*, pages 313–324, Boston, USA, 2004.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. Birch : An efficient data clustering method for very large databases. In *Proceedings of the International Conference on Management of Data*, pages 103–114, 1996.

[ZRL97] T. Zhang, R. Ramakrishnan, and M. Livny. Birch : A new data clustering algorithm and its applications. *Data Mining Knowledge Discovery*, 1(2) :141–182, 1997.

[ZS89] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6) :1245–1262, 1989.

[ZSD02] O. R. Zaïane, S. Simoff, and C. Djeraba, editors. *Mining Multimedia and Complex Data*, volume LNAI 2797 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 2002.

[ZZS07] X. Zhou, X. Zhou, and H. T. Shen. A new similarity measure for near duplicate video clip detection. In *Advances in Data and Web Management, Joint 9th Asia-Pacific Web Conference, APWeb 2007, and 8th International Conference, on Web-Age Information Management, WAIM 2007*, volume 4505 of *Lecture Notes in Computer Science*, pages 176–187, Huang Shan, China, 2007. Springer.

Annexe A

La norme Mpeg-7

Sommaire

A.1 Descripteurs et schémas de description MPEG-7

MPEG-7 [Mar04] utilise *XML Schéma* pour son langage de définition de description. Les descripteurs sont divisés en deux catégories : les descripteurs concernant l'audio et ceux concernant la vidéo. Parmi les descripteurs concernant l'audio, on trouve ceux qui se rapportent aux effets sonores, plus précisément à l'indexation et la catégorisation des effets sonores. Il existe d'autres descripteurs : par exemple pour le timbre des instruments de musique avec leurs caractéristiques perceptuelles, pour le contenu parlé, ou encore pour les mélodies. Certains de ces descripteurs audio utilisent des outils tels que le « Scale Tree » qui décrit des séries temporelles de descripteurs ou des segments de silence qui attachent une sémantique de silence à un segment temporel.

Mais la majeure partie des descripteurs de MPEG-7 concerne les descripteurs vidéo. Ils sont nombreux et concernent par exemple les caractéristiques de couleur, de texture, de forme et de mouvement. Nous allons décrire brièvement quelques uns de ces descripteurs. Les descripteurs de couleur permettent de choisir l'espace des couleurs, de connaître les couleurs dominantes d'une image ou d'un segment vidéo, la distribution spatiale des couleurs, ou encore l'histogramme d'une image. Les descripteurs de texture servent à décrire l'homogénéité de la texture d'une image et la caractéristique perceptuelle de la texture. Les descripteurs de forme permettent des recherches basées sur les contours, sur les régions et sur les objets 3D. Enfin les descripteurs de mouvement sont utilisés pour les déplacements de caméra, la localisation et le suivi d'un objet sur un segment vidéo.

Tous ces descripteurs servent aux schémas de description multimédia. Pour un survol général de ces schémas de description, nous proposons la figure A.1 tirée de [SS01].

Les éléments de base (partie inférieure du schéma) concernent les constructions fondamentales des documents Mpeg-7, les outils permettant l'utilisation des schémas de description, le lien ou la localisation de segments audiovisuels sur le média utilisé. Cette partie comprend aussi tous les schémas de description de base, par exemple le temps, le lieu, la personne et toutes les annotations textuelles libres ou structurées.

Au dessus, à gauche, se situent les schémas de description du contenu comprenant les informations structurelles (les segments vidéo ou la table des matières du document) liées aux

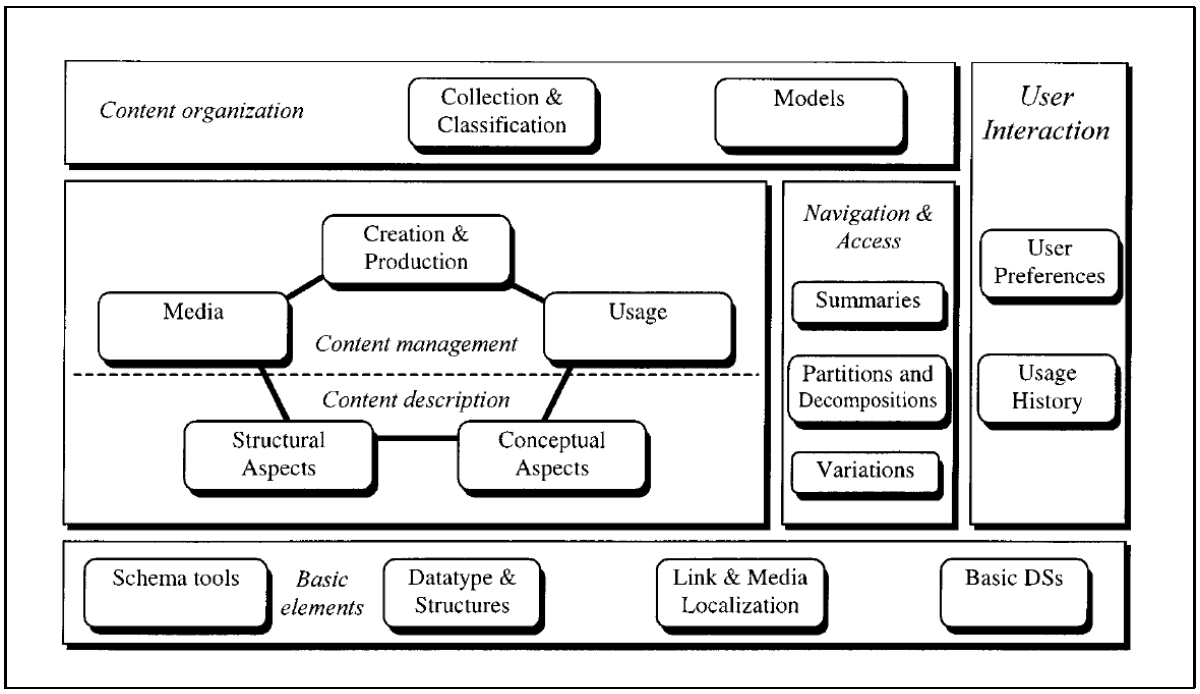

FIG. A.1 – Aperçu des schéma de descriptions de Mpeg-7

descriptions conceptuelles (événements, objets, relations). Cette partie est reliée à la gestion des contenus (juste au dessus sur le schéma) qui est utile pour les informations de création (titre, créateurs, lieu de création, classification,...), pour les informations d'utilisation (droits, disponibilité, ou financier) et pour les informations propres au document multimédia (format d'encodage, paramètre d'encodage, etc.).

A droite se trouve la partie navigation et accès comprenant le résumé des navigations, des accès, et des visualisations. Cette partie décrit les partitions du document, c'est à dire les différentes vues du même document et les variations possibles d'un document (différentes versions selon encodage ou format). En haut se trouve la partie organisation des contenus concernant les collections de documents, la gestion de ces collections ainsi que tous les ensembles non ordonnés de contenus audiovisuels et tout ce qui concerne les modèles, la description des collections ou classes de contenu avec l'utilisation de probabilités. Enfin la dernière partie du schéma décrit l'interaction avec l'utilisateur qui concerne les préférences de chaque utilisateur

ainsi que l'historique.

En résumé MPEG-7 est une spécification de descriptions normalisées du contenu multimédia, représentée sous forme de graphes d'objets hiérarchiques, utilisant le langage XML. La norme MPEG-7 est basée sur la norme MPEG-4. Les descripteurs et schémas de description ont été définis en fonction de MPEG-4 mais tout utilisateur peut créer ses propres descripteurs et schémas de description selon ses besoins. [Mpe] est l'un des sites de référence pour suivre les nouveautés de MPEG-7, et permet de trouver de nombreuses informations supplémentaires.

Bibliographie

[Mar04] José M. Martínez. Mpeg-7 overview, 2004. http ://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

[Mpe] Mpeg7. http ://www.chiariglione.org/mpeg/.

[SS01] Philippe Salembier and John R. Smith. Mpeg-7 multimedia description schemes. *IEEE Trans. Circuits Syst. Video Techn.*, 11(6) :748–759, 2001.